Geometry Foundations: Surface Representations II

Conversions

Olga Diamanti
Postdoc, Guibas Lab

slides credits: Olga Sorkine-Hornung, Daniele Panozzo, Chengcheng Tang, Justin Solomon
Triangle Meshes

- Connectivity: vertices, edges, triangles
- Geometry: vertex positions

\[
V = \{v_1, \ldots, v_n\} \\
E = \{e_1, \ldots, e_k\}, \quad e_i \in V \times V \\
F = \{f_1, \ldots, f_m\}, \quad f_i \in V \times V \times V \\
P = \{p_1, \ldots, p_n\}, \quad p_i \in \mathbb{R}^3
\]
Data Structures

What should be stored?
- Geometry: 3D coordinates
- Connectivity
- Adjacency relationships
- Attributes
  - Normal, color, texture coordinates
  - Per vertex, face, edge
What should be supported?
- Rendering
- Geometry queries
  - What are the vertices of face #2?
  - Is vertex A adjacent to vertex H?
  - Which faces are adjacent to face #1?
Modifications
- Remove/add a vertex/face
- Vertex split, edge collapse
How good is a data structure?
- Time to construct
- Time to answer a query
- Time to perform an operation
- Space complexity
- Redundancy

Criteria for design
- Expected number of vertices
- Available memory
- Required operations
- Distribution of operations
Simple Data Structures: Indexed Face Set

- **Used in formats**
  - OBJ, OFF, WRL

- **Storage**
  - **Vertex**: position
  - **Face**: vertex indices
  - 12 bytes per vertex
  - 12 bytes per face
  - $36*V$ bytes for the mesh

- **No explicit neighborhood info**
- **Finding neighboring vertices/edges/faces costs** $O(#V)$ time!
Neighborhood Relations

All possible neighborhood relationships:

1. Vertex – Vertex  VV
2. Vertex – Edge    VE
3. Vertex – Face    VF
4. Edge – Vertex    EV
5. Edge – Edge      EE
6. Edge – Face      EF
7. Face – Vertex    FV
8. Face – Edge      FE
9. Face – Face      FF

We’d like $O(1)$ time for queries and local updates of these relationships
Halfedge data structure

An edge-based representation

Why? Edge $\rightarrow$ vertex, edge $\rightarrow$ face is always one-to-two
Halfedge data structure

- Introduce orientation into data structure
- Oriented edges
Halfedge data structure

- Introduce orientation into data structure
  - Oriented edges
- Vertex
  - Position
  - 1 outgoing halfedge index
- Halfedge
  - 1 origin vertex index
  - 1 incident face index
  - 3 next, prev, twin halfedge indices
- Face
  - 1 adjacent halfedge index
- Easy traversal, full connectivity, suitable for non-triangles
Halfedge data structure

One-ring traversal
Start at vertex
Halfedge data structure

One-ring traversal
- Start at vertex
- Outgoing halfedge
Halfedge data structure

One-ring traversal

- Start at vertex
- Outgoing halfedge
- Twin halfedge
Halfedge data structure

- One-ring traversal
  - Start at vertex
  - Outgoing halfedge
  - Twin halfedge
  - Next halfedge
Halfedge data structure

- One-ring traversal
  - Start at vertex
  - Outgoing halfedge
  - Twin halfedge
  - Next halfedge
  - Twin ...

Halfedge data structure

**Pros:** (assuming bounded vertex valence)
- $O(1)$ time for neighborhood relationship queries
- $O(1)$ time and space for local modifications (edge collapse, vertex insertion...)

**Cons:**
- Heavy – requires storing and managing extra pointers
- Invariants need to be maintained!

* For every half-edge $e$:
  - $e = \text{prev}(\text{next}(e)) = \text{next}(\text{prev}(e)) = \text{opposite}(\text{opposite}(e))$
  - $\text{face}(e) = \text{face}(\text{next}(e))$.

* For every vertex $v$:
  - $\text{start}(\text{vertEdge}(v)) = v$.

* For every face $f$:
  - $\text{face}(\text{faceEdge}(f)) = f$. 
Points $\rightarrow$ Implicit
Implicit $\rightarrow$ Mesh
Mesh $\rightarrow$ Points (next time!)

CONVERSIONS
Implicit Surface Reconstruction

POINTS $\rightarrow$ IMPLICIT
Implicit Function Approach

Define a function

\[ f : \mathbb{R}^3 \rightarrow \mathbb{R} \]

with value < 0 outside the shape and > 0 inside
Define a function

\[ f : \mathbb{R}^3 \rightarrow \mathbb{R} \]

with value < 0 outside the shape and > 0 inside

Extract the zero-set

\[ \{ x : f(x) = 0 \} \]
SDF from Points and Normals

- Input: Points + Normals
- Normals help to distinguish between inside and outside
- Computed via locally fitting planes at the points

“Surface reconstruction from unorganized points”, Hoppe et al., ACM SIGGRAPH 1992
Implicit Surfaces

Converting from a point cloud to an implicit surface:

Simplest method:
1. Given a point $x$ in space, find nearest point $p$ in PCD.

2. Set $f(x) = (x - p)^T n_p$ – signed distance to the tangent plane.

“Surface reconstruction from unorganized points”, Hoppe et al., ACM SIGGRAPH 1992
Converting from a point cloud to an implicit surface:

Simplest method:
1. Given a point $x$ in space, find nearest point $p$ in PCD.

2. Set $f(x) = (x - p)^T n_p$ – signed distance to the tangent plane.
Implicit Surfaces

Converting from a point cloud to an implicit surface:

Simplest method:
1. Given a point $x$ in space, find nearest point $p$ in PCD.

2. Set $f(x) = (x - p)^T n_p$ – signed distance to the tangent plane.

“Surface reconstruction from unorganized points”, Hoppe et al., ACM SIGGRAPH 1992
SDF from Points and Normals

- Signed distance to the tangent plane of the closest point

\[ f(x) = (x - p)^T n_p \]

- The implicit function will be discontinuous!

- More advanced methods handle this!

“Reconstruction and representation of 3D objects with radial basis functions”, Carr et al., SIGGRAPH 2001

“Interpolating and Approximating Implicit Surfaces from Polygon Soup”, Shen et al., SIGGRAPH 2004
Reconstruction from points

distance to plane

RBF method
Marching Cubes

**IMPLICIT** → **MESH**
Extracting the Surface

Wish to compute a manifold mesh of the level set

\[ F(x) = 0 \rightarrow \text{surface} \]

\[ F(x) < 0 \rightarrow \text{inside} \]

\[ F(x) > 0 \rightarrow \text{outside} \]
Marching Cubes

Converting from implicit to explicit representations.

Goal: Given an implicit representation: \( \{ \mathbf{x}, \text{s.t.} \ f(\mathbf{x}) = 0 \} \)

Create a triangle mesh that approximates the surface.

Lorensen and Cline, SIGGRAPH ’87

[James Sharman]
Marching Squares (2D)

Given a function: \( f(x) \)

- \( f(x) < 0 \) inside
- \( f(x) > 0 \) outside

1. Discretize space.
2. Evaluate \( f(x) \) on a grid.
Marching Squares (2D)

Given a function: \( f(x) \)

- \( f(x) < 0 \) inside
- \( f(x) > 0 \) outside

1. Discretize space.
2. Evaluate \( f(x) \) on a grid.
3. Classify grid points (+/-)
4. Classify grid edges
5. Compute intersections
Computing the intersections:

- Edges with a sign switch contain intersections.
  \[
  f(x_1) < 0, \ f(x_2) > 0 \Rightarrow \\
  f(x_1 + t(x_2 - x_1)) = 0 \\
  \text{for some } 0 \leq t \leq 1
  \]

- Simplest way to compute \( t \): assume \( f \) is linear between \( x_1 \) and \( x_2 \):
  \[
  t = \frac{f(x_1)}{f(x_2) - f(x_1)}
  \]
Computing the intersections:

- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections
Marching Squares (2D)

Given a function: $f(x)$

- $f(x) < 0$ inside
- $f(x) > 0$ outside

1. Discretize space.
2. Evaluate $f(x)$ on a grid.
3. Classify grid points (+/-)
4. Classify grid edges
5. Compute intersections
6. Connect intersections
Marching Squares (2D)

Some cases are ambiguous!
Marching Squares (2D)

Some cases are ambiguous!

How to connect?

Two options:
1) Can resolve ambiguity by subsampling inside the cell.
2) If subsampling is impossible, pick one of the two possibilities.
Marching Cubes (3D)

Same machinery: cells $\rightarrow$ **cubes** (voxels), lines $\rightarrow$ triangles

- 256 different cases - 15 after symmetries, 6 ambiguous cases
- More subsampling rules $\rightarrow$ 33 unique cases

Chernyaev, Marching Cubes 33,'95
Sampling

MESH-\rightarrow\ \text{POINT\ CLOUD}
From Surface to Point Cloud - Why?

- Points are simple but expressive!
- Few points can suffice
- Flexible, unstructured, few constraints
- Also: ML applications!

CAD meshes:
- many components
- bad triangles
- connectivity problems
From Surface to Point Cloud - Why?

- Points are simple but expressive!
- Few points can suffice
- Flexible, unstructured, few constraints
- Also: ML applications!

CAD meshes:
- many components
- bad triangles
- connectivity problems

the problem: sampling the mesh
Farthest Point Sampling

- Introduced for progressive transmission/acquisition of images
- Quality of approximation improves with increasing number of samples
  - as opposed eg. to raster scan
- Key Idea: repeatedly place next sample in the middle of the least-known area of the domain.

Gonzalez 1985, “Clustering to minimize the maximum intercluster distance”
Hochbaum and Shmoys 1985, “A best possible heuristic for the k-center problem”
Pipeline

1. Create an initial sample point set $S$
   - Image corners + additional random point.

2. Find the point which is the farthest from all point in $S$

   \[
   d(p, S) = \max_{q \in A} (d(q, S))
   = \max_{q \in A} \left( \min_{0 \leq i < N} (d(q, s_i)) \right)
   \]

3. Insert the point to $S$ and update the distances

4. While more points are needed, iterate
Farthest Point Sampling

- Depends on a notion of distance on the sampling domain
- Can be made adaptive, via a weighted distance

FPS on surfaces

What’s an appropriate distance?

Intrinsically far

Extrinsically close
On-Surface Distances

**Geodesics:**
Straightest and **locally shortest** curves

- Isolines - Euclidean
- Isolines - Geodesic

Distance on Manifold

Euclidean distance
Discrete Geodesics

Recall: a mesh is a graph!

Approximate geodesics as paths along edges

\[ v_0 = \text{initial vertex} \]
\[ d_i = \text{current distance to vertex } i \]
\[ S = \text{vertices with known optimal distance} \]

\# initialize
\[ d_0 = 0 \]
\[ d_i = \begin{cases} \inf \text{ for } d \in d_i \\ S = \{ \} \end{cases} \]

for each iteration \( k \):
    \# update
    \[ k = \text{argmin}(d_k), \text{ for } v_k \text{ not in } S \]
    \[ S.\text{append}(v_k) \]
    for neighbors index \( v_l \) of \( v_k \):
    \[ d_l = \min(\[d_l, d_k + d_{kl}\]) \]
Dijkstra Geodesics

\[ l = \sqrt{2} \quad l = 2 \]

Can be asymmetric - no matter how fine the mesh!
Dijkstra Geodesics

Dijkstra as front propagation
Fast Marching Geodesics

A better approximation: allow fronts to cross triangles!

Kimmel and Sethian 1997, “Computing Geodesic Paths on Manifolds”
FPS on a Mesh

Peyré and Cohen 2003, Geodesic Remeshing Using Front Propagation
Recap: Conversions
Geometry Foundations: Discrete Differential Geometry

slides credits: Daniele Panozzo
Differential Geometry Basics

- Geometry of manifolds
- Things that can be discovered by local observation: point + neighborhood
Differential Geometry Basics

- Geometry of manifolds
- Things that can be discovered by local observation: point + neighborhood

manifold point
Differential Geometry Basics

- Geometry of manifolds
- Things that can be discovered by local observation: point + neighborhood

manifold point
Differential Geometry Basics

- Geometry of manifolds
- Things that can be discovered by local observation: point + neighborhood

![Diagram showing manifold point with continuous 1-1 mapping and text indicating it can be used to calculate things!](image)
Differential Geometry Basics

- Geometry of manifolds
- Things that can be discovered by local observation: point + neighborhood

Diagram:
- Manifold point
- Continuous 1-1 mapping
- Non-manifold point
Differential Geometry Basics

- Geometry of manifolds
- Things that can be discovered by local observation: point + neighborhood

![Diagram showing manifold point and non-manifold point with continuous 1-1 mapping](image.png)
Differential Geometry Basics

- Geometry of manifolds
- Things that can be discovered by local observation: point + neighborhood
Differential Geometry Basics

- Geometry of manifolds
- Things that can be discovered by local observation: point + neighborhood

Continuous 1-1 mapping
Differential Geometry Basics

Geometry of manifolds

Things that can be discovered by local observation: point + neighborhood

If a sufficiently smooth mapping can be constructed, we can look at its first and second derivatives

Tangents, normals, curvatures, curve angles, distances
Example: Local Distance

isolines - geodesic

another important example: curvature!
Curves

2D: \( \mathbf{p}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} \), \( t \in [t_0, t_1] \)

\( \mathbf{p}(t) \) must be continuous

\[
\text{len}(\mathbf{p}(t_0), \mathbf{p}(t)) = \int_{0}^{t} \| \mathbf{p}'(t) \| \, dt
\]
Arc Length Parameterization

- Equal pace of the parameter along the curve

$$\text{len } (\mathbf{p}(t_1), \mathbf{p}(t_2)) = |t_1 - t_2|$$

- Now parameter goes from 0 to L
Secant

A line through two points on the curve.
Secant

A line through two points on the curve.
Tangent

The limiting secant as the two points come together.
Secant and Tangent – Parametric Form

**Secant:** $p(t) - p(s)$

**Tangent:** $p'(t) = (x'(t), y'(t), \ldots)^T$

If $t$ is arc-length:

$$\|p'(t)\| = 1$$

Recall

$$len(p(t_0), p(t)) = \int_{0}^{t} \|p'(t)\| \, dt$$

(curve “geodesic”)
Circle of Curvature

Consider the circle passing through three points on the curve...
…the limiting circle as three points come together.
Tangent, normal, radius of curvature

Osculating circle
“best fitting circle”
Radius of Curvature, \( r = \frac{1}{\kappa} \)

Curvature

\[ \kappa = \frac{1}{r} \]
Curvature is scale dependent

\[ \kappa = \frac{1}{r} \]

\[ \kappa \]

\[ \kappa \]

\[ \kappa \]
Assuming $t$ is arc-length parameter:

$$\mathbf{p}''(t) = \kappa \hat{\mathbf{n}}(t)$$

normal to the curve

$$\kappa = \| \hat{\mathbf{n}}'(t) \|$$
Discrete Planar Curves
Tangents, Normals

For any point on the edge, the tangent is simply the unit vector along the edge and the normal is the perpendicular vector.
Tangents, Normals

For vertices, we have many options
Tangents, Normals

Can choose to average the adjacent edge normals

\[ \hat{n}_v = \frac{\hat{n}_{e_1} + \hat{n}_{e_2}}{\|\hat{n}_{e_1} + \hat{n}_{e_2}\|} \]
Tangents, Normals

Weight by edge lengths

$$\hat{n}_v = \frac{|e_1|\hat{n}_{e_1} + |e_2|\hat{n}_{e_2}}{||e_1|\hat{n}_{e_1} + |e_2|\hat{n}_{e_2}||}$$
The Length of a Discrete Curve

Sum of edge lengths

\[ \text{len}(p) = \sum_{i=1}^{n-1} \| p_{i+1} - p_i \| \]
Curvature of a Discrete Curve

Curvature is the change in normal direction as we travel along the curve.

no change along each edge – curvature is zero along edges
Curvature of a Discrete Curve

Curvature is the change in normal direction as we travel along the curve.

normal changes at vertices – record the turning angle!
Curvature of a Discrete Curve

Curvature is the change in normal direction as we travel along the curve.

normal changes at vertices – record the turning angle!
Curvature of a Discrete Curve

Curvature is the change in normal direction as we travel along the curve.

Same as the turning angle between the edges.
Curvature of a Discrete Curve

- Zero along the edges
- Turning angle at the vertices = the change in normal direction

\[ \alpha_1, \alpha_2 > 0, \quad \alpha_3 < 0 \]
Surfaces, Parametric Form

Continuous surface

\[ p(u, v) = \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix}, \ (u, v) \in \mathbb{R}^2 \]

Tangent plane at point \( p(u, v) \) is spanned by

\[ p_u = \frac{\partial p(u, v)}{\partial u}, \quad p_v = \frac{\partial p(u, v)}{\partial v} \]

These vectors don’t have to be orthogonal
Surface Normals

Surface normal:

\[ \mathbf{n}(u, v) = \frac{\mathbf{p}_u \times \mathbf{p}_v}{\| \mathbf{p}_u \times \mathbf{p}_v \|} \]

Assuming *regular* parameterization, i.e.,

\[ \mathbf{p}_u \times \mathbf{p}_v \neq 0 \]
Normal Curvature

Unit-length direction $\mathbf{t}$ in the tangent plane (if $\mathbf{p}_u$ and $\mathbf{p}_v$ are orthogonal):

$$\mathbf{t} = \cos \varphi \frac{\mathbf{p}_u}{\|\mathbf{p}_u\|} + \sin \varphi \frac{\mathbf{p}_v}{\|\mathbf{p}_v\|}$$

$$\mathbf{n}(u, v) = \frac{\mathbf{p}_u \times \mathbf{p}_v}{\|\mathbf{p}_u \times \mathbf{p}_v\|}$$
The curve $\gamma$ is the intersection of the surface with the plane through $\mathbf{n}$ and $\mathbf{t}$.

Normal curvature:

$$\kappa_n(\varphi) = \kappa(\gamma(p))$$
Reminder: Radius of Curvature

Curvature

$$\kappa = \frac{1}{r}$$

Curve

Osculating circle of radius $r$
Surface Curvatures

- Principal curvatures
  - Minimal curvature \( \kappa_1 = \kappa_{\text{min}} = \min_{\varphi} \kappa_n(\varphi) \)
  - Maximal curvature \( \kappa_2 = \kappa_{\text{max}} = \max_{\varphi} \kappa_n(\varphi) \)

- Mean curvature
  \[ H = \frac{\kappa_1 + \kappa_2}{2} = \frac{1}{2\pi} \int_0^{2\pi} \kappa_n(\varphi) d\varphi \]

- Gaussian curvature
  \[ K = \kappa_1 \cdot \kappa_2 \]
Principal Directions

Principal directions: tangent vectors corresponding to \( \varphi_{\text{max}} \) and \( \varphi_{\text{min}} \)

\[ t_1 \quad t_2 \]

tangent plane

min curvature

max curvature
Euler’s Theorem: Planes of principal curvature are **orthogonal** and independent of parameterization.

\[ \kappa_n(\varphi) = \kappa_1 \cos^2 \varphi + \kappa_2 \sin^2 \varphi, \quad \varphi = \text{angle with } t_1 \]
Principal Directions
A point $p$ on the surface is called
- Elliptic, if $K > 0$
- Parabolic, if $K = 0$
- Hyperbolic, if $K < 0$

Developable surface
iff $K = 0$

can be mapped to the plane without distortion
Local Surface Shape By Curvatures

Isotropic:
all directions are principal directions

- $K > 0$, $\kappa_1 = \kappa_2$
  - spherical (umbilical)

Anisotropic:
2 distinct principal directions

- $K > 0$, $\kappa_1 > 0$, $\kappa_2 > 0$
  - elliptic

- $K = 0$, $\kappa_1 = 0$, $\kappa_2 > 0$
  - parabolic

- $K < 0$, $\kappa_1 < 0$, $\kappa_2 > 0$
  - hyperbolic
Software

- MATLAB-style (flat) C++ library, based on indexed face set structure

OpenMesh [www.openmesh.org](http://www.openmesh.org)
- Mesh processing, based on half-edge data structure

CGAL [www.cgal.org](http://www.cgal.org)
- Computational geometry

MeshLab [http://www.meshlab.net/](http://www.meshlab.net/)
- Viewing and processing meshes
Software

Alec Jacobson’s GP toolbox
https://github.com/alecjacobson/gptoolbox
MATLAB, various mesh and matrix routines

Gabriel Peyre’s Fast Marching Toolbox
https://www.mathworks.com/matlabcentral/fileexchange/6110-toolbox-fast-marching
Geodesic (on-surface) distances

OpenFlipper https://www.openflipper.org/
Various GP algorithms + Viewer
Spectral mesh processing

THE LAPLACE-BELTRAMI OPERATOR
Represent a function as a weighted sum of sines and cosines (basis functions)

\[ f(x) = a_0 + a_1 \cos(x) + a_2 \cos(3x) + a_3 \cos(5x) + a_4 \cos(7x) + \ldots \]

Coefficients: co-integrate function with basis
More generally - Fourier analysis

- Inner product for $L_2$ function space
  \[ \langle f, g \rangle := \int_{-\infty}^{\infty} f(x) \overline{g(x)} \, dx \]

- Orthonormal basis: complex "waves"
  \[ e_\omega(x) := e^{i2\pi \omega x} = \cos(2\pi \omega x) - i \sin(2\pi \omega x) \]

\[ F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \omega x} \, dx \]

\[ f(x) = \int_{-\infty}^{\infty} F(\omega) e^{2\pi i \omega x} \, d\omega \]
We can also Fourier on rectangular 2D domains

Fourier (DCT) basis functions for 8x8 grayscale images

\[ \cos(2\pi \omega_h) \cos(2\pi \omega_v) \]
Smoothing = filtering high frequencies out

Spatial domain $f(x) \rightarrow$ Frequency domain $F(u)$

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi ux} \, dx$$

Multiply by low-pass filter $G(u)$

$$F(u) \leftarrow F(u) \cdot G(u)$$

Frequency domain $F(u) \rightarrow$ Spatial domain $f(x)$

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{i2\pi ux} \, du$$
Extend Fourier to meshes?

So far, our functions have been defined on parameterized patches $f(u,v)$
- Generalize to meshes!

Fourier basis functions are eigenfunctions of the (standard) Laplace operator $\Delta: L^2 \rightarrow L^2$

$$\Delta \left( e^{2\pi i \omega x} \right) = \frac{\partial^2}{\partial x^2} e^{2\pi i \omega x} = -(2\pi \omega)^2 e^{2\pi i \omega x}$$

We need a discrete (mesh-based) version of this operator!
Continuous Laplace Operator

\[ f : \mathbb{R}^3 \rightarrow \mathbb{R} \quad \Delta f : \mathbb{R}^3 \rightarrow \mathbb{R} \]

Laplace operator

\[ \Delta f = \text{div} \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \ldots \]

gradient operator

2nd partial derivatives

function in Euclidean space

divergence operator

Cartesian coordinates

\[ \text{grad} f = \nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right) \]

\[ \text{div} \mathbf{F} = \nabla \cdot \mathbf{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z} \]
Continuous Laplace-Beltrami Operator

Extension of Laplace operator to functions on manifolds

\[ f : \mathcal{M} \rightarrow \mathbb{R} \quad \Delta f : \mathcal{M} \rightarrow \mathbb{R} \]

Laplace-Beltrami

\[ \Delta_{\mathcal{M}} f = \text{div}_{\mathcal{M}} \nabla_{\mathcal{M}} f \]

function on surface \( \mathcal{M} \)

divergence operator

device gradient operator

\[ \partial u \]

div \( u \)
Differential Properties on Meshes

- So for Laplacian, we need differential quantities (gradient, divergence…)
- Assumption: meshes are piecewise linear approximations of smooth surfaces
- Can try fitting a smooth surface locally (say, a polynomial) and find differential quantities analytically
- But: it is often too slow for interactive setting and error prone
Discrete Differential Operators

Approach: approximate differential properties at point $v$ as spatial average over local mesh neighborhood $N(v)$ where typically

$v = \text{mesh vertex}$

$N_k(v) = k$-ring neighborhood
Discrete Laplace-Beltrami

Uniform discretization: $L(f)$ or $\Delta f$

$$\Delta f(v) = \sum_{v_j \in N(v)} (f(v_j) - f(v))$$

$$= \sum_{v_j \in N(v)} f(v_j) - kf(v), \ k = |N(v)|$$

Similar to 5 point stencil for images!

Depends only on connectivity: simple and efficient

Bad approximation for irregular triangulations
In matrix form

\[ \Delta f(v) = \sum_{v_j \in N(v)} f(v_j) - kf(v), \quad k = |N(v)| \]

\[ Y = LF \]

\[
Y = \begin{bmatrix}
\Delta f(v_1) \\
\Delta f(v_2) \\
\Delta f(v_3) \\
\vdots \\
\Delta f(v_N)
\end{bmatrix}
\]

\[
L = \begin{bmatrix}
w_{11} & w_{12} & \cdots & w_{1N} \\
w_{21} & w_{22} & \cdots & w_{2N} \\
\vdots & \vdots & \ddots & \vdots \\
w_{N1} & w_{N2} & \cdots & w_{NN}
\end{bmatrix}
\]

\[
F = \begin{bmatrix}
f(v_1) \\
f(v_2) \\
f(v_3) \\
\vdots \\
f(v_N)
\end{bmatrix}
\]

\[
w_{ij} = \begin{cases} 
0 & \text{if } i \neq j, \# \text{ edge } (i, j) \\
1 & \text{if } i \neq j, \exists \text{ edge } (i, j) \\
-|N(v_i)| & \text{if } i = j
\end{cases}
\]
Discrete LB - cotangent formula

Better: cotangent formula

\[ \Delta_s f(v_i) := \frac{1}{2A_i} \sum_{v_j \in \mathcal{N}_1(v_i)} \left( \cot \alpha_{ij} + \cot \beta_{ij} \right) \left( f(v_j) - f(v_i) \right) \]

- \( A_i \): vertex area (Voronoi, barycentric..)
- \( \alpha_{ij} \)
- \( \beta_{ij} \)

Can be derived by discretizing continuous L-B via linear Finite Elements!
In matrix form:

$$\Delta_S f(v_i) := \frac{1}{2A_i} \sum_{v_j \in N_1(v_i)} (\cot \alpha_{ij} + \cot \beta_{ij}) (f(v_j) - f(v_i))$$

$$Y = A^{-1} Lf$$

$$Y = \begin{bmatrix} \Delta f(v_1) \\ \Delta f(v_2) \\ \Delta f(v_3) \\ \vdots \\ \Delta f(v_N) \end{bmatrix}, \quad L = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & \cdots & w_{NN} \end{bmatrix}, \quad \{w_{ij}\} = F$$

$$\omega_{ij} = \begin{cases} 0, & i \neq j, \nexists \text{ edge}(i, j) \\ \frac{1}{2}(\cot \alpha_{ij} + \cot \beta_{ij}), & i \neq j, \exists \text{ edge}(i, j) \\ -\sum_{k \neq i} \omega_{ik}, & i = j \end{cases}$$

$$A_{ij} = \begin{cases} A_i, & i = j \\ 0, & i \neq j \end{cases}$$
Laplace-Beltrami and Curvature

Apply operator to coordinate functions

\[ \Delta_{\mathcal{M}} \mathbf{p} = \text{div}_{\mathcal{M}} \nabla_{\mathcal{M}} \mathbf{p} = -2H \mathbf{n} \in \mathbb{R}^3 \]

Laplace-Beltrami

Gradient operator

Mean curvature

Coordinate functions on surface \( \mathcal{M} \)

\( \mathbf{p} = (x, y, z) \)

Divergence operator

Unit surface normal
Effect of the Discretization

- Uniform Laplacian $L_u(v_i)$
- Cotangent Laplacian $L_c(v_i)$
- Normal
Effect of the Discretization

- Uniform Laplacian $L_u(v_i)$
- Cotangent Laplacian $L_c(v_i)$
- Normal

For nearly equal edge lengths
Uniform $\approx$ Cotangent
Effect of the Discretization

- Uniform Laplacian $L_u(v_i)$
- Cotangent Laplacian $L_c(v_i)$
- Normal

For nearly equal edge lengths
Uniform $\approx$ Cotangent

Cotangent Laplacian allows computing discrete normal

Nice property: gives zero for planar 1-rings!
L-B eigendecomposition

Admits discrete decomposition

\[ \Delta_M \phi = \lambda \phi \]

\[ 0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \]

\{\phi_0, \phi_1, \ldots\} comprise basis for \( L_2 \) functions defined on \( M \)
L-B eigendecomposition

- Eigendecomposition is **invariant** to **isometric** = metric-preserving transformations

\[ \Delta_M \phi = \lambda \phi \]
\[ 0 \leq \lambda_1 \leq \lambda_2 \leq ... \]

- Relation to heat equation - next lecture