Handbook of Geometric Deep Learning Loss Formulas

Integrating Geometry, Topology, and Differentiable Objectives

Mehdi Gorjian

Abstract

This handbook consolidates the essential mathematical and implementation formulas used across geometric deep learning, differentiable geometry, and shape analysis. It aims to serve as a precise yet practical reference for researchers developing 3D learning systems that integrate classical surface operators with modern loss formulations. The material covers geometric predicates, discrete surface operators, mesh quality priors, geometry-aware loss functions, topology-preserving constraints, and optimization stability techniques.

Keywords: geometric deep learning, differential geometry, mesh processing, loss functions, optimization, topology preservation, differentiable reconstruction.

Contents

1 Introduction

The rapid progress of geometric deep learning has brought together two traditionally distinct domains: classical geometry processing and neural optimization. Researchers now routinely integrate discrete differential geometry, mesh operators, and topological regularizers into the loss functions of neural networks for 3D reconstruction, implicit representation, and simulation tasks.

This handbook unifies mathematical expressions that appear repeatedly across literature—from cotangent Laplacians and curvature operators to Chamfer, Earth Mover's, and as-rigid-as-possible losses—presented in a format suitable for both theoretical derivation and immediate implementation.

2 Geometric Predicates and Distances

Geometric predicates form the foundation of all geometric reasoning and are crucial for stable loss computation in differentiable geometry and implicit field learning.

2.1 Orientation in 2D

For points $a, b, c \in \mathbb{R}^2$:

$$\text{orient2D}(a, b, c) = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x), \tag{1}$$

positive for counterclockwise orientation, negative for clockwise, and zero if collinear. The signed area of $\triangle abc$ is $\frac{1}{2}\,\text{orient2D}(a, b, c)$.

Implementation hint. In Python or C++, this operation can be vectorized over point batches; for robustness, use double precision and add a small ϵ threshold when checking orientation signs.

2.2 Orientation in 3D (Scalar Triple Product)

For vectors $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^3$:

$$[\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}] = \boldsymbol{u} \cdot (\boldsymbol{v} \times \boldsymbol{w}), \tag{2}$$

representing six times the signed volume of the tetrahedron defined by $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$.

Implementation hint. The triple product is implemented efficiently on GPUs via batched cross and dot products. In PyTorch: `torch.einsum('bi,bi->b', torch.cross(v,w,dim=1), u)`.

2.3 Point–Plane Distance and Projection

Given a plane with unit normal $\boldsymbol{n}$ passing through p_0:

$$\boldsymbol{n} \cdot (x - p_0) = 0. \tag{3}$$

For a query point q:

$$d(q, \Pi) = \boldsymbol{n} \cdot (q - p_0), \qquad \text{proj}_\Pi(q) = q - d(q, \Pi)\boldsymbol{n}. \tag{4}$$

Implementation hint. Common in implicit-SDF training loops. Normalize $\boldsymbol{n}$ and compute both d and $\mathrm{proj}_\Pi(q)$ in vectorized form to avoid instability when normals are not unit-length.

2.4 Closest Point on a Segment

Given segment endpoints a, b and a query q, define

$$t^\star = \frac{(q-a)\cdot(b-a)}{\|b-a\|^2}, \qquad \hat{t} = \min(1, \max(0, t^\star)). \tag{5}$$

Then the closest point is

$$\mathrm{cp}_{[a,b]}(q) = a + \hat{t}(b-a). \tag{6}$$

Implementation hint. Essential for point-to-surface distance computation in differentiable Chamfer losses; can be implemented branch-free using clamped interpolation.

2.5 Voronoi Bisector

Between two points p_i, p_j, the bisector satisfies

$$\|x-p_i\|^2 = \|x-p_j\|^2 \quad \Rightarrow \quad x\cdot(p_j-p_i) = \tfrac{1}{2}(\|p_j\|^2 - \|p_i\|^2). \tag{7}$$

Implementation hint. In Voronoi-based learning (e.g., differentiable Voronoi or occupancy partitioning), evaluate this plane implicitly and avoid explicit point-by-point comparison for efficiency.

2.6 Triangle Circumcenter

Let $\boldsymbol{u} = b-a$, $\boldsymbol{v} = c-a$, and $\boldsymbol{w} = \boldsymbol{u}\times\boldsymbol{v}$, then:

$$\mathrm{circ}(a,b,c) = a + \frac{\|\boldsymbol{u}\|^2(\boldsymbol{v}\times\boldsymbol{w}) + \|\boldsymbol{v}\|^2(\boldsymbol{w}\times\boldsymbol{u})}{2\|\boldsymbol{w}\|^2}. \tag{8}$$

Implementation hint. Useful for Delaunay triangulation and medial-axis computation. Ensure $\boldsymbol{w}$ is not degenerate; for nearly flat triangles, regularize by adding a small offset before division.

3 Surface Differential Operators

Discrete differential operators translate smooth geometry into a matrix form suitable for computation on meshes. They are widely used in Laplacian regularization, curvature-based smoothing, and shape optimization.

3.1 Per-Vertex Area (Mixed Voronoi or Barycentric)

The per-vertex area weight is defined as

$$A_i = \sum_{f\in\mathcal{N}_f(i)} A_f^{(i)}, \qquad A_f^{(i)} = \begin{cases} \frac{1}{8}\,\|e_{jk}\|^2\cot\alpha_i, & \text{acute,} \\ \frac{1}{4}A_f, & \text{obtuse at } i, \\ \frac{1}{8}A_f, & \text{obtuse not at } i, \end{cases} \tag{9}$$

where A_f is the triangle area and α_i is the corner angle at vertex i.

Implementation hint. Precompute A_i as vertex weights for Laplacian systems or mean-curvature loss functions; this can be efficiently done with adjacency lists or CSR representations.

3.2 Cotangent Laplace–Beltrami Operator

For a scalar function f defined on mesh vertices:

$$(\Delta f)_i = \frac{1}{2A_i} \sum_{j \in \mathcal{N}(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(f_j - f_i), \tag{10}$$

where α_{ij}, β_{ij} are angles opposite edge (i, j).

Implementation hint. Store cotangent weights as a sparse matrix L; in PyTorch or CUDA, multiplication by L provides a differentiable Laplacian smoothing layer.

3.3 Mean Curvature Normal

The mean curvature normal vector at vertex i:

$$\boldsymbol{H}_i = \frac{1}{2A_i} \sum_{j \in \mathcal{N}(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(\boldsymbol{v}_i - \boldsymbol{v}_j). \tag{11}$$

Implementation hint. This directly corresponds to the discrete Laplacian of vertex positions, i.e., $\boldsymbol{H} = L\boldsymbol{V}$. In neural networks, this can act as a curvature-based regularization loss.

3.4 Gaussian Curvature (Angle Deficit)

At vertex i:

$$K_i = \frac{1}{A_i} \left(2\pi - \sum_{f \in \mathcal{N}_f(i)} \theta_{i,f} \right), \tag{12}$$

where $\theta_{i,f}$ are the incident corner angles.

Implementation hint. For curvature learning or surface classification, precompute $\theta_{i,f}$ using normalized cross products of edge vectors; normalize K_i by mean curvature magnitude for stability.

4 Surface Properties and Integrals

Fundamental surface quantities such as areas, normals, and volumes appear in nearly every geometric learning loss. Accurate computation and differentiable implementation of these quantities are critical.

4.1 Triangle Area and Face Normal

For a triangle (a, b, c):

$$A_f = \frac{1}{2} \|(b-a) \times (c-a)\|, \qquad \hat{n}_f = \frac{(b-a) \times (c-a)}{\|(b-a) \times (c-a)\|}. \tag{13}$$

Implementation hint. In PyTorch, cross products can be batched; store both A_f and $\hat{n}_f$ for use in curvature and normal consistency losses.

4.2 Area-Weighted Vertex Normal

$$\hat{n}_i = \frac{\sum_{f \ni i} A_f \, \hat{n}_f}{\left\| \sum_{f \ni i} A_f \, \hat{n}_f \right\|}. \tag{14}$$

Implementation hint. Use vertex–face adjacency for aggregation. Normalize after summation to avoid drift when using gradient descent.

4.3 Volume of a Closed Triangulated Surface

For oriented faces $(\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2)$:

$$V = \frac{1}{6} \sum_{f \in \mathcal{F}} \boldsymbol{v}_0 \cdot (\boldsymbol{v}_1 \times \boldsymbol{v}_2). \tag{15}$$

Implementation hint. Integrate this as a differentiable volume regularization term; ensure face orientation consistency.

5 Per-Face Gradient of a Linear Function

On a triangle (i, j, k) with scalar values f_i, f_j, f_k:

$$\nabla f|_{(i,j,k)} = \frac{1}{2A_f} \Big(f_i (\boldsymbol{v}_j - \boldsymbol{v}_k)^\perp + f_j (\boldsymbol{v}_k - \boldsymbol{v}_i)^\perp + f_k (\boldsymbol{v}_i - \boldsymbol{v}_j)^\perp \Big), \tag{16}$$

where $(\cdot)^\perp$ denotes a 90° in-plane rotation.

Implementation hint. For diffusion-based regularization, store gradients per face and accumulate at vertices; implement with vectorized matrix multiplication.

6 Mesh Quality and Shape Priors

High-quality geometry priors ensure numerical stability and preserve shape fidelity during optimization.

6.1 Aspect Ratio Metrics

$$r_{\text{in}} = \frac{2A_f}{\|a-b\| + \|b-c\| + \|c-a\|}, \tag{17}$$

$$R_{\text{circ}} = \frac{\|a-b\|\|b-c\|\|c-a\|}{4A_f}, \tag{18}$$

$$\text{AR} = \frac{R_{\text{circ}}}{2r_{\text{in}}}. \tag{19}$$

Implementation hint. Monitor AR during mesh optimization to detect degeneracies.

6.2 Edge-Length Regularity

$$\sigma_l^2 = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} (\|e\| - \bar{l})^2. \tag{20}$$

Implementation hint. Used as an auxiliary loss to keep sampling uniform.

6.3 Laplacian Smoothness

$$E_{\text{smooth}} = \sum_i \left\| \boldsymbol{v}_i - \frac{1}{\deg(i)} \sum_{j \in N(i)} \boldsymbol{v}_j \right\|^2. \tag{21}$$

Implementation hint. Implemented as a sparse-matrix product; often combined with Chamfer loss.

6.4 Normal Consistency

$$E_{\text{normal}} = \sum_{(f,g)} (1 - \hat{n}_f \cdot \hat{n}_g)^2. \tag{22}$$

Implementation hint. Compute for adjacent faces sharing an edge; reduces shading artifacts in reconstructions.

7 Geometry-Aware Losses for Deep Learning

Losses connecting geometry and learning are core components of neural surface reconstruction.

7.1 Chamfer Distance

$$\text{CD}(P,Q) = \frac{1}{|P|} \sum_{p \in P} \min_{q \in Q} \|p-q\|^2 + \frac{1}{|Q|} \sum_{q \in Q} \min_{p \in P} \|q-p\|^2. \tag{23}$$

Implementation hint. Efficiently implemented via KD-tree or approximate nearest neighbor. Gradient-safe and differentiable.

7.2 Earth Mover's Distance

$$\text{EMD}(P,Q) = \min_{\pi \in \Pi} \sum_{m,n} \pi_{mn} \|p_m - q_n\|, \tag{24}$$

where Π denotes valid transport plans.

Implementation hint. Approximated via Sinkhorn iteration for differentiability.

7.3 Point-to-Surface Loss

$$E_{\text{p2s}} = \frac{1}{|Q|} \sum_{q \in Q} \min_{f \in \mathcal{F}} d(q, f)^2. \tag{25}$$

Implementation hint. Integrate distance-to-triangle kernels; requires efficient nearest-face queries (use PyTorch3D or CUDA kernels).

7.4 Laplacian Coordinate Matching

$$E_{\text{lap}} = \sum_i \left\| \sum_{j \in N(i)} w_{ij}(\boldsymbol{v}_i - \boldsymbol{v}_j) - \sum_{j \in N(i)} w_{ij}(\boldsymbol{v}_i^* - \boldsymbol{v}_j^*) \right\|^2. \tag{26}$$

Implementation hint. Encourages structural preservation; implemented by computing the Laplacian once and comparing transformed coordinates.

7.5 ARAP (As-Rigid-As-Possible) Deformation

$$E_{\text{ARAP}} = \sum_{(i,j) \in \mathcal{E}} w_{ij} \left\| (\boldsymbol{v}_i - \boldsymbol{v}_j) - R_i(\boldsymbol{v}_i^0 - \boldsymbol{v}_j^0) \right\|^2, \tag{27}$$

with $R_i \in SO(3)$ local rotations.

Implementation hint. Used in neural deformation models; alternate between estimating R_i and optimizing $\boldsymbol{v}_i$.

7.6 SDF-Based Losses

$$E_{\text{SDF\_data}} = \mathbb{E}_x |s_\theta(x) - \hat{s}(x)|, \tag{28}$$

$$E_{\text{eikonal}} = \mathbb{E}_x (\|\nabla s_\theta(x)\| - 1)^2. \tag{29}$$

Implementation hint. Compute gradients via automatic differentiation; the Eikonal term enforces metric regularity.

7.7 Occupancy BCE Loss

$$E_{\text{BCE}} = -\mathbb{E}_x \left[y(x) \log o_\theta(x) + (1 - y(x)) \log(1 - o_\theta(x)) \right]. \tag{30}$$

Implementation hint. Used for neural implicit fields (Occupancy Networks, DeepSDF variants).

8 Topology-Aware Objectives

Topology preservation ensures geometric consistency in learned surfaces.

8.1 Winding Number Constraint

$$w(x) = \frac{1}{4\pi} \sum_{f \in \mathcal{F}} \Omega_f(x), \quad E_{\text{wn}} = \mathbb{E}_x (w(x) - y(x))^2. \tag{31}$$

Implementation hint. Approximated using solid-angle summation; regularizes closed-surface networks.

8.2 Genus and Manifoldness

$$\chi = |\mathcal{V}| - |\mathcal{E}| + |\mathcal{F}|, \qquad g = 1 - \tfrac{1}{2}\chi. \tag{32}$$

Implementation hint. Penalty $(g - g_0)^2$ enforces desired topology; estimate χ via connectivity queries.

9 Optimization and Numerical Stability

Stable optimization bridges geometry and learning. Numerical care prevents divergence in large-scale 3D training.

- **Normalization.** Center vertices and scale to unit bounding box for consistent gradient magnitudes.
- **Gradient Clipping.** Limit per-vertex updates $\|g_i\| \leq \tau$.
- **Loss Balancing.** Normalize each loss term by its gradient norm to stabilize multi-loss training.
- **Preconditioning.** Add Tikhonov regularization $(L+\lambda I)$ when solving Laplacian systems.
- **Precision.** Use fp16/fp32 mixed precision; clamp divisions by small denominators.
- **Curvature Stability.** Avoid degenerate triangles by remeshing or clamping cotangent weights.
- **Batch Efficiency.** Use sparse storage (CSR/COO) for Laplacian and adjacency matrices.

References

[1] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy. *Polygon Mesh Processing*. AK Peters, 2010.

[2] K. Crane, U. Pinkall, and P. Schröder. Robust Fairing via Conformal Curvature Flow. *ACM Transactions on Graphics (SIGGRAPH)*, 32(4), 2013.

[3] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete Differential-Geometry Operators for Triangulated 2-Manifolds. In *Visualization and Mathematics III*, pages 35–57. Springer, 2003.

[4] R. Bridson. *Fluid Simulation for Computer Graphics*. A K Peters/CRC Press, 2nd edition, 2015.

[5] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian Surface Editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 175–184.

[6] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson Surface Reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, pages 61–70. 2006.

[7] L. Liu, H. Li, Y. Zhang, T. Funkhouser, and S. Rusinkiewicz. Learning to Reconstruct Shapes from Unseen Classes. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 37(6), 2018.

[8] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 38(5), 2019.

[9] Y. Ohtake, A. Belyaev, and H.-P. Seidel. A Multi-Scale Approach to 3D Scattered Data Interpolation with Compactly Supported Basis Functions. In *Shape Modeling International*, pages 153–161. 2003.

[10] M. Desbrun, M. Meyer, P. Schröder, and A. Barr. Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow. In *Proceedings of SIGGRAPH*, pages 317–324, 1999.

[11] Z. Zhang, T. Funkhouser, et al. Learning Mesh-Based SDFs for Geometry Processing. *arXiv preprint arXiv:2006.07852*, 2020.

[12] S. Lefebvre, A. Lagae, et al. Procedural Noise using Sparse Gabor Convolution. *ACM Transactions on Graphics*, 30(4), 2011.

[13] K. Crane, F. de Goes, et al. Digital Geometry Processing with Discrete Exterior Calculus. In *ACM SIGGRAPH Courses*, 2017.

[14] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral Networks and Locally Connected Networks on Graphs. In *ICLR*, 2014.