Geometric Modeling

Cotangent Laplacian Derivation

Acknowledgements: Olga Diamanti, Julian Panetta
Cotan Laplacian Motivation

- Discrete Laplacian enables many useful operations on triangle meshes:

\[ H_i = \frac{1}{2} \left\| [L v]_i \right\| \]

\[ n = -\frac{[L v]_i}{2H_i} \]

\[ (I - \lambda dt \mathbf{L})v^{n+1} = v^n \]

- Cotangent weights give \textbf{less mesh-dependent} results, ensuring \textbf{proper convergence} under mesh refinement
Where We’re Headed

\[ \Delta_M^{\cot\alpha} f(v) = \frac{1}{2A(v)} \sum_{v_i \in N(v)} (\cot(\alpha_i) + \cot(\beta_i)) \left( f(v_i) - f(v) \right) \]

- “Cotan Laplacian” refers to \( \frac{1}{2}(\cot(\alpha_i) + \cot(\beta_i)) \) weight on each edge contribution.

- We’ll consider two different ways to derive this formula:
  - “Traditional graphics approach:” Finite Volume-style
  - Finite Element Approach (more general)
Goal: Mesh Independence

• We want discrete Laplacian to converge to the Laplace Beltrami operator of smooth surface $S$ as mesh $M$ is refined to better approximate $S$.

$$\Delta^\text{cotan}_M F \rightarrow \Delta_S f$$

• $F$ is a vector of discrete values (per vertex), $f(x)$ is a smooth function.

• **Idea:** reinterpret $F$ as a function interpolating the smooth "$f$," and directly compute its Laplacian. Converge as $F \rightarrow f$
Piecewise-Linear Interpolation

• Since $F$ holds per-vertex values, it naturally corresponds to a linear function on each triangle

• (Corner values uniquely determine linear function over triangle)
"Hat Function" Basis

- Piecewise linear functions are expressed most simply using the "hat functions" (linear Lagrange polynomials)

- One basis function $\phi_i(x)$ per vertex, $v_i$

- **Unique linear function (on triangles)** having:

  $$\phi_i(v_j) = \begin{cases} 
  1 & \text{if } i = j \\
  0 & \text{otherwise}
  \end{cases}$$

- Interpolation of $F$ on mesh $M$ is just:

  $$f_M(x) = \sum_i f(v_i)\phi_i(x)$$
Gradients

• Recall, the Laplacian is the divergence of the gradient:

\[
\Delta f = \nabla \cdot (\nabla f)
\]

• So we first need to compute our interpolant’s gradient:

\[
\nabla f_M(x) = \nabla \left( \sum_i f(v_i) \phi_i(x) \right) = \sum_i f(v_i) \nabla \phi_i(x)
\]

• In other words, the gradient is just a linear combination of the basis function gradients \( \nabla \phi_i(x) \) weighted by the per-vertex \( f \) values.
Hat Function Gradients

- Basis functions are linear over each triangle, so these gradients are **constant on each triangle**

- We can determine gradient with simple geometric arguments:

\[
\begin{align*}
\phi_0(v_0) &= 1 \\
\phi_0(v_1) &= 0 \\
\phi_0(v_2) &= 0
\end{align*}
\]

Where does the gradient point? What about its magnitude?
Gradient Derivation

\[ \nabla \phi_0 \cdot e_0 = 0 \quad \implies \quad \nabla \phi_0 = \alpha e_0 \perp \]

\[ \nabla \phi_0 \cdot \left( h \frac{e_0^\perp}{\|e_0^\perp\|} \right) = 1 \]

\[ \implies \quad \nabla \phi_0 = \frac{1}{h} \frac{e_0^\perp}{\|e_0^\perp\|} \]

\[ \implies \quad \nabla \phi_0 = \frac{e_0^\perp}{2A} \]

(area is 1/2 * base * height)
Final Gradient Expression

\[ \nabla \phi_i |_T = \frac{\mathbf{e}_i}{2A_T} \]

- Note, this is the gradient evaluated on one triangle \( T \) incident vertex \( i \). (Different gradient on others).

- Works on curved triangle meshes too! Just need triangle area and inward-pointing perpendicular edge vectors:

\[ \mathbf{e}_i = \mathbf{n} \times \mathbf{e}_i \]

- Full gradient is then the linear combination

\[ \nabla f_M(x) = \sum_i f(v_i) \nabla \phi_i(x) \]
Now the Laplacian!

- Now we want to compute

\[ \Delta f_M = \nabla \cdot \nabla f_M(x) = \nabla \cdot \left( \sum_i f(v_i) \nabla \phi_i(x) \right) \]

- But the gradient is piecewise constant!
What does divergence even mean?
Too Many Derivatives!

- **Problem:** function is piecewise linear (only $C^0$ continuous) and we’re trying to take second derivatives

- But we can make sense of things by using the Divergence Theorem over some region “$R$”:

\[
\int_R \nabla \cdot (\nabla \phi) \, dx = \int_{\partial R} \mathbf{n} \cdot \nabla \phi \, ds
\]

- (This allows us to avoid explicitly computing divergence; just need dot products with the curve normals on $R$’s boundary.)
Approach 1: (Traditional in Graphics)

• Idea: define Laplacian on a vertex to be the average over some region surrounding the vertex.

\[
[LF]_i := \frac{1}{|R(v_i)|} \int_{R(v_i)} \nabla \cdot \nabla f_M(x) \, dx \\
= \frac{1}{|R(v_i)|} \int_{\partial R(v_i)} n \cdot \nabla f_M(x) \, ds
\]

• To get cotan Laplacian, you need to average over the vertices’ Voronoi regions (region for which v is the closest vtx).

• This region seems somewhat arbitrary; approach 2 won’t need this choice.
Approach 1 Continued

- Break into sum over triangles (so gradient is constant):

\[
[LF]_i = \frac{1}{|R(v_i)|} \int_{\partial R(v_i)} n \cdot \nabla f_M(x) \, ds
\]

\[
= \frac{1}{|R(v_i)|} \sum_{T \ni v_i} \int_{T \cap \partial R(v_i)} n \cdot \nabla f_M(x) \, ds
\]

\[
= \frac{1}{|R(v_i)|} \sum_{T \ni v_i} \nabla f_T \cdot \int_{T \cap \partial R(v_i)} n \, ds
\]

\[
= \frac{1}{|R(v_i)|} \sum_{T \ni v_i} \nabla f_T \cdot (n_1 \ell_1 + n_2 \ell_2)
\]
Voronoi Diagram Properties

\[ [LF]_i = \frac{1}{|R(v_i)|} \sum_{T \ni v_i} \nabla f_T \cdot (n_1 \ell_1 + n_2 \ell_2) \]

- Voronoi diagram edges **bisect** \( v_i \)'s incident edges

\[ n_1 \ell_1 + n_2 \ell_2 + n_3 \ell_3 = 0 \]

(Tri edge vectors sum to zero; so do their perpendiculars.)

\[ \Rightarrow \quad n_1 \ell_1 + n_2 \ell_2 = -n_3 \ell_3 = \left( \frac{v_k + v_i}{2} - \frac{v_j + v_i}{2} \right) \perp \]

\[ = \frac{1}{2} (v_k - v_j) \perp = \frac{e_i^\perp}{2} \]

\[ \Rightarrow \quad [LF]_i = \frac{1}{|R(v_i)|} \sum_{T \ni v_i} \nabla f_T \cdot \frac{e_i^\perp}{2} \]

CSCI-GA.3033-018 - Geometric Modeling - Daniele Panozzo
Plug in per-Tri Gradient

\[
[LF]_i = \frac{1}{|R(v_i)|} \sum_{T \ni v_i} \nabla f_T \cdot \frac{e_i^\perp}{2}
\]

\[
= \frac{1}{|R(v_i)|} \sum_{T \ni v_i} (f_i \nabla \phi_i + f_j \nabla \phi_j + f_k \nabla \phi_k) \cdot \frac{e_i^\perp}{2}
\]

\[
= \frac{1}{|R(v_i)|} \sum_{T \ni v_i} \left[ f_i \frac{e_i^\perp}{2AT} + f_j \frac{e_j^\perp}{2AT} + f_k \frac{e_k^\perp}{2AT} \right] \cdot \frac{e_i^\perp}{2}
\]

\[
= \frac{1}{|R(v_i)|} \sum_{T \ni v_i} \left[ -f_i \left( \frac{e_j^\perp}{2AT} + \frac{e_k^\perp}{2AT} \right) + f_j \frac{e_j^\perp}{2AT} + f_k \frac{e_k^\perp}{2AT} \right] \cdot \frac{e_i^\perp}{2}
\]

\[
= \frac{1}{|R(v_i)|} \sum_{T \ni v_i} \left[ (f_j - f_i) \frac{e_j^\perp}{2AT} + (f_k - f_i) \frac{e_k^\perp}{2AT} \right] \cdot \frac{e_i^\perp}{2}
\]
Final Steps

\[
[LF]_i = \frac{1}{|R(v_i)|} \sum_{T \ni v_i} \left[ \left( f_j - f_i \right) \frac{e_j}{2AT} + \left( f_k - f_i \right) \frac{e_k}{2AT} \right] \cdot \frac{e_i}{2}
\]

\[
= \frac{1}{|R(v_i)|} \sum_{T \ni v_i} \left( f_j - f_i \right) \frac{e_j}{4AT} + \left( f_k - f_i \right) \frac{e_k}{4AT}
\]

Note:

\[
\frac{e_i \cdot e_j}{4AT} = \frac{\|e_i\|\|e_j\| \cos(\alpha)}{2\|e_i\|\|e_j\| \sin(\alpha)} = \frac{1}{2} \cot(\alpha)
\]

\[
\frac{e_i \cdot e_k}{4AT} = \frac{\|e_i\|\|e_k\| \cos(\beta)}{2\|e_i\|\|e_k\| \sin(\beta)} = \frac{1}{2} \cot(\beta)
\]

\[
[LF]_i = \frac{1}{|R(v_i)|} \sum_{T \ni v_i} \left( f_j - f_i \right) \cot(\alpha) + \left( f_k - f_i \right) \cot(\beta)
\]
Regrouping Terms

- We have the cotan weights now, but expressed as a sum over triangles:

\[
[L_F]_i = \frac{1}{2|R(v_i)|} \sum_{T \ni v_i} (f_j - f_i) \cot \alpha + (f_k - f_i) \cot \beta
\]

- To get the standard formula, we can regroup these terms as a sum over edges:

\[
[L_F]_i = \frac{1}{2|R(v_i)|} \sum_{v_j \in N(v_i)} (\cot(\alpha_{ij}) + \cot(\beta_{ij})) \left( f(v_j) - f(v_i) \right)
\]

- \(|R(v_i)|\) is the area of the \(i^{th}\) vertex’s Voronoi region (the averaging area)
Approach 2: Finite Element Method

• Instead of trying to discretize the Laplacian itself (using averaging regions), we **discretize the whole PDE** containing it. For instance:

\[-\Delta u = b \quad \text{in } M\]

• Here “b” is a forcing term (e.g. injecting/sinking heat at each point), and we assume M is a closed surface mesh without a boundary.

• If M had boundary curves, we’d need to specify boundary conditions.
Weak Form of the Poisson Equation

• We can’t expect a piecewise linear function “u” to solve this PDE exactly (why does its Laplacian mean, anyway?)

• But we can ask it to be a “weak solution:”

\[- \int_M \psi \Delta u \, dx = \int_M \psi b \, dx \quad \forall \psi \text{ “test functions”}\]

• In other words, we ask for zero residual “along” all test functions.

• Linear FEM: choose piecewise linear functions for both \(u\) and \(\psi\)

• Physical interpretation: minimizing a potential energy over the space of piecewise linear functions
Expand Functions in Basis

\[- \int_M \psi \Delta u \, dx = \int_M \psi b \, dx\]

\[u(x) := \sum_j u_j \phi_j(x) \quad \psi(x) := \sum_i a_i \phi_i(x) \quad b(x) := \sum_j b_j \phi_j(x)\]

\[- \int_M \left( \sum_i a_i \phi_i(x) \right) \Delta \left( \sum_j u_j \phi_j(x) \right) \, dx = \int_M \left( \sum_i a_i \phi_i(x) \right) \left( \sum_j b_j \phi_j(x) \right) \, dx\]

\[\Rightarrow - \sum_{i,j} a_i u_j \left( \int_M \phi_i \Delta \phi_j \, dx \right) = \sum_{i,j} a_i b_j \left( \int_M \phi_i \phi_j \, dx \right)\]
Integration by Parts for \( \int_M \phi_i \Delta \phi_j \, dx \)

- To avoid second derivatives, apply **divergence theorem** in the form of an **integration by parts** (move derivative onto \( \phi_i \))

- Note the “product rule”: \( \nabla \cdot (\phi_i \nabla \phi_j) = \phi_i \Delta \phi_j + \nabla \phi_i \cdot \nabla \phi_j \quad \implies \quad \phi_i \Delta \phi_j = \nabla \cdot (\phi_i \nabla \phi_j) - \nabla \phi_i \cdot \nabla \phi_j \)

Apply divergence theorem

\[
\begin{align*}
- \int_M \phi_i \Delta \phi_j \, dx &= - \int_M \nabla \cdot (\phi_i \nabla \phi_j) - \nabla \phi_i \cdot \nabla \phi_j \, dx \\
&= - \int_{\partial M} \mathbf{n} \cdot (\phi_i \nabla \phi_j) \, ds + \int_M \nabla \phi_i \cdot \nabla \phi_j \, dx \\
&= \int_M \nabla \phi_i \cdot \nabla \phi_j \, dx
\end{align*}
\]
Laplacian and Mass Matrices

• Now our weak PDE looks like:

\[ \sum_{i,j} a_i u_j \left( \int_M \nabla \phi_i \nabla \phi_j \, dx \right) = \sum_{i,j} a_i b_j \left( \int_M \phi_i \phi_j \, dx \right) \]

\[ L_{ij} \quad M_{ij} \]

• Expressed in matrix form:

\[ a^T Lu = a^T Mb \quad \forall a \]

\[ \Rightarrow \quad Lu = Mb \]

• Plugging in gradient expressions:

\[ L_{ij} = \int_M \nabla \phi_i \cdot \nabla \phi_j \, dx = \sum_{T \ni v_i, v_j} \frac{\mathbf{e}_{T,i}^\perp}{2A_T} \cdot \frac{\mathbf{e}_{T,j}^\perp}{2A_T} A_T = \sum_{T \ni v_i, v_j} \frac{\mathbf{e}_{T,i}^\perp \cdot \mathbf{e}_{T,j}^\perp}{4A_T} \]
Laplacian Matrix Cotangent Weights

\[ L_{ij} = \begin{dcases} \sum_{T \ni v_i} \frac{e_{T,i}^T \cdot (-e_{T,j}^T - e_{T,k}^T)}{4A_T} & = -\frac{1}{2} \sum_{v_k \in N(v_i)} (\cot(\alpha_{ik}) + \cot(\beta_{ik})) \quad \text{if } i = j \\ \frac{1}{2} (\cot(\alpha_{ij}) + \cot(\beta_{ij})) & \quad \text{if } i \neq j \end{dcases} \]
FEM vs “Graphics Approach”

- Get exactly the same cotan weights (up to division by region area)
- FEM doesn’t require defining an averaging region
- FEM computes Laplacian \textbf{integrated against test function} instead of an \textbf{averaged} (point) quantity:
  \[ L_{\text{FEM}} = M_{\text{diag}} L_{\text{Avg}} \quad M_{\text{diag}} = \text{diag}([R(v_0), R(v_1), \cdots]) \]
- FEM uses \textbf{exact mass matrix} to integrate RHS against test functions; graphics approach uses \textbf{diagonal “Lumped” approximation} \( M_{\text{diag}} \) based on region areas (e.g. Voronoi)—still converges, but higher error. (Lumping effectively treats the RHS function “\( b \)” as \textbf{piecewise constant} over the averaging region, while FEM+exact mass matrix treats it as \textbf{piecewise linear}).
- FEM generalizes to higher degree; just use more basis functions!