Laplace-Beltrami:
The Swiss Army Knife of Geometry Processing

(SGP 2014 Tutorial—July 7 2014)

Justin Solomon / *Stanford University*
Keenan Crane / *Columbia University*
Etienne Vouga / *Harvard University*

INTRODUCTION

Introduction

- *Laplace-Beltrami operator* (“Laplacian”) provides a basis for a diverse variety of geometry processing tasks.

Introduction

- *Laplace-Beltrami operator* (“Laplacian”) provides a basis for a diverse variety of geometry processing tasks.
- Remarkably common pipeline:

Introduction

- *Laplace-Beltrami operator* (“Laplacian”) provides a basis for a diverse variety of geometry processing tasks.
- Remarkably common pipeline:
  1. simple pre-processing (build f)

Introduction

- *Laplace-Beltrami operator* ("Laplacian") provides a basis for a diverse variety of geometry processing tasks.
- Remarkably common pipeline:
  1. simple pre-processing (build f)
  2. solve a PDE involving the Laplacian (e.g., $\Delta u = f$)

Introduction

- *Laplace-Beltrami operator* ("Laplacian") provides a basis for a diverse variety of geometry processing tasks.
- Remarkably common pipeline:
  1. simple pre-processing (build f)
  2. solve a PDE involving the Laplacian (e.g., $\Delta u = f$)
  3. simple post-processing (do something with u)

- *Laplace-Beltrami operator* ("Laplacian") provides a basis for a diverse variety of geometry processing tasks.
- Remarkably common pipeline:
  1. simple pre-processing (build f)
  2. solve a PDE involving the Laplacian (e.g., $\Delta u = f$)
  3. simple post-processing (do something with u)
- Expressing tasks in terms of Laplacian/smooth PDEs makes life easier at code/implementation level.

Introduction

- *Laplace-Beltrami operator* ("Laplacian") provides a basis for a diverse variety of geometry processing tasks.
- Remarkably common pipeline:
  1. simple pre-processing (build f)
  2. solve a PDE involving the Laplacian (e.g., $\Delta u = f$)
  3. simple post-processing (do something with u)
- Expressing tasks in terms of Laplacian/smooth PDEs makes life easier at code/implementation level.
- Lots of existing theory to help understand/interpret algorithms, provide analysis/guarantees.

Introduction

- *Laplace-Beltrami operator* ("Laplacian") provides a basis for a diverse variety of geometry processing tasks.
- Remarkably common pipeline:
  1. simple pre-processing (build f)
  2. solve a PDE involving the Laplacian (e.g., $\Delta u = f$)
  3. simple post-processing (do something with u)
- Expressing tasks in terms of Laplacian/smooth PDEs makes life easier at code/implementation level.
- Lots of existing theory to help understand/interpret algorithms, provide analysis/guarantees.
- Also makes it easy to work with a broad range of geometric data structures (meshes, point clouds, etc.)

- Goals of this tutorial:

Introduction

- Goals of this tutorial:
  - Understand the Laplacian in the smooth setting. *(Etienne)*

Introduction

- Goals of this tutorial:
  - Understand the Laplacian in the smooth setting. *(Etienne)*

  - Build the Laplacian in the discrete setting. *(Keenan)*

Introduction

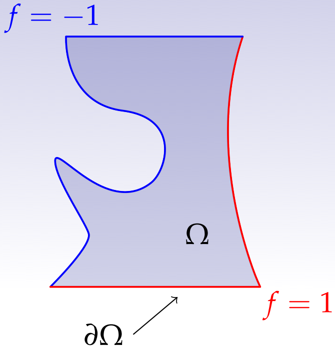
- Goals of this tutorial:
  - Understand the Laplacian in the smooth setting. *(Etienne)*

  - Build the Laplacian in the discrete setting. *(Keenan)*

  - Use Laplacian to implement a variety of methods. *(Justin)*

Smooth Theory

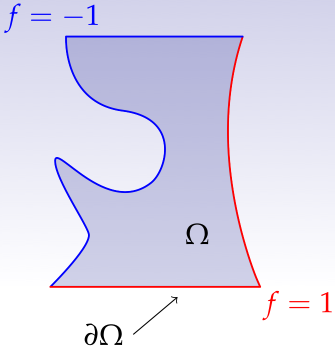
The Interpolation Problem



- given:
  - region $\Omega \subset \mathbb{R}^2$ with boundary $\partial\Omega$
  - function f on $\partial\Omega$

  fill in f "as smoothly as possible"

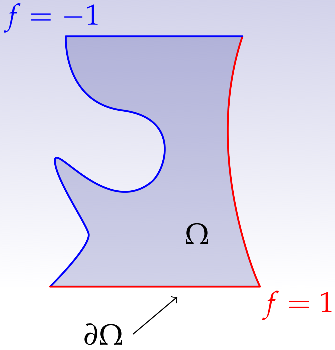
The Interpolation Problem



- given:
  - region $\Omega \subset \mathbb{R}^2$ with boundary $\partial\Omega$
  - function f on $\partial\Omega$

  fill in f "as smoothly as possible"
- *(what does this even mean?)*

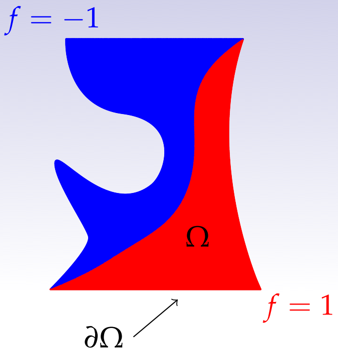
The Interpolation Problem



- given:
  - region $\Omega \subset \mathbb{R}^2$ with boundary $\partial\Omega$
  - function f on $\partial\Omega$

  fill in f "as smoothly as possible"
- *(what does this even mean?)*
- smooth:
  - constant functions
  - linear functions

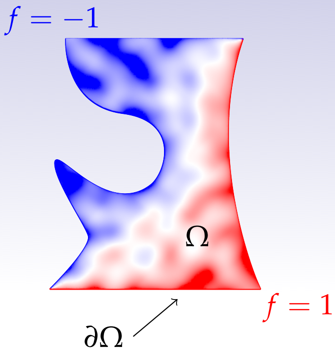
The Interpolation Problem



- given:
  - region $\Omega \subset \mathbb{R}^2$ with boundary $\partial\Omega$
  - function f on $\partial\Omega$

  fill in f “as smoothly as possible”
- *(what does this even mean?)*
- smooth:
  - constant functions
  - linear functions
- not smooth:
  - f not C^1

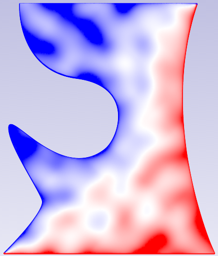
The Interpolation Problem



- given:
    - region $\Omega \subset \mathbb{R}^2$ with boundary $\partial\Omega$
    - function f on $\partial\Omega$

  fill in f "as smoothly as possible"
- *(what does this even mean?)*
- smooth:
    - constant functions
    - linear functions
- not smooth:
    - f not C^1
    - large variations over short distances
    - ($\|\nabla f\|$ large)

Dirichlet Energy

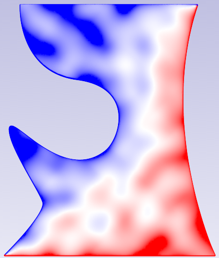


non-smooth $f(x)$

$\langle \nabla f, \nabla f \rangle$

- $E(f) = \int_\Omega \langle \nabla f, \nabla f \rangle \, dA$
- properties:
    - nonnegative
    - zero for constant functions
    - measures smoothness

Dirichlet Energy

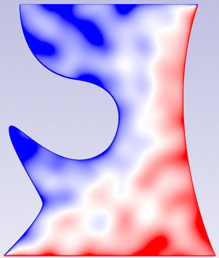


non-smooth $f(x)$

$\langle \nabla f, \nabla f \rangle$

- $E(f) = \int_\Omega \langle \nabla f, \nabla f \rangle \, dA$
- properties:
    - nonnegative
    - zero for constant functions
    - measures smoothness
- solution to interpolation problem is minimizer of E

Dirichlet Energy

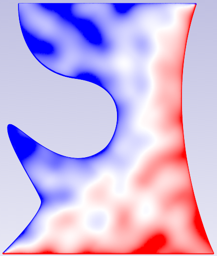


non-smooth $f(x)$

$\langle \nabla f, \nabla f \rangle$

- $E(f) = \int_\Omega \langle \nabla f, \nabla f \rangle \, dA$
- properties:
  - nonnegative
  - zero for constant functions
  - measures smoothness
- solution to interpolation problem is minimizer of E
- how do we find minimum?

Dirichlet Energy



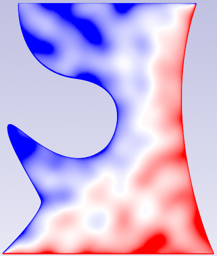
non-smooth $f(x)$

$\langle \nabla f, \nabla f \rangle$

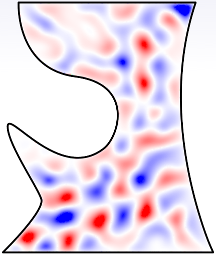
- $E(f) = \int_\Omega \langle \nabla f, \nabla f \rangle \, dA$
- it can be shown that:
  - $E(f) = C - \int_\Omega f \Delta f \, dA$

Dirichlet Energy

- $E(f) = \int_\Omega \langle \nabla f, \nabla f \rangle \, dA$
- it can be shown that:
  - $E(f) = C - \int_\Omega f \Delta f \, dA$
  - $-2\Delta f$ is the *gradient of Dirichlet energy*

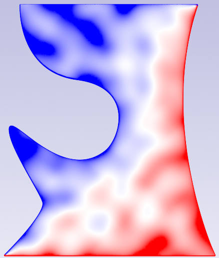


non-smooth $f(x)$

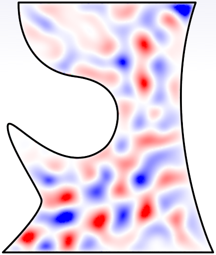


Δf

Dirichlet Energy



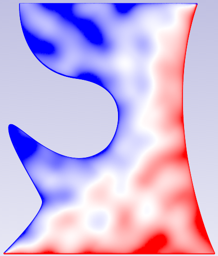
non-smooth $f(x)$



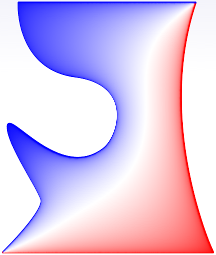
Δf

- $E(f) = \int_\Omega \langle \nabla f, \nabla f \rangle \, dA$
- it can be shown that:
    - $E(f) = C - \int_\Omega f \Delta f \, dA$
    - $-2\Delta f$ is the *gradient of Dirichlet energy*
    - f minimizes E if $\Delta f = 0$

Dirichlet Energy



non-smooth $f(x)$

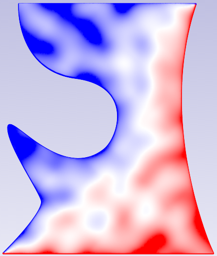


solution $\Delta f = 0$

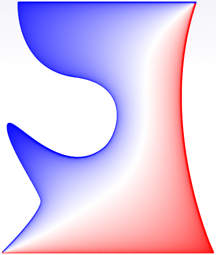
- $E(f) = \int_\Omega \langle \nabla f, \nabla f \rangle \, dA$
- it can be shown that:
  - $E(f) = C - \int_\Omega f \Delta f \, dA$
  - $-2\Delta f$ is the *gradient of Dirichlet energy*
  - f minimizes E if $\Delta f = 0$
- PDE form (*Laplace's Equation*):

$$\begin{aligned} \Delta f(x) &= 0 & x &\in \Omega \\ f(x) &= f_0(x) & x &\in \partial\Omega \end{aligned}$$

Dirichlet Energy



non-smooth $f(x)$



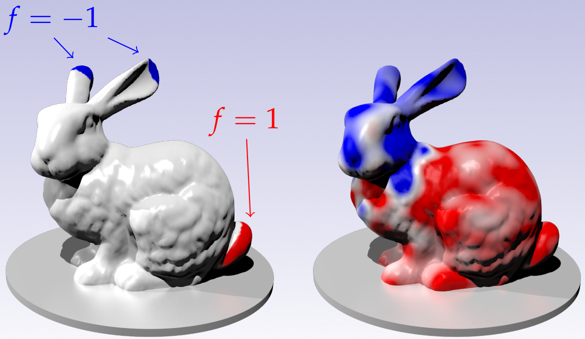
solution $\Delta f = 0$

- $E(f) = \int_\Omega \langle \nabla f, \nabla f \rangle \, dA$
- it can be shown that:
  - $E(f) = C - \int_\Omega f \Delta f \, dA$
  - $-2\Delta f$ is the *gradient of Dirichlet energy*
  - f minimizes E if $\Delta f = 0$
- PDE form (*Laplace's Equation*):

$$\begin{aligned} \Delta f(x) &= 0 & x &\in \Omega \\ f(x) &= f_0(x) & x &\in \partial\Omega \end{aligned}$$

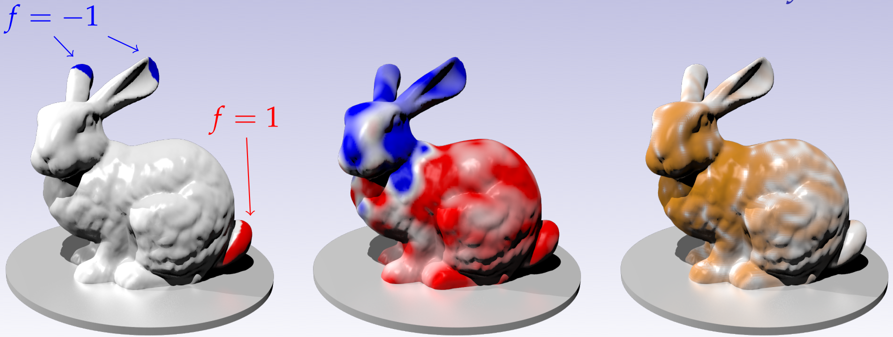
- physical interpretation: temperature at steady state

On a Surface



boundary conditions nonsmooth $f(x)$

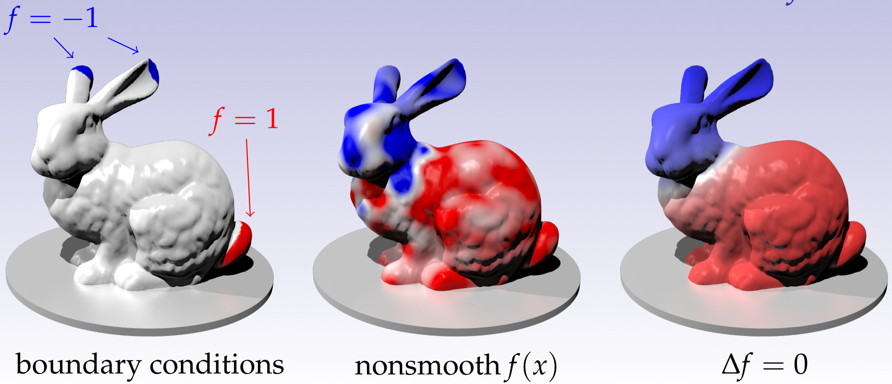
On a Surface



boundary conditions — nonsmooth $f(x)$ — $\langle \nabla f, \nabla f \rangle$

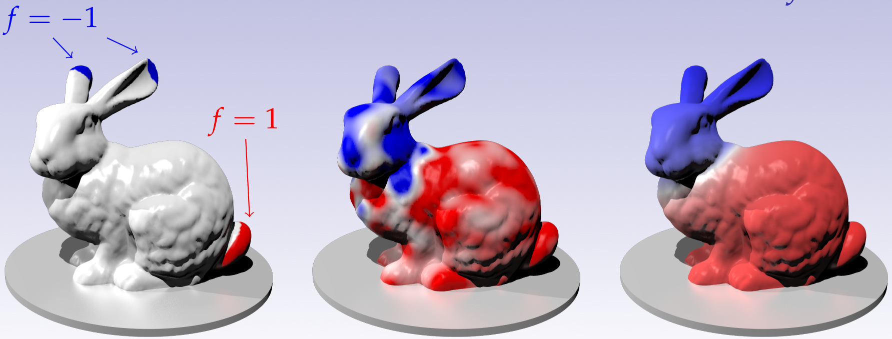
- can still define Dirichlet energy $E(f) = \int_M \langle \nabla f, \nabla f \rangle$

On a Surface



- can still define Dirichlet energy $E(f) = \int_M \langle \nabla f, \nabla f \rangle$
- $\nabla E(f) = -\Delta f$, now Δ is the *Laplace-Beltrami operator* of M

On a Surface



boundary conditions | nonsmooth $f(x)$ | $\Delta f = 0$

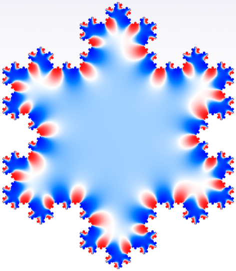
- can still define Dirichlet energy $E(f) = \int_M \langle \nabla f, \nabla f \rangle$
- $\nabla E(f) = -\Delta f$, now Δ is the *Laplace-Beltrami operator* of M
- also works in higher dimensions, on discrete graphs/point clouds, ...

Existence and Uniqueness

- Laplace's equation

$$\begin{aligned} \Delta f(x) &= 0 & x &\in M \\ f(x) &= f_0(x) & x &\in \partial M \end{aligned}$$

has a unique solution for all reasonable[1] surfaces M



[1] e.g. compact, smooth, with piecewise smooth boundary

Existence and Uniqueness

- Laplace's equation

$$\begin{aligned} \Delta f(x) &= 0 & x &\in M \\ f(x) &= f_0(x) & x &\in \partial M \end{aligned}$$

  has a unique solution for all reasonable[1] surfaces M

- physical interpretation: apply heating/cooling f_0 to the boundary of a metal plate. Interior temperature will reach *some* steady state

[1] e.g. compact, smooth, with piecewise smooth boundary

Existence and Uniqueness

- Laplace's equation

$$\begin{aligned} \Delta f(x) &= 0 & x &\in M \\ f(x) &= f_0(x) & x &\in \partial M \end{aligned}$$

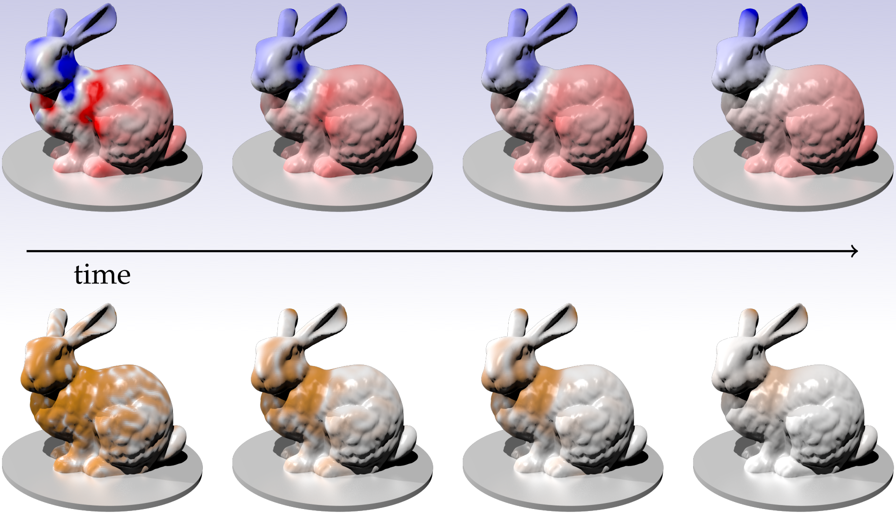
has a unique solution for all reasonable[1] surfaces M

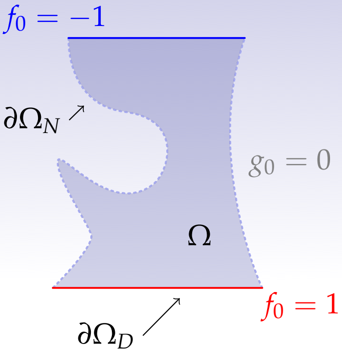
- physical interpretation: apply heating/cooling f_0 to the boundary of a metal plate. Interior temperature will reach *some* steady state
- gradient descent is exactly the *heat* or *diffusion* equation

$$\frac{df}{dt}(x) = \Delta f(x).$$

[1] e.g. compact, smooth, with piecewise smooth boundary

Heat Equation Illustrated

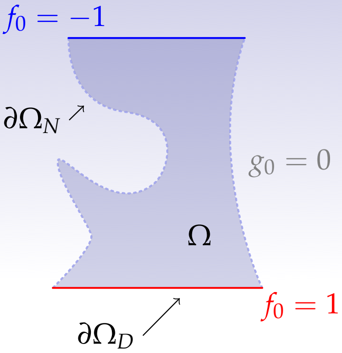




- can specify $\nabla f \cdot \hat{n}$ on boundary instead of f:

$$\begin{aligned} \Delta f(x) &= 0 && x \in \Omega \\ f(x) &= f_0(x) && x \in \partial\Omega_D && \textit{(Dirichlet bdry)} \\ \nabla f \cdot \hat{n} &= g_0(x) && x \in \partial\Omega_N && \textit{(Neumann bdry)} \end{aligned}$$

Boundary Conditions

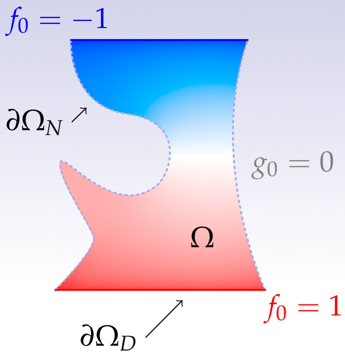


- can specify $\nabla f \cdot \hat{n}$ on boundary instead of f:

$$\begin{aligned} \Delta f(x) &= 0 & x &\in \Omega & \\ f(x) &= f_0(x) & x &\in \partial\Omega_D & \textit{(Dirichlet bdry)} \\ \nabla f \cdot \hat{n} &= g_0(x) & x &\in \partial\Omega_N & \textit{(Neumann bdry)} \end{aligned}$$

- usually: $g_0 = 0$ (*natural* bdry conds)

Boundary Conditions



- can specify $\nabla f \cdot \hat{n}$ on boundary instead of f:

$$\begin{aligned} \Delta f(x) &= 0 & x &\in \Omega & \\ f(x) &= f_0(x) & x &\in \partial\Omega_D & \textit{(Dirichlet bdry)} \\ \nabla f \cdot \hat{n} &= g_0(x) & x &\in \partial\Omega_N & \textit{(Neumann bdry)} \end{aligned}$$

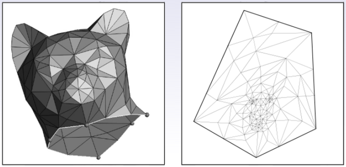
- usually: $g_0 = 0$ (*natural* bdry conds)
- physical interpretation: free boundary through which heat cannot flow

Interpolation with Δ in Practice

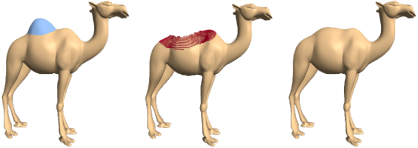
in geometry processing:

- positions
- displacements
- vector fields
- parameterizations
- …you name it

Joshi et al

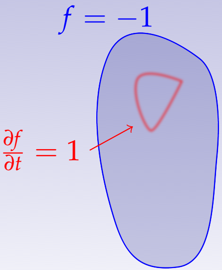


Eck et al



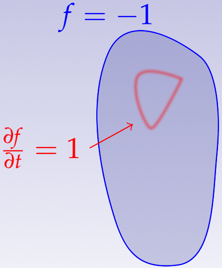
Sorkine and Cohen-Or

Heat Equation with Source



- what if you add heat sources inside Ω?

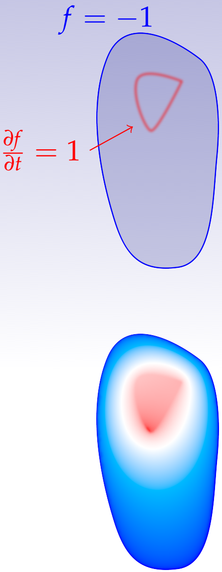
Heat Equation with Source



- what if you add heat sources inside Ω?

$$\frac{df}{dt}(x) = g(x) + \Delta f(x)$$

Heat Equation with Source



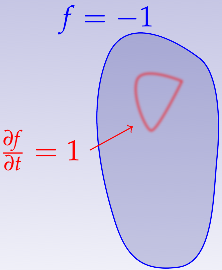
- what if you add heat sources inside Ω?

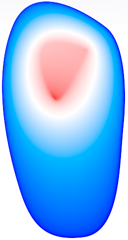
$$\frac{df}{dt}(x) = g(x) + \Delta f(x)$$

- PDE form: *Poisson's equation*

$$\begin{aligned} \Delta f(x) &= g(x) \qquad && x \in \Omega \\ f(x) &= f_0(x) \qquad && x \in \partial\Omega \end{aligned}$$

Heat Equation with Source





- what if you add heat sources inside Ω?

$$\frac{df}{dt}(x) = g(x) + \Delta f(x)$$

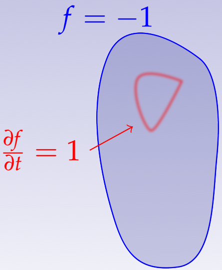
- PDE form: *Poisson's equation*

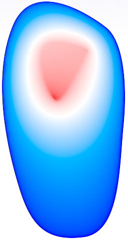
$$\begin{aligned} \Delta f(x) &= g(x) & x &\in \Omega \\ f(x) &= f_0(x) & x &\in \partial\Omega \end{aligned}$$

- common variational problem:

$$\min_f \int_M \langle \nabla f - \mathbf{v}, \nabla f - \mathbf{v} \rangle \, dA$$

Heat Equation with Source





- what if you add heat sources inside Ω?

$$\frac{df}{dt}(x) = g(x) + \Delta f(x)$$

- PDE form: *Poisson's equation*

$$\begin{aligned} \Delta f(x) &= g(x) \qquad && x \in \Omega \\ f(x) &= f_0(x) \qquad && x \in \partial\Omega \end{aligned}$$

- common variational problem:

$$\min_f \int_M \langle \nabla f - \mathbf{v}, \nabla f - \mathbf{v} \rangle \, dA$$

- becomes Poisson problem, $g = \nabla \cdot \mathbf{v}$

Essential Algebraic Properties I

- *linearity*: $$\Delta\left(f(x) + \alpha g(x)\right) = \Delta f(x) + \alpha \Delta g(x)$$

Essential Algebraic Properties I

- *linearity*: $\Delta\left(f(x)+\alpha g(x)\right)=\Delta f(x)+\alpha\Delta g(x)$
- *constants in kernel*: $\Delta\alpha=0$

Essential Algebraic Properties I

- *linearity*: $\Delta\left(f(x) + \alpha g(x)\right) = \Delta f(x) + \alpha \Delta g(x)$
- *constants in kernel*: $\Delta \alpha = 0$

for functions that vanish on ∂M:

- *self-adjoint*: $\int_M f \Delta g \, dA = -\int_M \langle \nabla f, \nabla g \rangle \, dA = \int_M g \Delta f \, dA$
- *negative*: $\int_M f \Delta f \, dA \leq 0$

Essential Algebraic Properties I

- *linearity*: $\Delta\left(f(x)+\alpha g(x)\right)=\Delta f(x)+\alpha\Delta g(x)$
- *constants in kernel*: $\Delta\alpha=0$

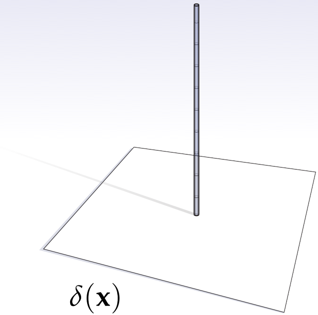
for functions that vanish on ∂M:

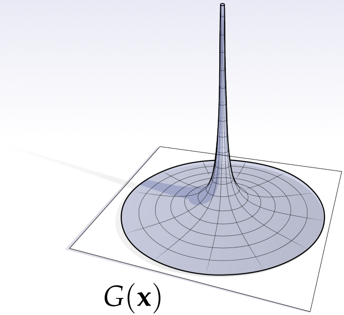
- *self-adjoint*: $\int_M f\,\Delta g\,dA=-\int_M\langle\nabla f,\nabla g\rangle\,dA=\int_M g\Delta f\,dA$
- *negative*: $\int_M f\,\Delta f\,dA\leq 0$

(intuition: $\Delta\approx$ an ∞-*dimensional* negative-semidefinite matrix)

Solving Poisson's Equation with Green's Functions

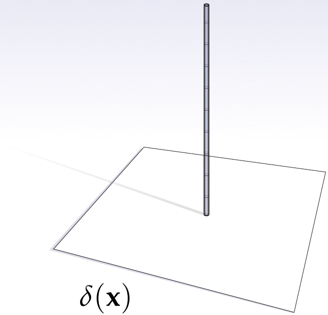
- the *Green's function* G on $\mathbb{R}^2$ solves $\Delta f = g$ for $g = \delta$



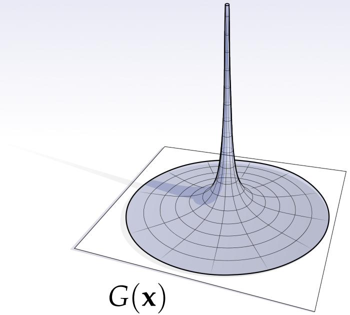


Solving Poisson's Equation with Green's Functions

- the *Green's function* G on $\mathbb{R}^2$ solves $\Delta f = g$ for $g = \delta$
- linearity: if $g = \sum \alpha_i \delta(\mathbf{x} - \mathbf{x}_i)$, $f = \sum \alpha_i G(\mathbf{x} - \mathbf{x}_i)$



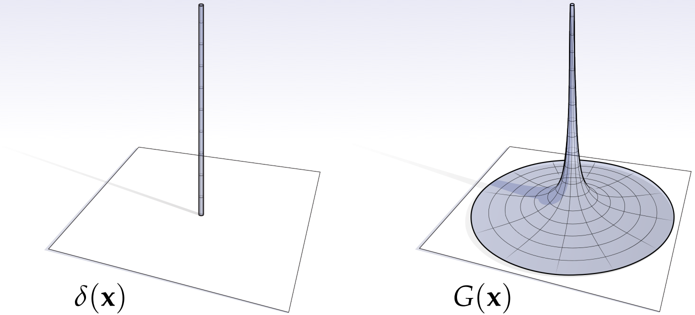
$\delta(\mathbf{x})$



$G(\mathbf{x})$

Solving Poisson's Equation with Green's Functions

- the *Green's function* G on $\mathbb{R}^2$ solves $\Delta f = g$ for $g = \delta$
- linearity: if $g = \sum \alpha_i \delta(\mathbf{x} - \mathbf{x}_i)$, $f = \sum \alpha_i G(\mathbf{x} - \mathbf{x}_i)$
- for any g, $f = G * g$

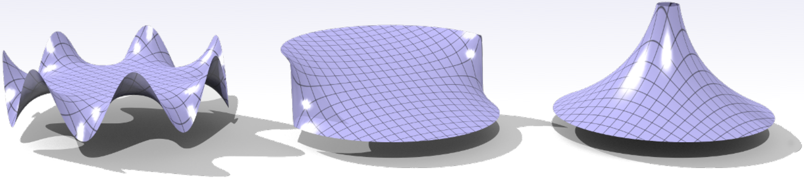


$\delta(\mathbf{x})$ $G(\mathbf{x})$

Essential Algebraic Properties II

a function $f : M \to \mathbb{R}$ with $\Delta f = 0$ is called *harmonic*. Properties:

- f is smooth and analytic



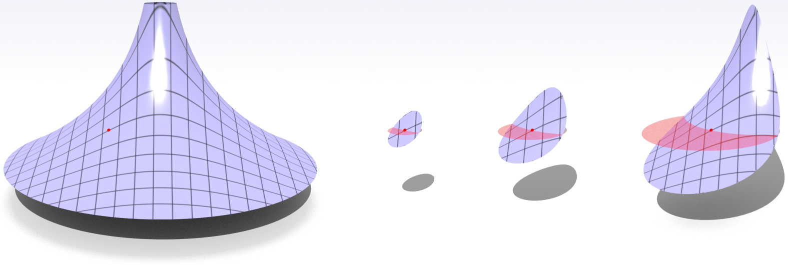
some harmonic $f(x, y)$

Essential Algebraic Properties II

a function $f : M \to \mathbb{R}$ with $\Delta f = 0$ is called *harmonic*. Properties:

- f is smooth and analytic
- $f(x)$ is the *average* of f over any disk around x:

$$f(x) = \frac{1}{\pi r^2} \int_{B(x,r)} f(y)\, dA$$



Essential Algebraic Properties II

a function $f : M \to \mathbb{R}$ with $\Delta f = 0$ is called *harmonic*. Properties:

- f is smooth and analytic
- $f(x)$ is the *average* of f over any disk around x:

$$f(x) = \frac{1}{\pi r^2} \int_{B(x,r)} f(y)\, dA$$

- *maximum principle*: f has no local maxima or minima in M

Essential Algebraic Properties II

a function $f : M \to \mathbb{R}$ with $\Delta f = 0$ is called *harmonic*. Properties:

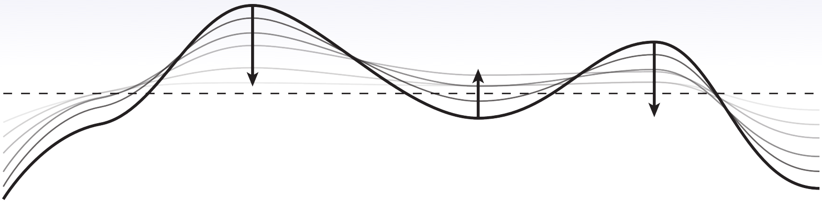
- f is smooth and analytic
- $f(x)$ is the *average* of f over any disk around x:

$$f(x) = \frac{1}{\pi r^2} \int_{B(x,r)} f(y)\, dA$$

- *maximum principle*: f has no local maxima or minima in M
- (can have saddle points)

for a curve $\gamma(u) = (x[u], y[u]) : \mathbb{R} \to \mathbb{R}^2$

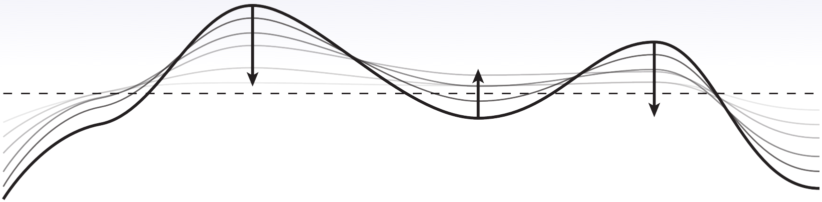
- $\Delta\gamma = (\Delta x, \Delta y)$ is gradient of arc length



Essential Geometric Properties I

for a curve $\gamma(u) = (x[u], y[u]) : \mathbb{R} \to \mathbb{R}^2$

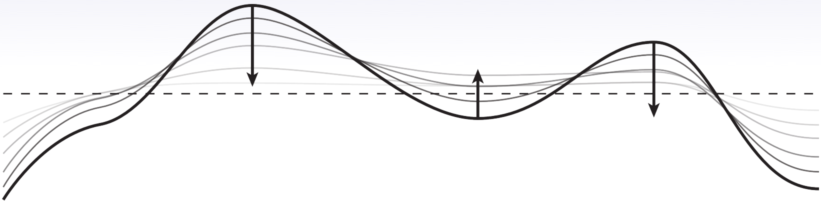
- $\Delta\gamma = (\Delta x, \Delta y)$ is gradient of arc length
- $\Delta\gamma$ is the *curvature normal* $\kappa\hat{n}$



Essential Geometric Properties I

for a curve $\gamma(u) = (x[u], y[u]) : \mathbb{R} \to \mathbb{R}^2$

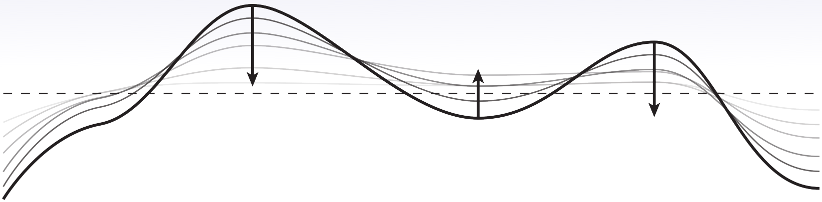
- $\Delta\gamma = (\Delta x, \Delta y)$ is gradient of arc length
- $\Delta\gamma$ is the *curvature normal $\kappa\hat{n}$*
- minimal curves are harmonic



Essential Geometric Properties I

for a curve $\gamma(u) = (x[u], y[u]) : \mathbb{R} \to \mathbb{R}^2$

- $\Delta\gamma = (\Delta x, \Delta y)$ is gradient of arc length
- $\Delta\gamma$ is the *curvature normal* $\kappa\hat{n}$
- minimal curves are harmonic (straight lines)



Essential Geometric Properties II

for a surface $r(u, v) = (x[u, v], y[u, v], z[u, v]) : \mathbb{R} \to \mathbb{R}^3$

- $\Delta r = (\Delta x, \Delta y, \Delta z)$ is gradient of surface area

Essential Geometric Properties II

for a surface $r(u,v) = (x[u,v], y[u,v], z[u,v]) : \mathbb{R} \to \mathbb{R}^3$

- $\Delta r = (\Delta x, \Delta y, \Delta z)$ is gradient of surface area
- Δr is the *mean curvature normal* $2H\hat{n}$

Essential Geometric Properties II

for a surface $r(u,v) = (x[u,v], y[u,v], z[u,v]) : \mathbb{R} \to \mathbb{R}^3$

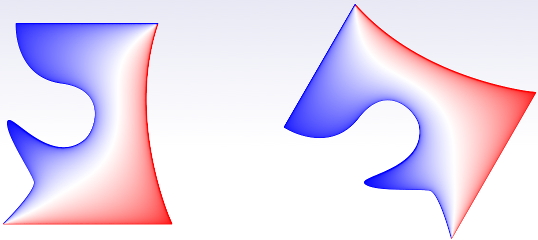
- $\Delta r = (\Delta x, \Delta y, \Delta z)$ is gradient of surface area
- Δr is the *mean curvature normal* $2H\hat{n}$
- minimal surfaces are harmonic!

Essential Geometric Properties III

- Δ is *intrinsic*

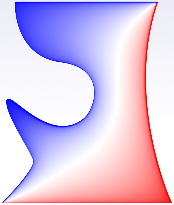
Essential Geometric Properties III

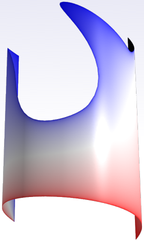
- Δ is *intrinsic*
- for $\Omega \subset \mathbb{R}^2$, rigid motions of Ω don't change Δ



Essential Geometric Properties III

- Δ is *intrinsic*
- for $\Omega \subset \mathbb{R}^2$, rigid motions of Ω don't change Δ
- for a surface Ω, isometric deformations of Ω don't change Δ





Laplacian Spectrum

- ϕ is a (Dirichlet) eigenfunction of Δ on M w/ eigenvalue λ:

$$
\begin{aligned}
\Delta\phi(x) &= \lambda\phi(x), && x \in M \\
0 &= \phi(x), && x \in \partial M \\
1 &= \int_M \langle \phi, \phi \rangle \, dA.
\end{aligned}
$$

Laplacian Spectrum

- ϕ is a (Dirichlet) eigenfunction of Δ on M w/ eigenvalue λ:

$$\begin{aligned} \Delta\phi(x) &= \lambda\phi(x), && x \in M \\ 0 &= \phi(x), && x \in \partial M \\ 1 &= \int_M \langle \phi, \phi \rangle \, dA. \end{aligned}$$

- recall intuition: Δ as ∞-dim negative-semidefinite matrix

Laplacian Spectrum

- ϕ is a (Dirichlet) eigenfunction of Δ on M w/ eigenvalue λ:

$$\begin{aligned} \Delta\phi(x) &= \lambda\phi(x), && x \in M \\ 0 &= \phi(x), && x \in \partial M \\ 1 &= \int_M \langle \phi, \phi \rangle \, dA. \end{aligned}$$

- recall intuition: Δ as ∞-dim negative-semidefinite matrix
- expect orthogonal eigenfunctions with negative eigenvalue

Laplacian Spectrum

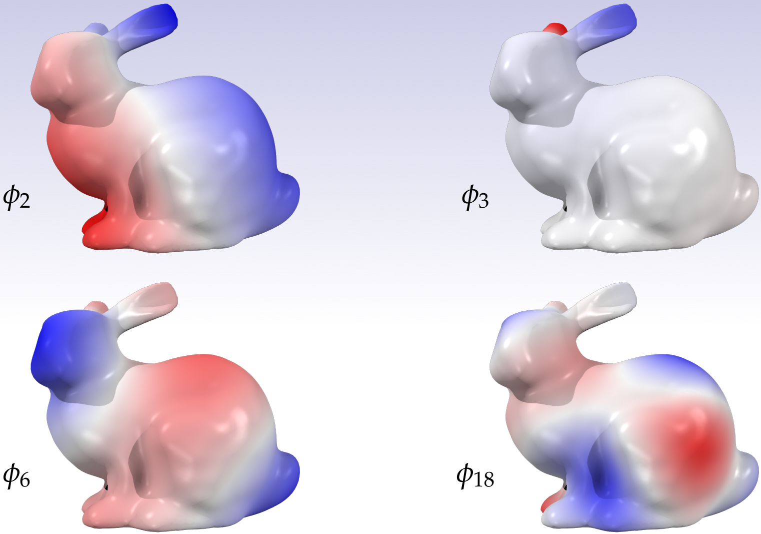
- ϕ is a (Dirichlet) eigenfunction of Δ on M w/ eigenvalue λ:

$$
\begin{aligned}
\Delta\phi(x) &= \lambda\phi(x), && x \in M \\
0 &= \phi(x), && x \in \partial M \\
1 &= \int_M \langle \phi, \phi \rangle \, dA.
\end{aligned}
$$

- recall intuition: Δ as ∞-dim negative-semidefinite matrix
- expect orthogonal eigenfunctions with negative eigenvalue
- spectrum is *discrete*: countably many eigenfunctions,

$$0 \geq \lambda_1 \geq \lambda_2 \geq \lambda_3 \ldots$$

Laplacian Spectrum of Bunny



Laplacian Spectrum: Why do We Care?

- expand function f in eigenbasis:

$$f(x) = \sum_i \alpha_i \phi_i(x)$$

- Dirichlet energy of f:

$$E(f) = \int_M \langle \nabla f, \nabla f \rangle \, dA = -\int_M f \Delta f \, dA = \sum_i \alpha_i^2 \lambda_i$$

Laplacian Spectrum: Why do We Care?

- expand function f in eigenbasis:

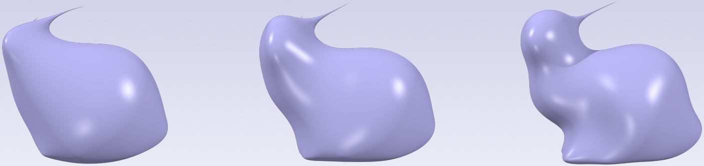
$$f(x) = \sum_i \alpha_i \phi_i(x)$$

- Dirichlet energy of f:

$$E(f) = \int_M \langle \nabla f, \nabla f \rangle \, dA = -\int_M f \Delta f \, dA = \sum_i \alpha_i^2 \lambda_i$$

- large λ_i terms dominate

Laplacian Spectrum: Why do We Care?



10 modes 25 modes 50 modes

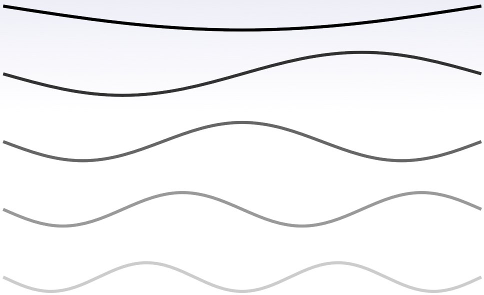
- large λ_i terms dominate

$$f(x) = \underbrace{\sum_{i=1}^{N} \alpha_i \phi_i(x)}_{\text{low-frequency base}} + \underbrace{\sum_{i=N+1}^{\infty} \alpha_i \phi_i(x)}_{\text{high-frequency detail}}$$

Laplacian Spectrum: Special Cases

perhaps you've heard of

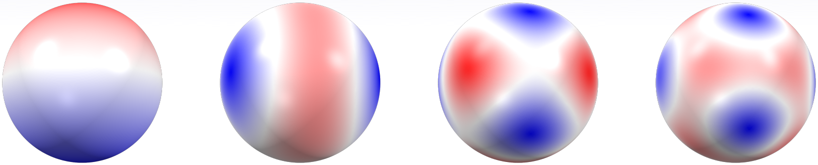
- Fourier basis: $M = \mathbb{R}^n$



Laplacian Spectrum: Special Cases

perhaps you've heard of

- Fourier basis: $M = \mathbb{R}^n$
- spherical harmonics: $M =$ sphere

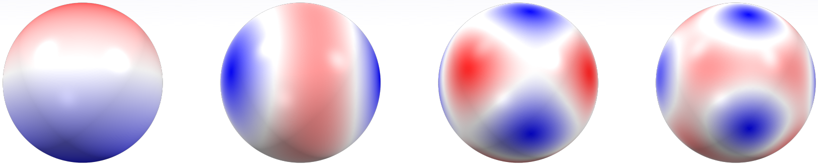


Laplacian Spectrum: Special Cases

perhaps you've heard of

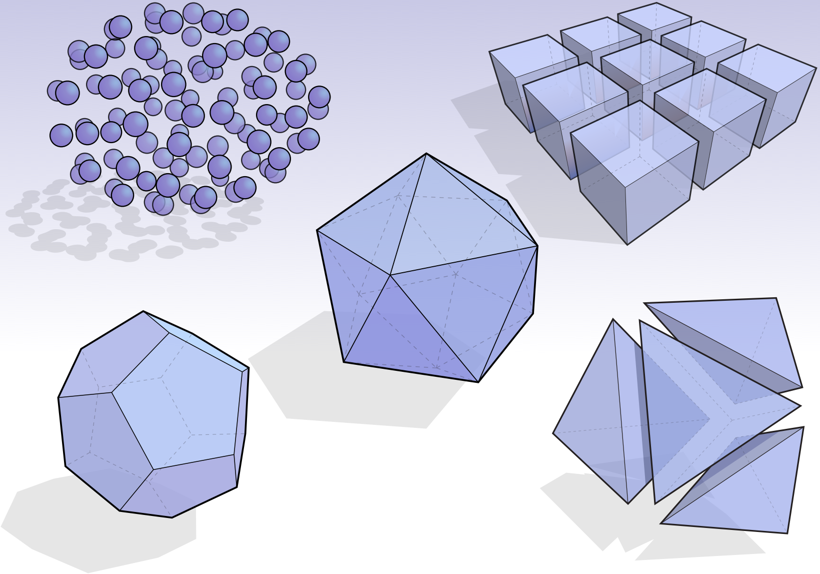
- Fourier basis: $M = \mathbb{R}^n$
- spherical harmonics: $M =$ sphere

Laplacian spectrum generalizes these to any surface

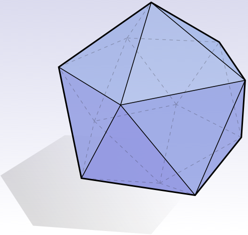


DISCRETIZATION

Discrete Geometry

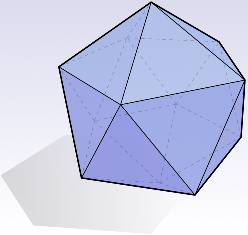


Triangle Meshes



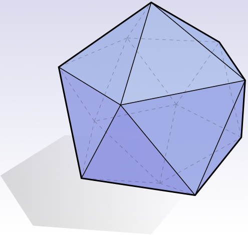
- approximate surface by *triangles*

Triangle Meshes



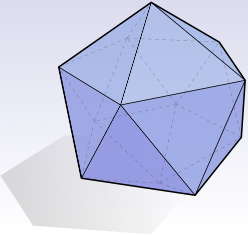
- approximate surface by *triangles*
- “glued together” along edges

Triangle Meshes



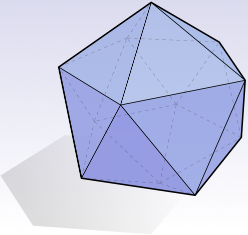
- approximate surface by *triangles*
- “glued together” along edges
- many possible data structures

Triangle Meshes



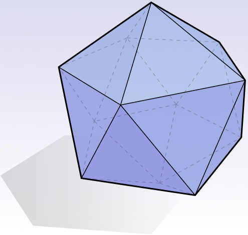
- approximate surface by *triangles*
- "glued together" along edges
- many possible data structures
- *half edge, quad edge, corner table, ...*

Triangle Meshes



- approximate surface by *triangles*
- "glued together" along edges
- many possible data structures
- *half edge, quad edge, corner table, ...*
- for simplicity: *vertex-face adjacency list*

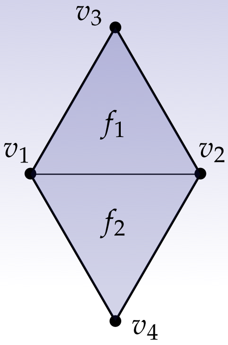
Triangle Meshes



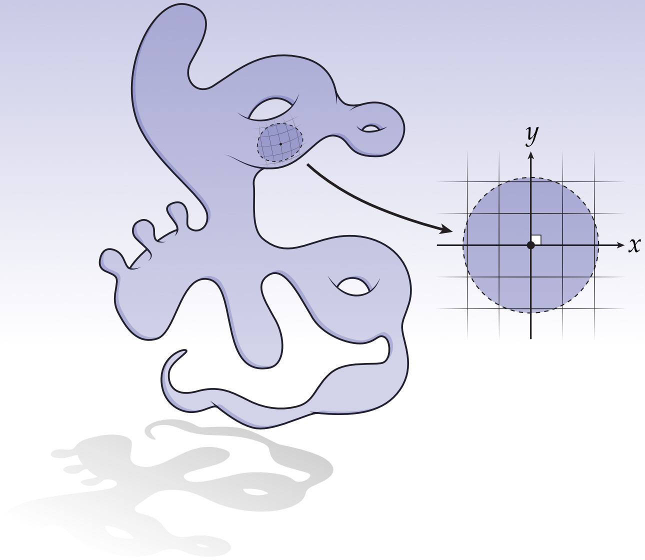
- approximate surface by *triangles*
- "glued together" along edges
- many possible data structures
- *half edge, quad edge, corner table, . . .*
- for simplicity: *vertex-face adjacency list*
- (will be enough for our applications!)

Vertex-Face Adjacency List—Example

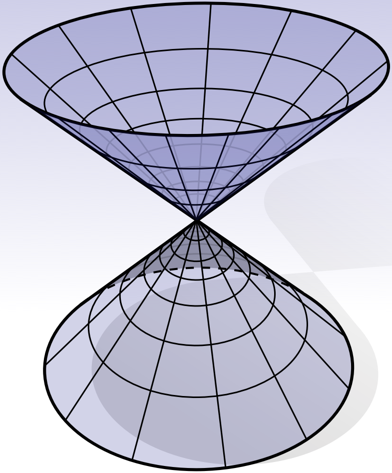
```
# xyz-coordinates of vertices
v 0 0 0
v 1 0 0
v .5 .866 0
v .5 -.866 0

# vertex-face adjacency info
f 1 2 3
f 1 4 2
```



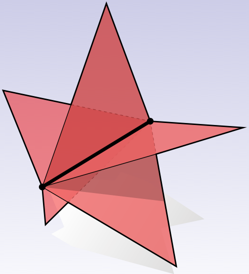
Manifold

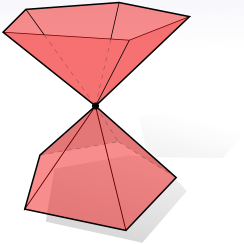


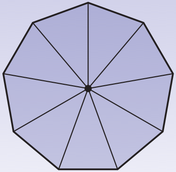
Nonmanifold

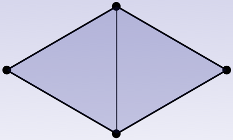


Manifold Triangle Mesh



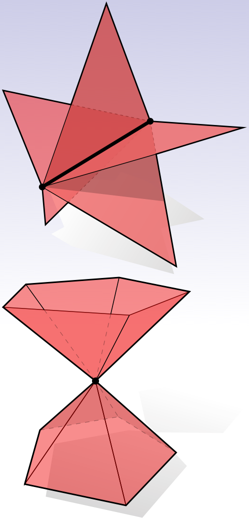


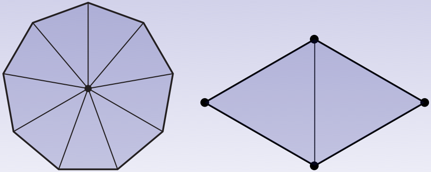




- *manifold* $\Longleftrightarrow$ "locally disk-like"

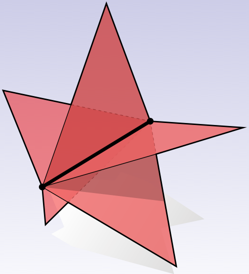
Manifold Triangle Mesh

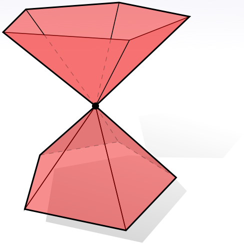


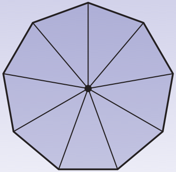


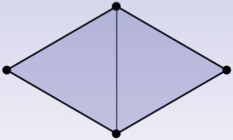
- *manifold* $\iff$ "locally disk-like"
- Which triangle meshes are manifold?

Manifold Triangle Mesh



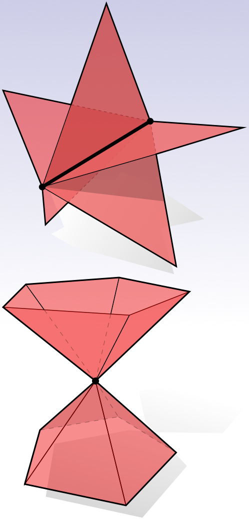


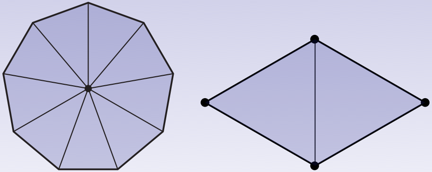




- *manifold* $\Longleftrightarrow$ "locally disk-like"
- Which triangle meshes are manifold?
- Two triangles per edge (no "fins")

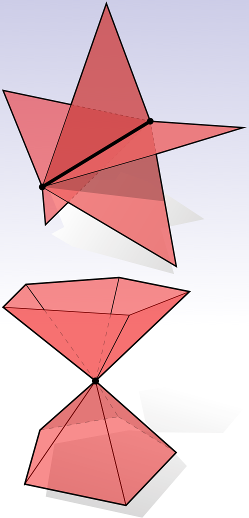
Manifold Triangle Mesh

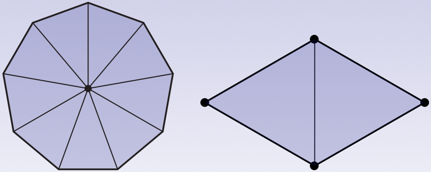




- *manifold* $\iff$ “locally disk-like”
- Which triangle meshes are manifold?
- Two triangles per edge (no “fins”)
- Every vertex looks like a “fan”

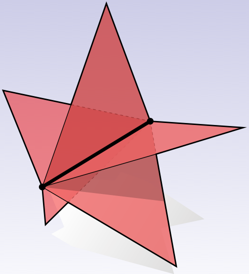
Manifold Triangle Mesh

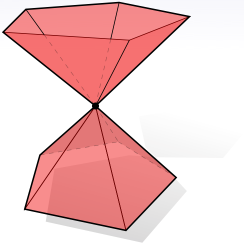


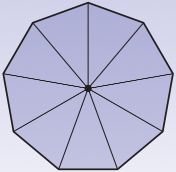


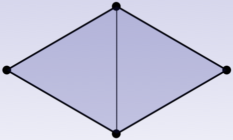
- *manifold* $\iff$ "locally disk-like"
- Which triangle meshes are manifold?
- Two triangles per edge (no "fins")
- Every vertex looks like a "fan"
- Why? *Simplicity.*

Manifold Triangle Mesh



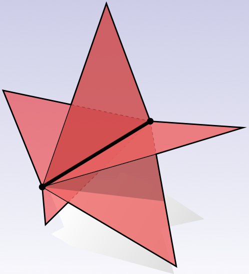


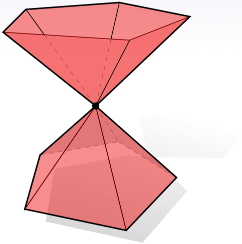


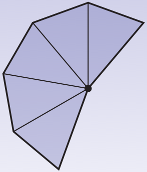


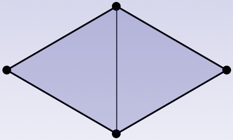
- *manifold* $\iff$ "locally disk-like"
- Which triangle meshes are manifold?
- Two triangles per edge (no "fins")
- Every vertex looks like a "fan"
- Why? *Simplicity.*
- (Sometimes not necessary…)

Manifold Triangle Mesh





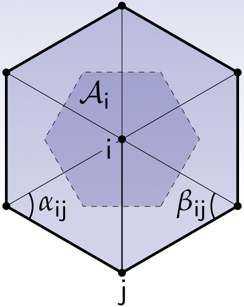




- *manifold* $\iff$ "locally disk-like"
- Which triangle meshes are manifold?
- Two triangles per edge (no "fins")
- Every vertex looks like a "fan"
- Why? *Simplicity.*
- (Sometimes not necessary…)

(Assuming a manifold triangle mesh…)

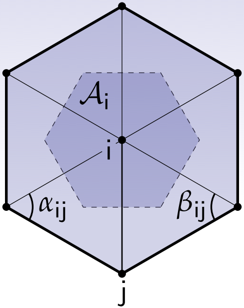
$$(\Delta u)_{\mathsf{i}} \approx \frac{1}{2\mathcal{A}_{\mathsf{i}}} \sum_{\mathsf{j}\in\mathcal{N}(\mathsf{i})} (\cot\alpha_{\mathsf{ij}} + \cot\beta_{\mathsf{ij}})(\mathsf{u_i} - \mathsf{u_j})$$



The Cotangent Laplacian

(Assuming a manifold triangle mesh…)

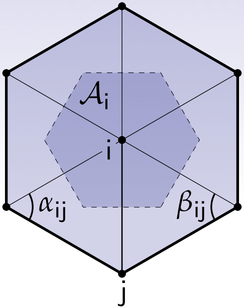
$$(\Delta u)_\mathsf{i} \approx \frac{1}{2\mathcal{A}_\mathsf{i}} \sum_{\mathsf{j}\in\mathcal{N}(\mathsf{i})} (\cot\alpha_{\mathsf{ij}} + \cot\beta_{\mathsf{ij}})(\mathsf{u_i} - \mathsf{u_j})$$



The set $\mathcal{N}(\mathsf{i})$ contains the immediate neighbors of vertex i

(Assuming a manifold triangle mesh…)

$$(\Delta u)_\mathsf{i} \approx \frac{1}{2\mathcal{A}_\mathsf{i}} \sum_{\mathsf{j}\in\mathcal{N}(\mathsf{i})} (\cot \alpha_\mathsf{ij} + \cot \beta_\mathsf{ij})(\mathsf{u_i} - \mathsf{u_j})$$



The set $\mathcal{N}(\mathsf{i})$ contains the immediate neighbors of vertex i

The quantity $\mathcal{A}_\mathsf{i}$ is *vertex area*—for now: 1/3rd of triangle areas

Origin of the Cotan Formula?

- Many different ways to derive it

Origin of the Cotan Formula?

- Many different ways to derive it
  - piecewise linear finite elements (FEM)

Origin of the Cotan Formula?

- Many different ways to derive it
  - piecewise linear finite elements (FEM)
  - finite volumes

Origin of the Cotan Formula?

- Many different ways to derive it
  - piecewise linear finite elements (FEM)
  - finite volumes
  - discrete exterior calculus (DEC)

Origin of the Cotan Formula?

- Many different ways to derive it
  - piecewise linear finite elements (FEM)
  - finite volumes
  - discrete exterior calculus (DEC)
  - …

Origin of the Cotan Formula?

- Many different ways to derive it
  - piecewise linear finite elements (FEM)
  - finite volumes
  - discrete exterior calculus (DEC)
  - …
- Re-derived in many different contexts:
  - mean curvature flow [Desbrun et al., 1999]

Origin of the Cotan Formula?

- Many different ways to derive it
    - piecewise linear finite elements (FEM)
    - finite volumes
    - discrete exterior calculus (DEC)
    - …
- Re-derived in many different contexts:
    - mean curvature flow [Desbrun et al., 1999]
    - minimal surfaces [Pinkall and Polthier, 1993]

Origin of the Cotan Formula?

- Many different ways to derive it
  - piecewise linear finite elements (FEM)
  - finite volumes
  - discrete exterior calculus (DEC)
  - ...
- Re-derived in many different contexts:
  - mean curvature flow [Desbrun et al., 1999]
  - minimal surfaces [Pinkall and Polthier, 1993]
  - electrical networks [Duffin, 1959]

Origin of the Cotan Formula?

- Many different ways to derive it
  - piecewise linear finite elements (FEM)
  - finite volumes
  - discrete exterior calculus (DEC)
  - …
- Re-derived in many different contexts:
  - mean curvature flow [Desbrun et al., 1999]
  - minimal surfaces [Pinkall and Polthier, 1993]
  - electrical networks [Duffin, 1959]
  - Poisson equation [MacNeal, 1949]

Origin of the Cotan Formula?

- Many different ways to derive it
  - piecewise linear finite elements (FEM)
  - finite volumes
  - discrete exterior calculus (DEC)
  - ...
- Re-derived in many different contexts:
  - mean curvature flow [Desbrun et al., 1999]
  - minimal surfaces [Pinkall and Polthier, 1993]
  - electrical networks [Duffin, 1959]
  - Poisson equation [MacNeal, 1949]
  - (Courant? Frankel? *Manhattan Project?*)

Origin of the Cotan Formula?

- Many different ways to derive it
  - piecewise linear finite elements (FEM)
  - finite volumes
  - discrete exterior calculus (DEC)
  - …
- Re-derived in many different contexts:
  - mean curvature flow [Desbrun et al., 1999]
  - minimal surfaces [Pinkall and Polthier, 1993]
  - electrical networks [Duffin, 1959]
  - Poisson equation [MacNeal, 1949]
  - (Courant? Frankel? *Manhattan Project?*)
- *All these different viewpoints yield **exact same** cotan formula*

Origin of the Cotan Formula?

- Many different ways to derive it
  - piecewise linear finite elements (FEM)
  - finite volumes
  - discrete exterior calculus (DEC)
  - ...
- Re-derived in many different contexts:
  - mean curvature flow [Desbrun et al., 1999]
  - minimal surfaces [Pinkall and Polthier, 1993]
  - electrical networks [Duffin, 1959]
  - Poisson equation [MacNeal, 1949]
  - (Courant? Frankel? *Manhattan Project?*)
- *All these different viewpoints yield **exact same** cotan formula*
- For three different derivations, see [Crane et al., 2013a]

MacNeal, 1949

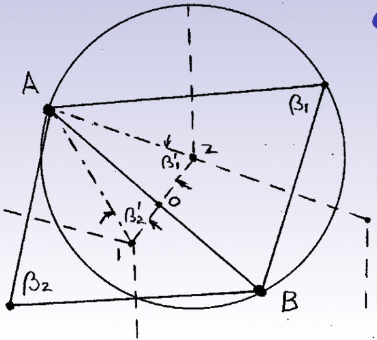
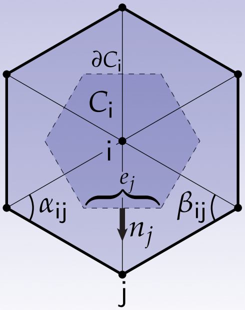


Fig. 25.

"If the network is first laid out on a large sheet of drawing paper, the angles can be measured with a protractor and the distances scaled off with sufficient accuracy in a short time."

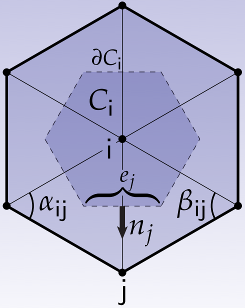
"If the mesh is sufficiently fine, this will not lead to a large error. It indicates, however, that an attempt should be made to keep the triangles as nearly regular as possible."

Cotan-Laplacian via Finite Volumes



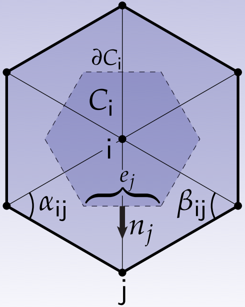
- Integrate over each dual cell C_i

Cotan-Laplacian via Finite Volumes



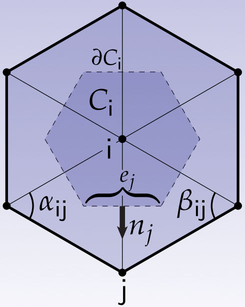
- Integrate over each dual cell C_i
- $\int_{C_i} \Delta u = \int_{C_i} f$ ("weak")

Cotan-Laplacian via Finite Volumes



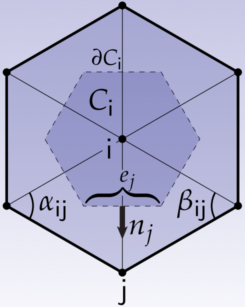
- Integrate over each dual cell C_i
- $\int_{C_i} \Delta u = \int_{C_i} f$ ("weak")
- Right-hand side approximated as $\mathcal{A}_i f_i$

Cotan-Laplacian via Finite Volumes



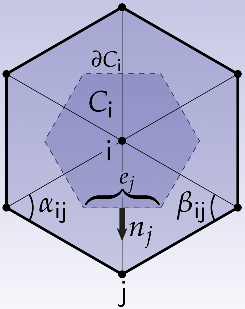
- Integrate over each dual cell C_i
- $\int_{C_i} \Delta u = \int_{C_i} f$ ("weak")
- Right-hand side approximated as $\mathcal{A}_i f_i$
- Left-hand side becomes $\int_{C_i} \nabla \cdot \nabla u = \int_{\partial C_i} n \cdot \nabla u$ (Stokes')

Cotan-Laplacian via Finite Volumes



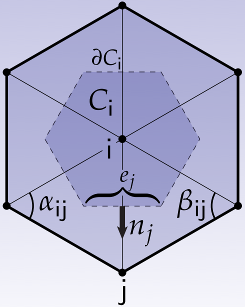
- Integrate over each dual cell C_i
- $\int_{C_i} \Delta u = \int_{C_i} f$ ("weak")
- Right-hand side approximated as $\mathcal{A}_i f_i$
- Left-hand side becomes $\int_{C_i} \nabla \cdot \nabla u = \int_{\partial C_i} n \cdot \nabla u$ (Stokes')
- Get piecewise integral over boundary $\sum_{e_j \in \partial C_i} \int_{e_j} n_j \cdot \nabla u$

Cotan-Laplacian via Finite Volumes



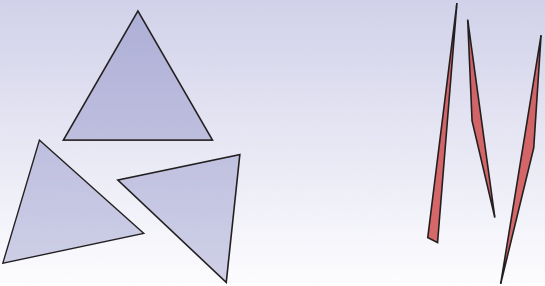
- Integrate over each dual cell C_i
- $\int_{C_i} \Delta u = \int_{C_i} f$ ("weak")
- Right-hand side approximated as $\mathcal{A}_i f_i$
- Left-hand side becomes $\int_{C_i} \nabla \cdot \nabla u = \int_{\partial C_i} n \cdot \nabla u$ (Stokes')
- Get piecewise integral over boundary $\sum_{e_j \in \partial C_i} \int_{e_j} n_j \cdot \nabla u$
- After some trigonometry: $\frac{1}{2} \sum_{j \in \mathcal{N}(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(u_i - u_j)$

Cotan-Laplacian via Finite Volumes



- Integrate over each dual cell C_i
- $\int_{C_i} \Delta u = \int_{C_i} f$ ("weak")
- Right-hand side approximated as $\mathcal{A}_i f_i$
- Left-hand side becomes $\int_{C_i} \nabla \cdot \nabla u = \int_{\partial C_i} n \cdot \nabla u$ (Stokes')
- Get piecewise integral over boundary $\sum_{e_j \in \partial C_i} \int_{e_j} n_j \cdot \nabla u$
- After some trigonometry: $\frac{1}{2} \sum_{j \in \mathcal{N}(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(u_i - u_j)$
- (Can divide by $\mathcal{A}_i$ to approximate *pointwise* value)

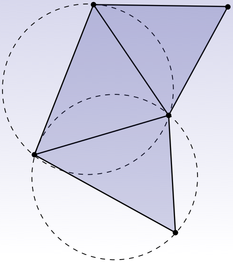
Triangle Quality—Rule of Thumb



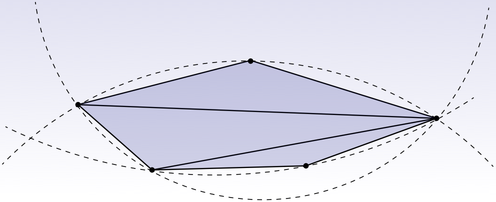
good triangles bad triangles

(For further discussion see Shewchuk, *"What Is a Good Linear Finite Element?"*)

Triangle Quality—Delaunay Property

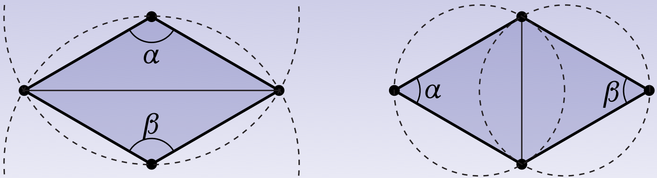


Delaunay



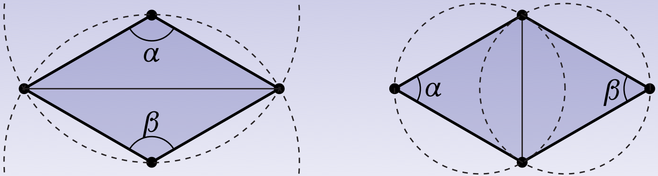
Not Delaunay

Local Mesh Improvement



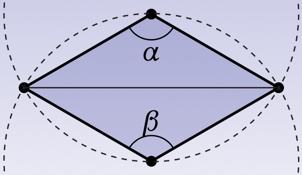
- Some simple ways to improve quality of Laplacian

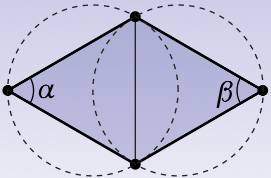
Local Mesh Improvement



- Some simple ways to improve quality of Laplacian
- E.g., if $\alpha + \beta > \pi$, "flip" the edge; after enough flips, mesh will be Delaunay [Bobenko and Springborn, 2005]

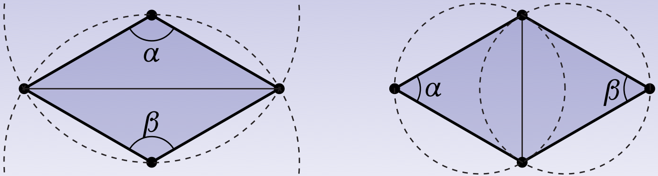
Local Mesh Improvement





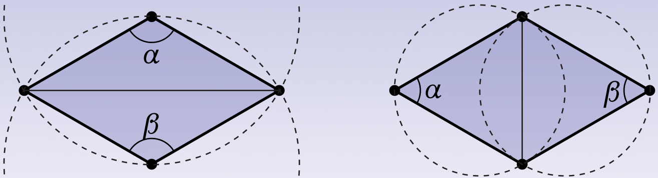
- Some simple ways to improve quality of Laplacian
- E.g., if $\alpha + \beta > \pi$, "flip" the edge; after enough flips, mesh will be Delaunay [Bobenko and Springborn, 2005]
- Other ways to improve mesh (edge collapse, edge split, ...)

Local Mesh Improvement



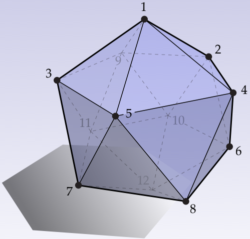
- Some simple ways to improve quality of Laplacian
- E.g., if $\alpha + \beta > \pi$, "flip" the edge; after enough flips, mesh will be Delaunay [Bobenko and Springborn, 2005]
- Other ways to improve mesh (edge collapse, edge split, ...)
- Particular interest recently in *interface tracking*

Local Mesh Improvement



- Some simple ways to improve quality of Laplacian
- E.g., if $\alpha + \beta > \pi$, "flip" the edge; after enough flips, mesh will be Delaunay [Bobenko and Springborn, 2005]
- Other ways to improve mesh (edge collapse, edge split, ...)
- Particular interest recently in *interface tracking*
- For more, see [Dunyach et al., 2013, Wojtan et al., 2011].

Meshes and Matrices

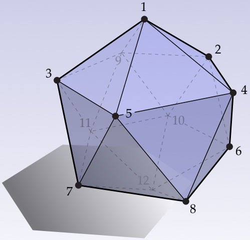


$$
\begin{bmatrix}
-5 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & -5 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & -5 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & -5 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & -5 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & -5 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -5 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -5 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -5 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & -5
\end{bmatrix}
$$

(Laplace matrix, ***ignoring weights!***)

- So far, Laplacian expressed as a sum:

Meshes and Matrices

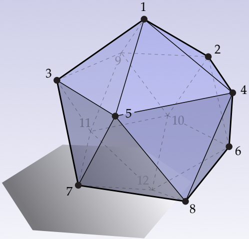


- So far, Laplacian expressed as a sum:
- $\frac{1}{2}\sum_{j \in \mathcal{N}(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(u_j - u_i)$

$$\begin{bmatrix}
-5 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & -5 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & -5 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & -5 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & -5 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & -5 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -5 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -5 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -5 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & -5
\end{bmatrix}$$

(Laplace matrix, ***ignoring weights!***)

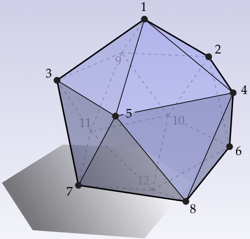
Meshes and Matrices



$$\begin{bmatrix} -5 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & -5 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & -5 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & -5 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & -5 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & -5 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -5 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -5 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -5 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & -5 \end{bmatrix}$$

(Laplace matrix, ***ignoring weights!***)

- So far, Laplacian expressed as a sum:
- $\frac{1}{2}\sum_{j\in\mathcal{N}(i)}(\cot\alpha_{ij}+\cot\beta_{ij})(u_j-u_i)$
- For computation, encode using *matrices*

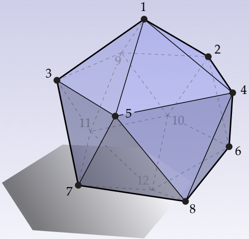


- So far, Laplacian expressed as a sum:
- $\frac{1}{2}\sum_{j\in\mathcal{N}(i)}(\cot\alpha_{ij}+\cot\beta_{ij})(u_j-u_i)$
- For computation, encode using *matrices*
- First, give each vertex an index $1,\ldots,|V|$

$$\begin{bmatrix}
-5 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & -5 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & -5 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & -5 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & -5 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & -5 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -5 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -5 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -5 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & -5
\end{bmatrix}$$

(Laplace matrix, ***ignoring weights!***)

Meshes and Matrices

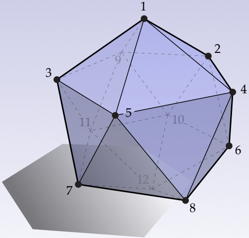


$$\begin{bmatrix}
-5 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & -5 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & -5 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & -5 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & -5 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & -5 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -5 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -5 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -5 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & -5
\end{bmatrix}$$

(Laplace matrix, ***ignoring weights!***)

- So far, Laplacian expressed as a sum:
- $\frac{1}{2}\sum_{\mathsf{j}\in\mathcal{N}(\mathsf{i})}(\cot\alpha_{\mathsf{ij}} + \cot\beta_{\mathsf{ij}})(\mathsf{u_j} - \mathsf{u_i})$
- For computation, encode using *matrices*
- First, give each vertex an index $1, \ldots, |V|$
- *Weak Laplacian* is matrix $\mathsf{C} \in \mathbb{R}^{|V|\times|V|}$

Meshes and Matrices

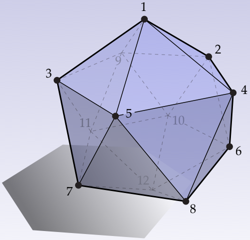


- So far, Laplacian expressed as a sum:
- $\frac{1}{2}\sum_{j\in\mathcal{N}(i)}(\cot\alpha_{ij}+\cot\beta_{ij})(u_j-u_i)$
- For computation, encode using *matrices*
- First, give each vertex an index $1,\ldots,|V|$
- *Weak Laplacian* is matrix $C\in\mathbb{R}^{|V|\times|V|}$
- Row i represents sum for ith vertex

$$\begin{bmatrix}
-5 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\
1 & -5 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0\\
1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0\\
1 & 1 & 0 & -5 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0\\
1 & 0 & 1 & 1 & -5 & 0 & 1 & 1 & 0 & 0 & 0 & 0\\
0 & 1 & 0 & 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1\\
0 & 0 & 1 & 0 & 1 & 0 & -5 & 1 & 0 & 0 & 1 & 1\\
0 & 0 & 0 & 1 & 1 & 1 & 1 & -5 & 0 & 0 & 0 & 1\\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -5 & 1 & 1 & 0\\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -5 & 1 & 1\\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -5 & 1\\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & -5
\end{bmatrix}$$

(Laplace matrix, ***ignoring weights!***)

Meshes and Matrices

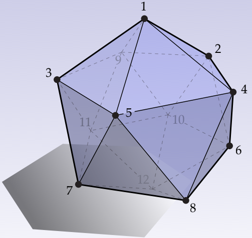


$$\begin{bmatrix}
-5 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & -5 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & -5 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & -5 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & -5 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & -5 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -5 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -5 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -5 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & -5
\end{bmatrix}$$

(Laplace matrix, ***ignoring weights!***)

- So far, Laplacian expressed as a sum:
- $\frac{1}{2}\sum_{j \in \mathcal{N}(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(u_j - u_i)$
- For computation, encode using *matrices*
- First, give each vertex an index $1, \ldots, |V|$
- *Weak Laplacian* is matrix $C \in \mathbb{R}^{|V| \times |V|}$
- Row i represents sum for ith vertex
  - $C_{ij} = \frac{1}{2} \cot \alpha_{ij} + \cot \beta_{ij}$ for $j \in \mathcal{N}(i)$

Meshes and Matrices

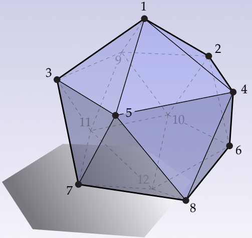


$$\begin{bmatrix}
-5 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & -5 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & -5 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & -5 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & -5 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & -5 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -5 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -5 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -5 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & -5
\end{bmatrix}$$

(Laplace matrix, ***ignoring weights!***)

- So far, Laplacian expressed as a sum:
- $\frac{1}{2}\sum_{j \in \mathcal{N}(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(u_j - u_i)$
- For computation, encode using *matrices*
- First, give each vertex an index $1, \ldots, |V|$
- *Weak Laplacian* is matrix $C \in \mathbb{R}^{|V| \times |V|}$
- Row i represents sum for ith vertex
  - $C_{ij} = \frac{1}{2} \cot \alpha_{ij} + \cot \beta_{ij}$ for $j \in \mathcal{N}(i)$
  - $C_{ii} = -\sum_{j \in \mathcal{N}(i)} C_{ij}$

Meshes and Matrices

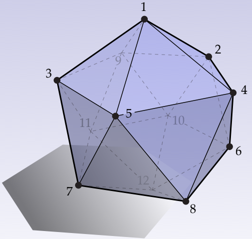


$$\begin{bmatrix} -5 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & -5 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & -5 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & -5 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & -5 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & -5 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -5 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -5 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -5 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & -5 \end{bmatrix}$$

(Laplace matrix, ***ignoring weights!***)

- So far, Laplacian expressed as a sum:
- $\frac{1}{2}\sum_{j\in\mathcal{N}(i)}(\cot\alpha_{ij} + \cot\beta_{ij})(u_j - u_i)$
- For computation, encode using *matrices*
- First, give each vertex an index $1, \ldots, |V|$
- *Weak Laplacian* is matrix $C \in \mathbb{R}^{|V|\times|V|}$
- Row i represents sum for ith vertex
  - $C_{ij} = \frac{1}{2}\cot\alpha_{ij} + \cot\beta_{ij}$ for $j \in \mathcal{N}(i)$
  - $C_{ii} = -\sum_{j\in\mathcal{N}(i)} C_{ij}$
- All other entries are zero

Meshes and Matrices

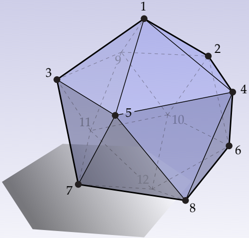


$$
\begin{bmatrix}
-5 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & -5 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & -5 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & -5 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & -5 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & -5 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -5 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -5 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -5 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & -5
\end{bmatrix}
$$

(Laplace matrix, ***ignoring weights!***)

- So far, Laplacian expressed as a sum:
- $\frac{1}{2}\sum_{j\in\mathcal{N}(i)}(\cot\alpha_{ij} + \cot\beta_{ij})(u_j - u_i)$
- For computation, encode using *matrices*
- First, give each vertex an index $1, \ldots, |V|$
- *Weak Laplacian* is matrix $C \in \mathbb{R}^{|V|\times|V|}$
- Row i represents sum for ith vertex
  - $C_{ij} = \frac{1}{2}\cot\alpha_{ij} + \cot\beta_{ij}$ for $j \in \mathcal{N}(i)$
  - $C_{ii} = -\sum_{j\in\mathcal{N}(i)} C_{ij}$
- All other entries are zero
- *Use **sparse** matrices!*

Meshes and Matrices



$$\begin{bmatrix}
-5 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & -5 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & -5 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & -5 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & -5 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & -5 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & -5 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -5 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -5 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -5 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & -5
\end{bmatrix}$$

(Laplace matrix, ***ignoring weights!***)

- So far, Laplacian expressed as a sum:
- $\frac{1}{2}\sum_{j\in\mathcal{N}(i)}(\cot\alpha_{ij} + \cot\beta_{ij})(u_j - u_i)$
- For computation, encode using *matrices*
- First, give each vertex an index $1,\ldots,|V|$
- *Weak Laplacian* is matrix $C \in \mathbb{R}^{|V|\times|V|}$
- Row i represents sum for ith vertex
  - $C_{ij} = \frac{1}{2}\cot\alpha_{ij} + \cot\beta_{ij}$ for $j \in \mathcal{N}(i)$
  - $C_{ii} = -\sum_{j\in\mathcal{N}(i)} C_{ij}$
- All other entries are zero
- *Use **sparse** matrices!*
- (MATLAB: `sparse`, SuiteSparse: `cholmod_sparse`, Eigen: `SparseMatrix`)

Mass Matrix

- Matrix C encodes only *part* of Laplacian—recall that

$$(\Delta u)_{\mathsf{i}} = \frac{1}{2\mathcal{A}_{\mathsf{i}}} \sum_{\mathsf{j} \in \mathcal{N}(\mathsf{i})} (\cot \alpha_{\mathsf{ij}} + \cot \beta_{\mathsf{ij}})(\mathsf{u}_{\mathsf{j}} - \mathsf{u}_{\mathsf{i}})$$

Mass Matrix

- Matrix C encodes only *part* of Laplacian—recall that

$$(\Delta u)_{\mathsf{i}} = \frac{1}{2\mathcal{A}_{\mathsf{i}}} \sum_{\mathsf{j} \in \mathcal{N}(\mathsf{i})} (\cot \alpha_{\mathsf{ij}} + \cot \beta_{\mathsf{ij}})(\mathsf{u}_{\mathsf{j}} - \mathsf{u}_{\mathsf{i}})$$

- Still need to incorporate vertex areas $\mathcal{A}_{\mathsf{i}}$

Mass Matrix

- Matrix C encodes only *part* of Laplacian—recall that

$$(\Delta u)_{\mathsf{i}} = \frac{1}{2\mathcal{A}_{\mathsf{i}}} \sum_{\mathsf{j} \in \mathcal{N}(\mathsf{i})} (\cot \alpha_{\mathsf{ij}} + \cot \beta_{\mathsf{ij}})(\mathsf{u_j} - \mathsf{u_i})$$

- Still need to incorporate vertex areas $\mathcal{A}_{\mathsf{i}}$
- For convenience, build diagonal *mass matrix* $\mathsf{M} \in \mathbb{R}^{|V| \times |V|}$:

$$\mathsf{M} = \begin{bmatrix} \mathcal{A}_1 & & \\ & \ddots & \\ & & \mathcal{A}_{|V|} \end{bmatrix}$$

Mass Matrix

- Matrix C encodes only *part* of Laplacian—recall that

$$(\Delta u)_{\mathsf{i}} = \frac{1}{2\mathcal{A}_{\mathsf{i}}} \sum_{\mathsf{j} \in \mathcal{N}(\mathsf{i})} (\cot \alpha_{\mathsf{ij}} + \cot \beta_{\mathsf{ij}})(\mathsf{u_j} - \mathsf{u_i})$$

- Still need to incorporate vertex areas $\mathcal{A}_{\mathsf{i}}$
- For convenience, build diagonal *mass matrix* $\mathsf{M} \in \mathbb{R}^{|V| \times |V|}$:

$$\mathsf{M} = \begin{bmatrix} \mathcal{A}_1 & & \\ & \ddots & \\ & & \mathcal{A}_{|V|} \end{bmatrix}$$

- Entries are just $\mathsf{M_{ii}} = \mathcal{A}_{\mathsf{i}}$ (all other entries are zero)

Mass Matrix

- Matrix C encodes only *part* of Laplacian—recall that

$$(\Delta u)_{\mathsf{i}} = \frac{1}{2\mathcal{A}_{\mathsf{i}}}\sum_{\mathsf{j}\in\mathcal{N}(\mathsf{i})}(\cot\alpha_{\mathsf{ij}} + \cot\beta_{\mathsf{ij}})(\mathsf{u}_{\mathsf{j}} - \mathsf{u}_{\mathsf{i}})$$

- Still need to incorporate vertex areas $\mathcal{A}_{\mathsf{i}}$
- For convenience, build diagonal *mass matrix* $\mathsf{M} \in \mathbb{R}^{|V|\times|V|}$:

$$\mathsf{M} = \begin{bmatrix} \mathcal{A}_1 & & \\ & \ddots & \\ & & \mathcal{A}_{|V|} \end{bmatrix}$$

- Entries are just $\mathsf{M}_{\mathsf{ii}} = \mathcal{A}_{\mathsf{i}}$ (all other entries are zero)
- Laplace operator is then $\mathsf{L} := \mathsf{M}^{-1}\mathsf{C}$

Mass Matrix

- Matrix C encodes only *part* of Laplacian—recall that

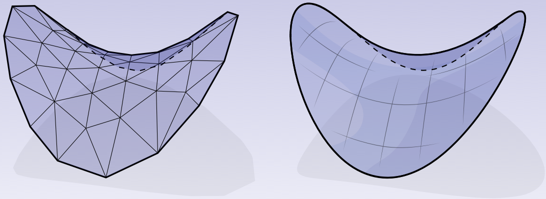
$$(\Delta u)_{\mathsf{i}} = \frac{1}{2\mathcal{A}_{\mathsf{i}}} \sum_{\mathsf{j}\in\mathcal{N}(\mathsf{i})} (\cot\alpha_{\mathsf{ij}} + \cot\beta_{\mathsf{ij}})(\mathsf{u_j} - \mathsf{u_i})$$

- Still need to incorporate vertex areas $\mathcal{A}_{\mathsf{i}}$
- For convenience, build diagonal *mass matrix* $\mathsf{M} \in \mathbb{R}^{|V|\times|V|}$:

$$\mathsf{M} = \begin{bmatrix} \mathcal{A}_1 & & \\ & \ddots & \\ & & \mathcal{A}_{|V|} \end{bmatrix}$$

- Entries are just $\mathsf{M_{ii}} = \mathcal{A}_{\mathsf{i}}$ (all other entries are zero)
- Laplace operator is then $\mathsf{L} := \mathsf{M}^{-1}\mathsf{C}$
- Applying L to a column vector $u \in \mathbb{R}^{|V|}$ "implements" the cotan formula shown above

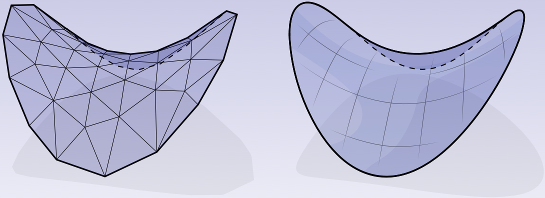
Discrete Poisson / Laplace Equation



- Poisson equation $\Delta u = f$ becomes linear algebra problem:

$$\mathsf{L}\mathsf{u} = \mathsf{f}$$

Discrete Poisson / Laplace Equation

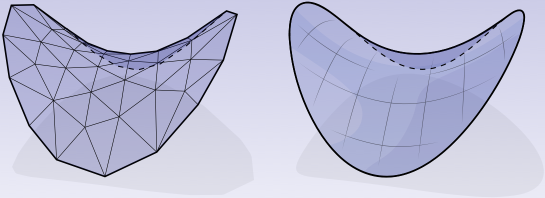


- Poisson equation $\Delta u = f$ becomes linear algebra problem:

$$\mathsf{L}\mathsf{u} = \mathsf{f}$$

- Vector $\mathsf{f} \in \mathbb{R}^{|V|}$ is given data; $\mathsf{u} \in \mathbb{R}^{|V|}$ is unknown.

Discrete Poisson / Laplace Equation

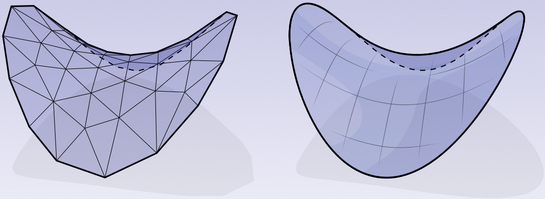


- Poisson equation $\Delta u = f$ becomes linear algebra problem:

$$\mathsf{Lu} = \mathsf{f}$$

- Vector $\mathsf{f} \in \mathbb{R}^{|V|}$ is given data; $\mathsf{u} \in \mathbb{R}^{|V|}$ is unknown.
- Discrete approximation u approaches smooth solution u as mesh is refined (for smooth data, "good" meshes…).

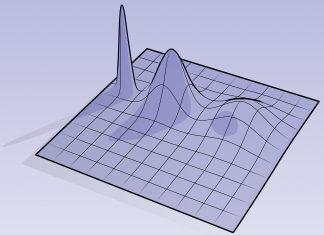
Discrete Poisson / Laplace Equation



- Poisson equation $\Delta u = f$ becomes linear algebra problem:

$$\mathsf{L}\mathsf{u} = \mathsf{f}$$

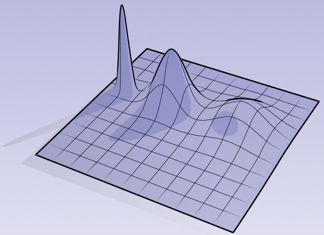
- Vector $\mathsf{f} \in \mathbb{R}^{|V|}$ is given data; $\mathsf{u} \in \mathbb{R}^{|V|}$ is unknown.
- Discrete approximation u approaches smooth solution u as mesh is refined (for smooth data, "good" meshes…).
- Laplace is just Poisson with "zero" on right hand side!



Discrete Heat Equation

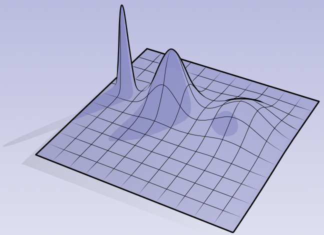
- Heat equation $\frac{du}{dt} = \Delta u$ must also be discretized in *time*

Discrete Heat Equation



- Heat equation $\frac{du}{dt} = \Delta u$ must also be discretized in *time*
- Replace time derivative with *finite difference*:

$$\frac{du}{dt} \Rightarrow \frac{\mathsf{u}_{\mathsf{k}+1} - \mathsf{u}_{\mathsf{k}}}{\mathsf{h}}, \quad \underbrace{\mathsf{h} > 0}_{\text{“time step”}}$$

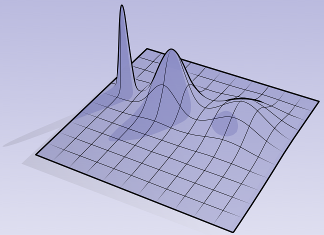


Discrete Heat Equation

- Heat equation $\frac{du}{dt} = \Delta u$ must also be discretized in *time*
- Replace time derivative with *finite difference*:

$$\frac{du}{dt} \Rightarrow \frac{\mathsf{u}_{\mathsf{k}+1} - \mathsf{u}_{\mathsf{k}}}{\mathsf{h}}, \quad \underbrace{\mathsf{h} > 0}_{\text{“time step”}}$$

- How (or really, “when”) do we approximate Δu?

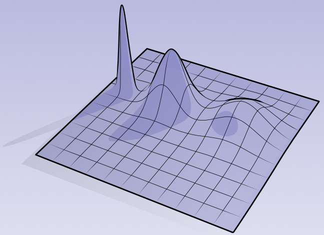


Discrete Heat Equation

- Heat equation $\frac{du}{dt} = \Delta u$ must also be discretized in *time*
- Replace time derivative with *finite difference*:

$$\frac{du}{dt} \Rightarrow \frac{\mathsf{u}_{k+1} - \mathsf{u}_k}{\mathsf{h}}, \quad \underbrace{\mathsf{h} > 0}_{\text{“time step”}}$$

- How (or really, “when”) do we approximate Δu?
- *Explicit:* $(\mathsf{u}_{k+1} - \mathsf{u}_k)/\mathsf{h} = \mathsf{L}\mathsf{u}_k$ (cheaper to compute)

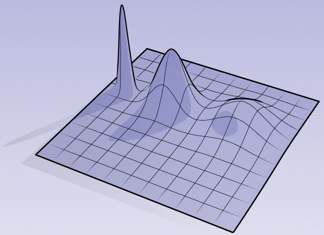


Discrete Heat Equation

- Heat equation $\frac{du}{dt} = \Delta u$ must also be discretized in *time*
- Replace time derivative with *finite difference*:

$$\frac{du}{dt} \Rightarrow \frac{\mathsf{u}_{k+1} - \mathsf{u}_k}{\mathsf{h}}, \quad \underbrace{\mathsf{h} > 0}_{\text{“time step”}}$$

- How (or really, “when”) do we approximate Δu?
- *Explicit:* $(\mathsf{u}_{k+1} - \mathsf{u}_k)/\mathsf{h} = \mathsf{L}\mathsf{u}_k$ (cheaper to compute)
- *Implicit:* $(\mathsf{u}_{k+1} - \mathsf{u}_k)/\mathsf{h} = \mathsf{L}\mathsf{u}_{k+1}$ (more *stable*)

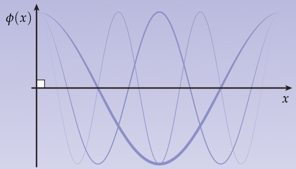


Discrete Heat Equation

- Heat equation $\frac{du}{dt} = \Delta u$ must also be discretized in *time*
- Replace time derivative with *finite difference*:

$$\frac{du}{dt} \Rightarrow \frac{\mathsf{u}_{\mathsf{k}+1} - \mathsf{u}_{\mathsf{k}}}{\mathsf{h}}, \quad \underbrace{\mathsf{h} > 0}_{\text{“time step”}}$$

- How (or really, “when”) do we approximate Δu?
- *Explicit:* $(\mathsf{u}_{\mathsf{k}+1} - \mathsf{u}_{\mathsf{k}})/\mathsf{h} = \mathsf{L}\mathsf{u}_{\mathsf{k}}$ (cheaper to compute)
- *Implicit:* $(\mathsf{u}_{\mathsf{k}+1} - \mathsf{u}_{\mathsf{k}})/\mathsf{h} = \mathsf{L}\mathsf{u}_{\mathsf{k}+1}$ (more *stable*)
- Implicit update becomes linear system $(\mathsf{I} - \mathsf{h}\mathsf{L})\mathsf{u}_{\mathsf{k}+1} = \mathsf{u}_{\mathsf{k}}$

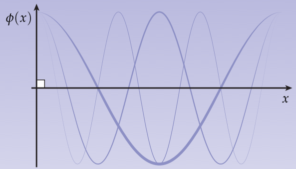


Discrete Eigenvalue Problem

- Smallest eigenvalue problem $\Delta u = \lambda u$ becomes

$$\mathsf{L}\mathsf{u} = \lambda \mathsf{u}$$

for smallest nonzero eigenvalue λ.

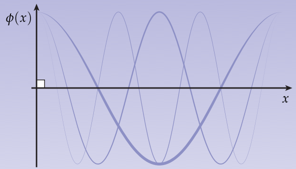


Discrete Eigenvalue Problem

- Smallest eigenvalue problem $\Delta u = \lambda u$ becomes

$$\mathsf{L}\mathsf{u} = \lambda \mathsf{u}$$

  for smallest nonzero eigenvalue λ.
- Can be solved using *(inverse) power method*:
  - Pick random u_0
  - Until convergence:
    - Solve $\mathsf{L}\mathsf{u}_{k+1} = \mathsf{u}_k$
    - Remove mean value from u_{k+1}
    - $\mathsf{u}_{k+1} \leftarrow \mathsf{u}_{k+1} / |\mathsf{u}_{k+1}|$



Discrete Eigenvalue Problem

- Smallest eigenvalue problem $\Delta u = \lambda u$ becomes

$$\mathsf{L}\mathsf{u} = \lambda \mathsf{u}$$

  for smallest nonzero eigenvalue λ.
- Can be solved using *(inverse) power method*:
  - Pick random u_0
  - Until convergence:
    - Solve $\mathsf{L}\mathsf{u}_{k+1} = \mathsf{u}_k$
    - Remove mean value from u_{k+1}
    - $\mathsf{u}_{k+1} \leftarrow \mathsf{u}_{k+1}/|\mathsf{u}_{k+1}|$
- By ***prefactoring*** L, overall cost is nearly identical to solving a single Poisson equation!

Properties of cotan-Laplace

- *Always, always, always* positive-semidefinite $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f} \geq 0$ (even if cotan weights are negative!)

Properties of cotan-Laplace

- *Always, always, always* positive-semidefinite $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f} \geq 0$ (even if cotan weights are negative!)
- Why? $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f}$ is *identical* to summing $||\nabla f||^2$!

Properties of cotan-Laplace

- *Always, always, always* positive-semidefinite $\mathsf{f}^\mathsf{T}\mathsf{C}\mathsf{f} \geq 0$ (even if cotan weights are negative!)
- Why? $\mathsf{f}^\mathsf{T}\mathsf{C}\mathsf{f}$ is *identical* to summing $||\nabla f||^2$!
- No boundary $\Rightarrow$ constant vector in the kernel / cokernel

Properties of cotan-Laplace

- *Always, always, always* positive-semidefinite $\mathsf{f}^\mathsf{T}\mathsf{C}\mathsf{f} \geq 0$ (even if cotan weights are negative!)
- Why? $\mathsf{f}^\mathsf{T}\mathsf{C}\mathsf{f}$ is *identical* to summing $||\nabla f||^2$!
- No boundary $\Rightarrow$ constant vector in the kernel / cokernel
- Why does it matter? E.g., for Poisson equation:

Properties of cotan-Laplace

- *Always, always, always* positive-semidefinite $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f} \geq 0$ (even if cotan weights are negative!)
- Why? $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f}$ is *identical* to summing $||\nabla f||^2$!
- No boundary $\Rightarrow$ constant vector in the kernel / cokernel
- Why does it matter? E.g., for Poisson equation:
    - solution is unique only up to constant shift

Properties of cotan-Laplace

- *Always, always, always* positive-semidefinite $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f} \geq 0$ (even if cotan weights are negative!)
- Why? $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f}$ is *identical* to summing $||\nabla f||^2$!
- No boundary $\Rightarrow$ constant vector in the kernel / cokernel
- Why does it matter? E.g., for Poisson equation:
  - solution is unique only up to constant shift
  - if RHS has nonzero mean, cannot be solved!

Properties of cotan-Laplace

- *Always, always, always* positive-semidefinite $\mathsf{f}^\mathsf{T}\mathsf{C}\mathsf{f} \geq 0$ (even if cotan weights are negative!)
- Why? $\mathsf{f}^\mathsf{T}\mathsf{C}\mathsf{f}$ is *identical* to summing $||\nabla f||^2$!
- No boundary $\Rightarrow$ constant vector in the kernel / cokernel
- Why does it matter? E.g., for Poisson equation:
  - solution is unique only up to constant shift
  - if RHS has nonzero mean, cannot be solved!
- Exhibits *maximum principle* on Delaunay mesh

Properties of cotan-Laplace

- *Always, always, always* positive-semidefinite $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f} \geq 0$ (even if cotan weights are negative!)
- Why? $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f}$ is *identical* to summing $||\nabla f||^2$!
- No boundary $\Rightarrow$ constant vector in the kernel / cokernel
- Why does it matter? E.g., for Poisson equation:
  - solution is unique only up to constant shift
  - if RHS has nonzero mean, cannot be solved!
- Exhibits *maximum principle* on Delaunay mesh
  - Delaunay: triangle circumcircles are empty

Properties of cotan-Laplace

- *Always, always, always* positive-semidefinite $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f} \geq 0$ (even if cotan weights are negative!)
- Why? $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f}$ is *identical* to summing $||\nabla f||^2$!
- No boundary $\Rightarrow$ constant vector in the kernel / cokernel
- Why does it matter? E.g., for Poisson equation:
  - solution is unique only up to constant shift
  - if RHS has nonzero mean, cannot be solved!
- Exhibits *maximum principle* on Delaunay mesh
  - Delaunay: triangle circumcircles are empty
  - Maximum principle: solution to Laplace equation has no interior extrema (local max or min)

Properties of cotan-Laplace

- *Always, always, always* positive-semidefinite $\mathsf{f}^\mathsf{T}\mathsf{C}\mathsf{f} \geq 0$ (even if cotan weights are negative!)
- Why? $\mathsf{f}^\mathsf{T}\mathsf{C}\mathsf{f}$ is *identical* to summing $||\nabla f||^2$!
- No boundary $\Rightarrow$ constant vector in the kernel / cokernel
- Why does it matter? E.g., for Poisson equation:
    - solution is unique only up to constant shift
    - if RHS has nonzero mean, cannot be solved!
- Exhibits *maximum principle* on Delaunay mesh
    - Delaunay: triangle circumcircles are empty
    - Maximum principle: solution to Laplace equation has no interior extrema (local max or min)
    - **NOTE**: non-Delaunay meshes can also exhibit max principle! (And often do.) Delaunay *sufficient* but not *necessary*. Currently no nice, simple *necessary* condition on mesh geometry.

Properties of cotan-Laplace

- *Always, always, always* positive-semidefinite $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f} \geq 0$ (even if cotan weights are negative!)
- Why? $\mathsf{f}^{\mathsf{T}}\mathsf{C}\mathsf{f}$ is *identical* to summing $||\nabla f||^2$!
- No boundary $\Rightarrow$ constant vector in the kernel / cokernel
- Why does it matter? E.g., for Poisson equation:
  - solution is unique only up to constant shift
  - if RHS has nonzero mean, cannot be solved!
- Exhibits *maximum principle* on Delaunay mesh
  - Delaunay: triangle circumcircles are empty
  - Maximum principle: solution to Laplace equation has no interior extrema (local max or min)
  - **NOTE**: non-Delaunay meshes can also exhibit max principle! (And often do.) Delaunay *sufficient* but not *necessary*. Currently no nice, simple *necessary* condition on mesh geometry.
- For more, see [Wardetzky et al., 2007]

Numerical Issues—Symmetry

- "Best" case for sparse linear systems:

  symmetric positive-(semi)definite $(A^T = A,\ x^T A x \geq 0\ \forall x)$

Numerical Issues—Symmetry

- "Best" case for sparse linear systems:

  symmetric positive-(semi)definite ($\mathsf{A}^\mathsf{T} = \mathsf{A}$, $\mathsf{x}^\mathsf{T}\mathsf{A}\mathsf{x} \geq 0 \ \forall \mathsf{x}$)
- Many good solvers (Cholesky, conjugate gradient, …)

Numerical Issues—Symmetry

- "Best" case for sparse linear systems:

  symmetric positive-(semi)definite ($\mathsf{A}^\mathsf{T} = \mathsf{A}$, $\mathsf{x}^\mathsf{T}\mathsf{A}\mathsf{x} \geq 0\ \forall \mathsf{x}$)
- Many good solvers (Cholesky, conjugate gradient, ...)
- Discrete Poisson equation looks like $\mathsf{M}^{-1}\mathsf{C}\mathsf{u} = \mathsf{f}$

Numerical Issues—Symmetry

- "Best" case for sparse linear systems:

  symmetric positive-(semi)definite ($\mathsf{A}^\mathsf{T} = \mathsf{A}, \mathsf{x}^\mathsf{T}\mathsf{A}\mathsf{x} \geq 0 \; \forall \mathsf{x}$)
- Many good solvers (Cholesky, conjugate gradient, ...)
- Discrete Poisson equation looks like $\mathsf{M}^{-1}\mathsf{C}\mathsf{u} = \mathsf{f}$
- C is symmetric, but $\mathsf{M}^{-1}\mathsf{C}$ is not!
- Can easily be made symmetric:

$$\mathsf{C}\mathsf{u} = \mathsf{M}\mathsf{f}$$

Numerical Issues—Symmetry

- "Best" case for sparse linear systems:

  symmetric positive-(semi)definite ($\mathsf{A}^\mathsf{T} = \mathsf{A}$, $\mathsf{x}^\mathsf{T}\mathsf{A}\mathsf{x} \geq 0\ \forall \mathsf{x}$)
- Many good solvers (Cholesky, conjugate gradient, ...)
- Discrete Poisson equation looks like $\mathsf{M}^{-1}\mathsf{C}\mathsf{u} = \mathsf{f}$
- C is symmetric, but $\mathsf{M}^{-1}\mathsf{C}$ is not!
- Can easily be made symmetric:

$$\mathsf{C}\mathsf{u} = \mathsf{M}\mathsf{f}$$

- In other words: just multiply by vertex areas!

Numerical Issues—Symmetry

- "Best" case for sparse linear systems:

  symmetric positive-(semi)definite ($A^T = A$, $x^T A x \geq 0 \; \forall x$)
- Many good solvers (Cholesky, conjugate gradient, ...)
- Discrete Poisson equation looks like $M^{-1} C u = f$
- C is symmetric, but $M^{-1} C$ is not!
- Can easily be made symmetric:

$$Cu = Mf$$

- In other words: just multiply by vertex areas!
- Seemingly superficial change...

Numerical Issues—Symmetry

- "Best" case for sparse linear systems:

  symmetric positive-(semi)definite ($A^T = A$, $x^T A x \geq 0\ \forall x$)
- Many good solvers (Cholesky, conjugate gradient, ...)
- Discrete Poisson equation looks like $M^{-1}Cu = f$
- C is symmetric, but $M^{-1}C$ is not!
- Can easily be made symmetric:

$$Cu = Mf$$

- In other words: just multiply by vertex areas!
- Seemingly superficial change...
- ...but makes computation simpler / more efficient

Numerical Issues—Symmetry

- "Best" case for sparse linear systems:

  symmetric positive-(semi)definite ($A^T = A$, $x^T A x \geq 0\ \forall x$)
- Many good solvers (Cholesky, conjugate gradient, ...)
- Discrete Poisson equation looks like $M^{-1}Cu = f$
- C is symmetric, but $M^{-1}C$ is not!
- Can easily be made symmetric:

$$Cu = Mf$$

- In other words: just multiply by vertex areas!
- Seemingly superficial change...
- ...but makes computation simpler / more efficient

Numerical Issues—Symmetry, continued

- Can also make heat equation symmetric

Numerical Issues—Symmetry, continued

- Can also make heat equation symmetric
- Instead of $(\mathsf{I} - h\mathsf{L})\mathsf{u}_{k+1} = \mathsf{u}_k$, use

$$(\mathsf{M} - h\mathsf{C})\mathsf{u}_{k+1} = \mathsf{M}\mathsf{u}_k$$

Numerical Issues—Symmetry, continued

- Can also make heat equation symmetric
- Instead of $(\mathsf{I} - h\mathsf{L})\mathsf{u}_{k+1} = \mathsf{u}_k$, use

$$(\mathsf{M} - h\mathsf{C})\mathsf{u}_{k+1} = \mathsf{M}\mathsf{u}_k$$

- What about smallest eigenvalue problem $\mathsf{L}\mathsf{u} = \lambda \mathsf{u}$?

Numerical Issues—Symmetry, continued

- Can also make heat equation symmetric
- Instead of $(\mathsf{I} - h\mathsf{L})\mathsf{u}_{k+1} = \mathsf{u}_k$, use

$$(\mathsf{M} - h\mathsf{C})\mathsf{u}_{k+1} = \mathsf{M}\mathsf{u}_k$$

- What about smallest eigenvalue problem $\mathsf{L}\mathsf{u} = \lambda\mathsf{u}$?
- Two options:

Numerical Issues—Symmetry, continued

- Can also make heat equation symmetric
- Instead of $(\mathsf{I} - h\mathsf{L})\mathsf{u}_{k+1} = \mathsf{u}_k$, use

$$(\mathsf{M} - h\mathsf{C})\mathsf{u}_{k+1} = \mathsf{M}\mathsf{u}_k$$

- What about smallest eigenvalue problem $\mathsf{L}\mathsf{u} = \lambda\mathsf{u}$?
- Two options:
  1. Solve *generalized* eigenvalue problem $\mathsf{C}\mathsf{u} = \lambda\mathsf{M}\mathsf{u}$

Numerical Issues—Symmetry, continued

- Can also make heat equation symmetric
- Instead of $(I - hL)u_{k+1} = u_k$, use

$$(M - hC)u_{k+1} = Mu_k$$

- What about smallest eigenvalue problem $Lu = \lambda u$?
- Two options:
  1. Solve *generalized* eigenvalue problem $Cu = \lambda Mu$
  2. Solve $M^{-1/2}CM^{-1/2}\tilde{u} = \lambda\tilde{u}$, recover $u = M^{-1/2}\tilde{u}$

Numerical Issues—Symmetry, continued

- Can also make heat equation symmetric
- Instead of $(\mathsf{I} - \mathsf{hL})\mathsf{u}_{\mathsf{k}+1} = \mathsf{u}_\mathsf{k}$, use

$$(\mathsf{M} - \mathsf{hC})\mathsf{u}_{\mathsf{k}+1} = \mathsf{M}\mathsf{u}_\mathsf{k}$$

- What about smallest eigenvalue problem $\mathsf{Lu} = \lambda\mathsf{u}$?
- Two options:
  1. Solve *generalized* eigenvalue problem $\mathsf{Cu} = \lambda\mathsf{Mu}$
  2. Solve $\mathsf{M}^{-1/2}\mathsf{C}\mathsf{M}^{-1/2}\tilde{\mathsf{u}} = \lambda\tilde{\mathsf{u}}$, recover $\mathsf{u} = \mathsf{M}^{-1/2}\tilde{\mathsf{u}}$
- Note: $M^{-1/2}$ just means *"put $1/\sqrt{\mathcal{A}_\mathsf{i}}$ on the diagonal!"*

Numerical Issues—Symmetry, continued

- Can also make heat equation symmetric
- Instead of $(\mathsf{I} - \mathsf{hL})\mathsf{u}_{\mathsf{k}+1} = \mathsf{u}_\mathsf{k}$, use

$$(\mathsf{M} - \mathsf{hC})\mathsf{u}_{\mathsf{k}+1} = \mathsf{M}\mathsf{u}_\mathsf{k}$$

- What about smallest eigenvalue problem $\mathsf{Lu} = \lambda \mathsf{u}$?
- Two options:
  1. Solve *generalized* eigenvalue problem $\mathsf{Cu} = \lambda \mathsf{Mu}$
  2. Solve $\mathsf{M}^{-1/2}\mathsf{C}\mathsf{M}^{-1/2}\tilde{\mathsf{u}} = \lambda \tilde{\mathsf{u}}$, recover $\mathsf{u} = \mathsf{M}^{-1/2}\tilde{\mathsf{u}}$
- Note: $M^{-1/2}$ just means *"put $1/\sqrt{\mathcal{A}_\mathsf{i}}$ on the diagonal!"*

Numerical Issues—Direct vs. Iterative Solvers

- Direct (e.g., LL^T, LU, QR, $\ldots$)

Numerical Issues—Direct vs. Iterative Solvers

- Direct (e.g., LL^T, LU, QR, $\ldots$)
  - **pros:** great for multiple right-hand sides; (can be) less sensitive to numerical instability; solve many types of problems, under/overdetermined systems.

Numerical Issues—Direct vs. Iterative Solvers

- Direct (e.g., LL^T, LU, QR, $\ldots$)
  - **pros:** great for multiple right-hand sides; (can be) less sensitive to numerical instability; solve many types of problems, under/overdetermined systems.
  - **cons:** prohibitively expensive for large problems; factors are quite dense for 3D (volumetric) problems

Numerical Issues—Direct vs. Iterative Solvers

- Direct (e.g., LL^T, LU, QR, $\ldots$)
  - **pros:** great for multiple right-hand sides; (can be) less sensitive to numerical instability; solve many types of problems, under/overdetermined systems.
  - **cons:** prohibitively expensive for large problems; factors are quite dense for 3D (volumetric) problems
- Iterative (e.g., conjugate gradient, multigrid, $\ldots$)

Numerical Issues—Direct vs. Iterative Solvers

- Direct (e.g., LL^T, LU, QR, $\ldots$)
  - **pros:** great for multiple right-hand sides; (can be) less sensitive to numerical instability; solve many types of problems, under/overdetermined systems.
  - **cons:** prohibitively expensive for large problems; factors are quite dense for 3D (volumetric) problems
- Iterative (e.g., conjugate gradient, multigrid, $\ldots$)
  - **pros:** can handle very large problems; can be implemented via *callback* (instead of matrix); asymptotic running times approaching linear time (in theory$\ldots$)

Numerical Issues—Direct vs. Iterative Solvers

- Direct (e.g., LL^T, LU, QR, $\ldots$)
  - **pros:** great for multiple right-hand sides; (can be) less sensitive to numerical instability; solve many types of problems, under/overdetermined systems.
  - **cons:** prohibitively expensive for large problems; factors are quite dense for 3D (volumetric) problems
- Iterative (e.g., conjugate gradient, multigrid, . . .)
  - **pros:** can handle very large problems; can be implemented via *callback* (instead of matrix); asymptotic running times approaching linear time (in theory. . .)
  - **cons:** poor performance without good preconditioners; less benefit for multiple right-hand sides; best-in-class methods may handle only symmetric positive-(semi)definite systems

Numerical Issues—Direct vs. Iterative Solvers

- Direct (e.g., LL^T, LU, QR, ...)
  - **pros:** great for multiple right-hand sides; (can be) less sensitive to numerical instability; solve many types of problems, under/overdetermined systems.
  - **cons:** prohibitively expensive for large problems; factors are quite dense for 3D (volumetric) problems
- Iterative (e.g., conjugate gradient, multigrid, ...)
  - **pros:** can handle very large problems; can be implemented via *callback* (instead of matrix); asymptotic running times approaching linear time (in theory...)
  - **cons:** poor performance without good preconditioners; less benefit for multiple right-hand sides; best-in-class methods may handle only symmetric positive-(semi)definite systems
- No perfect solution! Each problem is different.

Solving Equations in Linear Time

- Is solving Poisson, Laplace, etc., *truly* linear time in 2D?

Solving Equations in Linear Time

- Is solving Poisson, Laplace, etc., *truly* linear time in 2D?
- Jury is still out, but keep inching forward:
  - [Vaidya, 1991]—use spanning tree as preconditioner
  - [Alon et al., 1995]—use low-stretch spanning trees
  - [Spielman and Teng, 2004]—first "nearly linear time" solver
  - [Krishnan et al., 2013]—practical solver for graphics
  - Lots of recent activity in both preconditioners and direct solvers (e.g., [Koutis et al., 2011], [Gillman and Martinsson, 2013])

Solving Equations in Linear Time

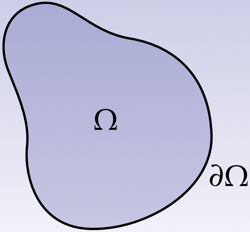
- Is solving Poisson, Laplace, etc., *truly* linear time in 2D?
- Jury is still out, but keep inching forward:
  - [Vaidya, 1991]—use spanning tree as preconditioner
  - [Alon et al., 1995]—use low-stretch spanning trees
  - [Spielman and Teng, 2004]—first "nearly linear time" solver
  - [Krishnan et al., 2013]—practical solver for graphics
  - Lots of recent activity in both preconditioners and direct solvers (e.g., [Koutis et al., 2011], [Gillman and Martinsson, 2013])
- *Best theoretical results may lack practical implementations!*

Solving Equations in Linear Time

- Is solving Poisson, Laplace, etc., *truly* linear time in 2D?
- Jury is still out, but keep inching forward:
  - [Vaidya, 1991]—use spanning tree as preconditioner
  - [Alon et al., 1995]—use low-stretch spanning trees
  - [Spielman and Teng, 2004]—first "nearly linear time" solver
  - [Krishnan et al., 2013]—practical solver for graphics
  - Lots of recent activity in both preconditioners and direct solvers (e.g., [Koutis et al., 2011], [Gillman and Martinsson, 2013])
- *Best theoretical results may lack practical implementations!*
- Older codes benefit from extensive low-level optimization

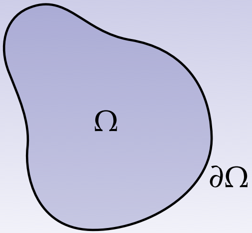
Solving Equations in Linear Time

- Is solving Poisson, Laplace, etc., *truly* linear time in 2D?
- Jury is still out, but keep inching forward:
  - [Vaidya, 1991]—use spanning tree as preconditioner
  - [Alon et al., 1995]—use low-stretch spanning trees
  - [Spielman and Teng, 2004]—first "nearly linear time" solver
  - [Krishnan et al., 2013]—practical solver for graphics
  - Lots of recent activity in both preconditioners and direct solvers (e.g., [Koutis et al., 2011], [Gillman and Martinsson, 2013])
- *Best theoretical results may lack practical implementations!*
- Older codes benefit from extensive low-level optimization
- Long term: probably indistinguishable from $O(n)$



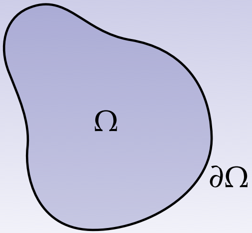
- PDE (Laplace, Poisson, heat equation, etc.) determines behavior "inside" domain Ω

Boundary Conditions



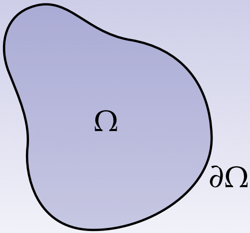
- PDE (Laplace, Poisson, heat equation, etc.) determines behavior "inside" domain Ω
- Also need to say how solution behaves on boundary $\partial\Omega$

Boundary Conditions



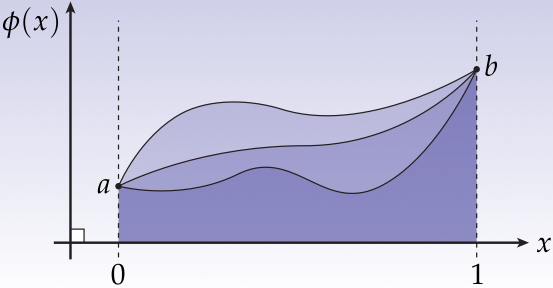
- PDE (Laplace, Poisson, heat equation, etc.) determines behavior "inside" domain Ω
- Also need to say how solution behaves on boundary $\partial\Omega$
- Often trickiest part (both mathematically & numerically)

Boundary Conditions



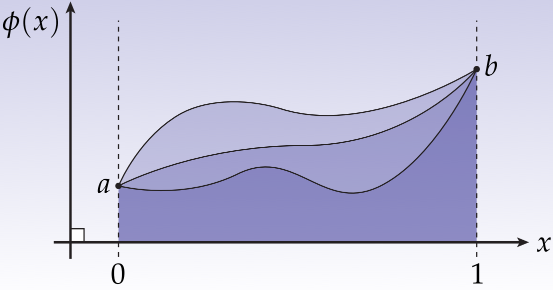
- PDE (Laplace, Poisson, heat equation, etc.) determines behavior "inside" domain Ω
- Also need to say how solution behaves on boundary $\partial\Omega$
- Often trickiest part (both mathematically & numerically)
- Very easy to get wrong!

Dirichlet Boundary Conditions



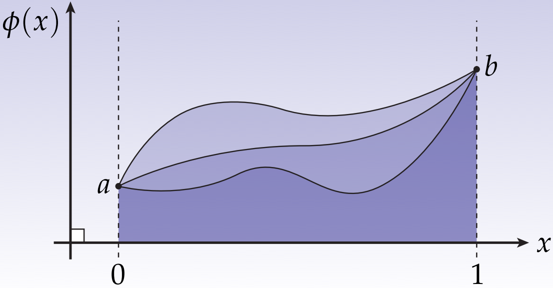
- "Dirichlet" $\iff$ prescribe *values*

Dirichlet Boundary Conditions



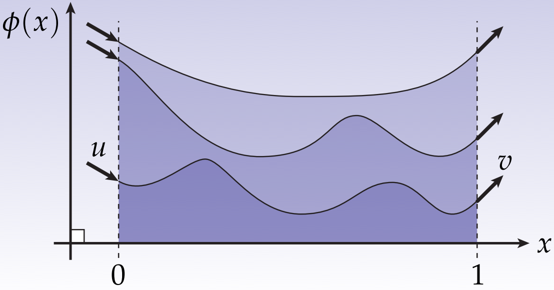
- "Dirichlet" $\iff$ prescribe *values*
- E.g., $\phi(0) = a$, $\phi(1) = b$

Dirichlet Boundary Conditions



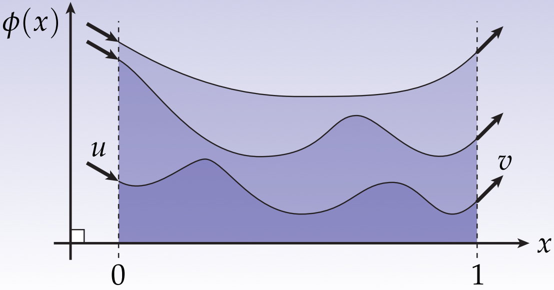
- "Dirichlet" $\iff$ prescribe *values*
- E.g., $\phi(0) = a$, $\phi(1) = b$
- (Many possible functions "in between!")

Neumann Boundary Conditions



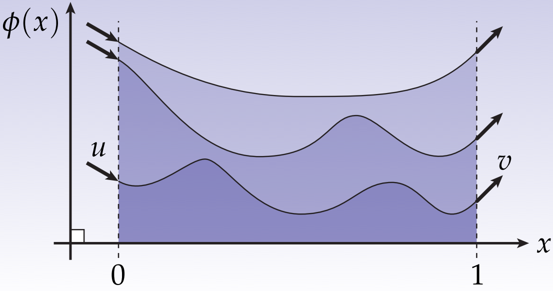
- "Neumann" $\Longleftrightarrow$ prescribe *derivatives*

Neumann Boundary Conditions



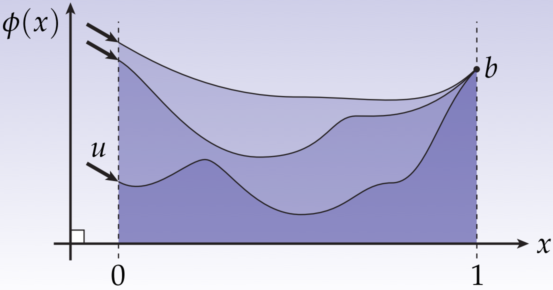
- "Neumann" $\iff$ prescribe *derivatives*
- E.g., $\phi'(0) = u$, $\phi'(1) = v$

Neumann Boundary Conditions



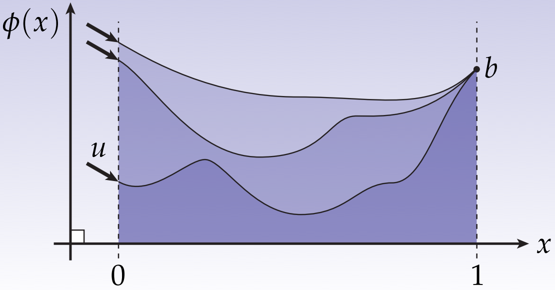
- "Neumann" $\iff$ prescribe *derivatives*
- E.g., $\phi'(0) = u$, $\phi'(1) = v$
- (Again, many possible solutions.)

Both Neumann & Dirichlet



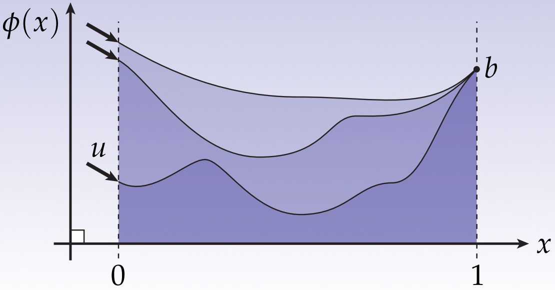
- Or: prescribe some values, some derivatives

Both Neumann & Dirichlet



- Or: prescribe some values, some derivatives
- E.g., $\phi'(0) = u$, $\phi(1) = b$

Both Neumann & Dirichlet



- Or: prescribe some values, some derivatives
- E.g., $\phi'(0) = u$, $\phi(1) = b$
- (What about $\phi'(1) = v$, $\phi(1) = b$?)

Laplace w/ Dirichlet Boundary Conditions (1D)

- 1D Laplace: $\partial^2\phi/\partial x^2 = 0$

Laplace w/ Dirichlet Boundary Conditions (1D)

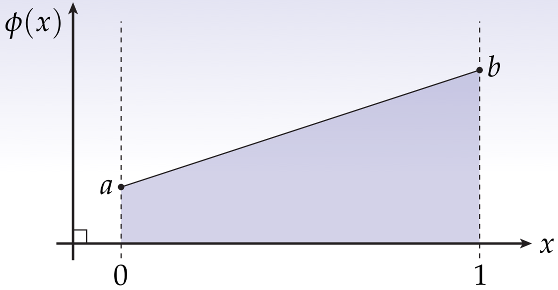
- 1D Laplace: $\partial^2\phi/\partial x^2 = 0$
- Solutions: $\phi(x) = cx + d$ (linear functions)

Laplace w/ Dirichlet Boundary Conditions (1D)

- 1D Laplace: $\partial^2\phi/\partial x^2 = 0$
- Solutions: $\phi(x) = cx + d$ (linear functions)
- Can we always satisfy Dirichlet boundary conditions?

Laplace w/ Dirichlet Boundary Conditions (1D)

- 1D Laplace: $\partial^2\phi/\partial x^2 = 0$
- Solutions: $\phi(x) = cx + d$ (linear functions)
- Can we always satisfy Dirichlet boundary conditions?



- Yes: a line can interpolate any two points

Laplace w/ Neumann Boundary Conditions (1D)

- What about Neumann boundary conditions?

Laplace w/ Neumann Boundary Conditions (1D)

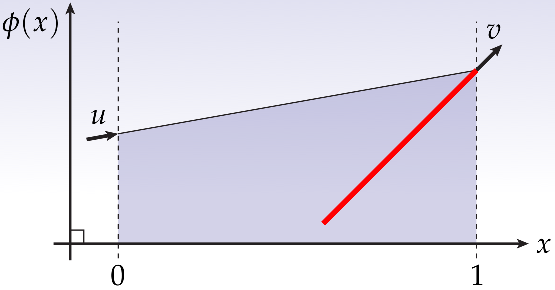
- What about Neumann boundary conditions?
- Solution must still be a line: $\phi(x) = cx + d$

Laplace w/ Neumann Boundary Conditions (1D)

- What about Neumann boundary conditions?
- Solution must still be a line: $\phi(x) = cx + d$
- Can we prescribe the derivative at both ends?

Laplace w/ Neumann Boundary Conditions (1D)

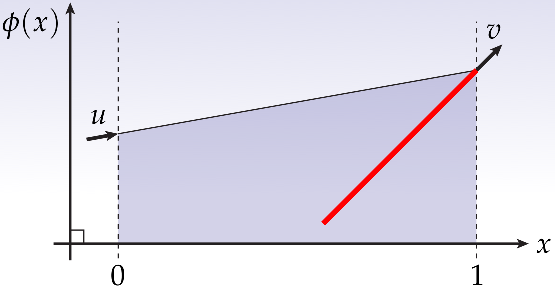
- What about Neumann boundary conditions?
- Solution must still be a line: $\phi(x) = cx + d$
- Can we prescribe the derivative at both ends?



- *No!* A line can have only one slope!

Laplace w/ Neumann Boundary Conditions (1D)

- What about Neumann boundary conditions?
- Solution must still be a line: $\phi(x) = cx + d$
- Can we prescribe the derivative at both ends?



- *No!* A line can have only one slope!
- In general: solutions to PDE may *not* exist for given BCs

Laplace w/ Dirichlet Boundary Conditions (2D)

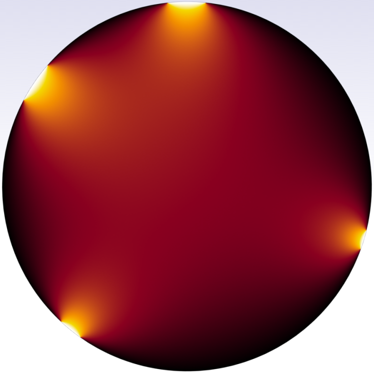
- 2D Laplace: $\Delta\phi = 0$

Laplace w/ Dirichlet Boundary Conditions (2D)

- 2D Laplace: $\Delta\phi = 0$
- Can we always satisfy Dirichlet boundary conditions?

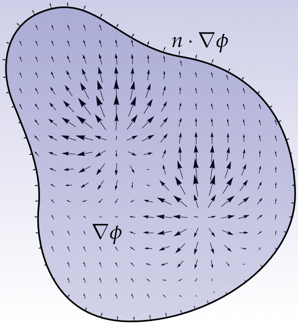
Laplace w/ Dirichlet Boundary Conditions (2D)

- 2D Laplace: $\Delta\phi = 0$
- Can we always satisfy Dirichlet boundary conditions?
- Yes: Laplace is steady-state solution to heat flow $\frac{d}{dt}\phi = \Delta\phi$



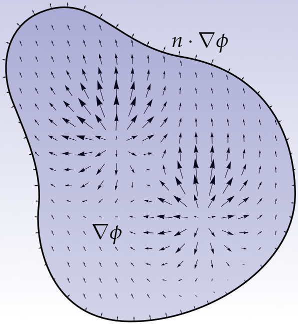
- Dirichlet data is just "heat" along boundary

Laplace w/ Neumann Boundary Conditions (2D)



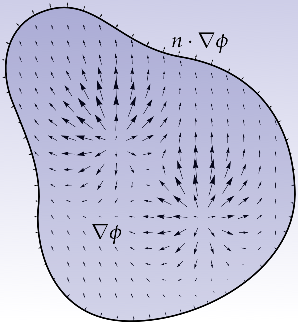
- What about Neumann boundary conditions?

Laplace w/ Neumann Boundary Conditions (2D)



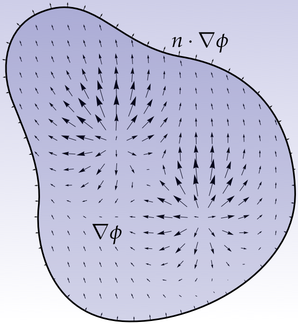
- What about Neumann boundary conditions?
- Still want to solve $\Delta\phi = 0$
- Want to prescribe *normal derivative* $n \cdot \nabla\phi$

Laplace w/ Neumann Boundary Conditions (2D)



- What about Neumann boundary conditions?
- Still want to solve $\Delta\phi = 0$
- Want to prescribe *normal derivative* $n \cdot \nabla\phi$
- Wasn't always possible in 1D…

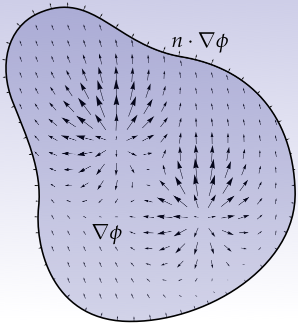
Laplace w/ Neumann Boundary Conditions (2D)



- What about Neumann boundary conditions?
- Still want to solve $\Delta\phi = 0$
- Want to prescribe *normal derivative* $n \cdot \nabla\phi$
- Wasn't always possible in 1D…
- In 2D, we have divergence theorem:

$$\int_\Omega 0 \stackrel{!}{=} \int_\Omega \Delta\phi = \int_\Omega \nabla \cdot \nabla\phi = \int_{\partial\Omega} n \cdot \nabla\phi$$

Laplace w/ Neumann Boundary Conditions (2D)



- What about Neumann boundary conditions?
- Still want to solve $\Delta\phi = 0$
- Want to prescribe *normal derivative* $n \cdot \nabla\phi$
- Wasn't always possible in 1D…
- In 2D, we have divergence theorem:

$$\int_\Omega 0 \overset{!}{=} \int_\Omega \Delta\phi = \int_\Omega \nabla \cdot \nabla\phi = \int_{\partial\Omega} n \cdot \nabla\phi$$

- Conclusion: can only solve $\Delta\phi = 0$ if Neumann BCs have *zero mean!*

Discrete Boundary Conditions - Dirichlet

- Suppose we want to solve $\Delta u = f$ s.t. $u|_{\partial\Omega} = g$ (Poisson equation w/ Dirichlet boundary conditions)

Discrete Boundary Conditions - Dirichlet

- Suppose we want to solve $\Delta u = f$ s.t. $u|_{\partial\Omega} = g$ (Poisson equation w/ Dirichlet boundary conditions)
- Discretized Poisson equation as $\mathsf{Cu} = \mathsf{Mf}$

Discrete Boundary Conditions - Dirichlet

- Suppose we want to solve $\Delta u = f$ s.t. $u|_{\partial\Omega} = g$ (Poisson equation w/ Dirichlet boundary conditions)
- Discretized Poisson equation as $\mathsf{C}\mathsf{u} = \mathsf{M}\mathsf{f}$
- Let I, B denote interior, boundary vertices, respectively. Get

$$\begin{bmatrix} \mathsf{C}_{II} & \mathsf{C}_{IB} \\ \mathsf{C}_{BI} & \mathsf{C}_{BB} \end{bmatrix} \begin{bmatrix} \mathsf{u}_I \\ \mathsf{u}_B \end{bmatrix} = \begin{bmatrix} \mathsf{M}_{II} & 0 \\ 0 & \mathsf{M}_{BB} \end{bmatrix} \begin{bmatrix} \mathsf{f}_I \\ \mathsf{f}_B \end{bmatrix}$$

Discrete Boundary Conditions - Dirichlet

- Suppose we want to solve $\Delta u = f$ s.t. $u|_{\partial\Omega} = g$ (Poisson equation w/ Dirichlet boundary conditions)
- Discretized Poisson equation as $\mathsf{C}\mathsf{u} = \mathsf{M}\mathsf{f}$
- Let I, B denote interior, boundary vertices, respectively. Get

$$\begin{bmatrix} \mathsf{C}_{II} & \mathsf{C}_{IB} \\ \mathsf{C}_{BI} & \mathsf{C}_{BB} \end{bmatrix} \begin{bmatrix} \mathsf{u}_I \\ \mathsf{u}_B \end{bmatrix} = \begin{bmatrix} \mathsf{M}_{II} & 0 \\ 0 & \mathsf{M}_{BB} \end{bmatrix} \begin{bmatrix} \mathsf{f}_I \\ \mathsf{f}_B \end{bmatrix}$$

- Since u_B is known (boundary values), solve just $\mathsf{C}_{II}\mathsf{u}_I = \mathsf{M}_{II}\mathsf{f}_I - \mathsf{C}_{IB}\mathsf{u}_B$ for u_I (right-hand side is known).

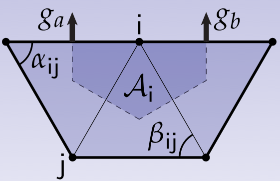
Discrete Boundary Conditions - Dirichlet

- Suppose we want to solve $\Delta u = f$ s.t. $u|_{\partial\Omega} = g$ (Poisson equation w/ Dirichlet boundary conditions)
- Discretized Poisson equation as $\mathsf{C}\mathsf{u} = \mathsf{M}\mathsf{f}$
- Let I, B denote interior, boundary vertices, respectively. Get

$$\begin{bmatrix} \mathsf{C}_{II} & \mathsf{C}_{IB} \\ \mathsf{C}_{BI} & \mathsf{C}_{BB} \end{bmatrix} \begin{bmatrix} \mathsf{u}_I \\ \mathsf{u}_B \end{bmatrix} = \begin{bmatrix} \mathsf{M}_{II} & 0 \\ 0 & \mathsf{M}_{BB} \end{bmatrix} \begin{bmatrix} \mathsf{f}_I \\ \mathsf{f}_B \end{bmatrix}$$

- Since u_B is known (boundary values), solve just $\mathsf{C}_{II}\mathsf{u}_I = \mathsf{M}_{II}\mathsf{f}_I - \mathsf{C}_{IB}\mathsf{u}_B$ for u_I (right-hand side is known).
- Can skip matrix multiply and compute entries of RHS directly: $\mathcal{A}_\mathsf{i} f_\mathsf{i} - \sum_{\mathsf{j} \in \mathcal{N}_\partial(\mathsf{i})} (\cot \alpha_{ij} + \cot \beta_{ij}) u_j$
- Here $\mathcal{N}_\partial(\mathsf{i})$ denotes neighbors of i *on the boundary*

Discrete Boundary Conditions - Neumann



- Integrate both sides of $\Delta u = f$ over cell C_i ("finite volume")

$$\int_{C_i} f \stackrel{!}{=} \int_{C_i} \Delta u = \int_{C_i} \nabla \cdot \nabla u = \int_{\partial C_i} n \cdot \nabla u$$

- Gives usual cotangent formula for interior vertices; for boundary vertex i, yields

$$\mathcal{A}_{ii} \stackrel{!}{=} \frac{1}{2}(g_a + g_b) + \frac{1}{2} \sum_{j \in \mathcal{N}_{\text{int}}} (\cot \alpha_{ij} + \cot \beta_{ij})(u_j - u_i)$$

- Here g_a, g_b are prescribed normal derivatives; just subtract from RHS and solve $Cu = Mf$ as usual

Discrete Boundary Conditions - Neumann

- Other possible boundary conditions (e.g., Robin)

Discrete Boundary Conditions - Neumann

- Other possible boundary conditions (e.g., Robin)
- Dirichlet, Neumann most common—implementation of other BCs will be similar

Discrete Boundary Conditions - Neumann

- Other possible boundary conditions (e.g., Robin)
- Dirichlet, Neumann most common—implementation of other BCs will be similar
- When in doubt, return to smooth equations and *integrate!*

Discrete Boundary Conditions - Neumann

- Other possible boundary conditions (e.g., Robin)
- Dirichlet, Neumann most common—implementation of other BCs will be similar
- When in doubt, return to smooth equations and *integrate!*
- …and *make sure your equation has a solution!*

Discrete Boundary Conditions - Neumann

- Other possible boundary conditions (e.g., Robin)
- Dirichlet, Neumann most common—implementation of other BCs will be similar
- When in doubt, return to smooth equations and *integrate!*
- …and *make sure your equation has a solution!*
- Solver will *NOT* always tell you if there's a problem!

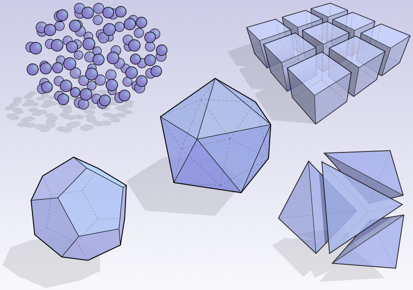
Discrete Boundary Conditions - Neumann

- Other possible boundary conditions (e.g., Robin)
- Dirichlet, Neumann most common—implementation of other BCs will be similar
- When in doubt, return to smooth equations and *integrate!*
- …and *make sure your equation has a solution!*
- Solver will *NOT* always tell you if there's a problem!
- Easy test? Compute the residual $r := Ax - b$. If the relative residual $||r||_\infty / ||b||_\infty$ is far from zero (e.g., greater than 10^{-14} in double precision), *you did not actually solve your problem!*

Discrete Boundary Conditions - Neumann

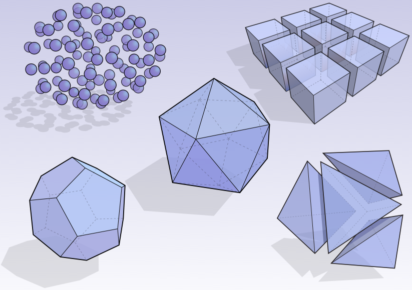
- Other possible boundary conditions (e.g., Robin)
- Dirichlet, Neumann most common—implementation of other BCs will be similar
- When in doubt, return to smooth equations and *integrate!*
- …and *make sure your equation has a solution!*
- Solver will *NOT* always tell you if there's a problem!
- Easy test? Compute the residual $r := Ax - b$. If the relative residual $||r||_\infty / ||b||_\infty$ is far from zero (e.g., greater than 10^{-14} in double precision), *you did not actually solve your problem!*

Alternative Discretizations



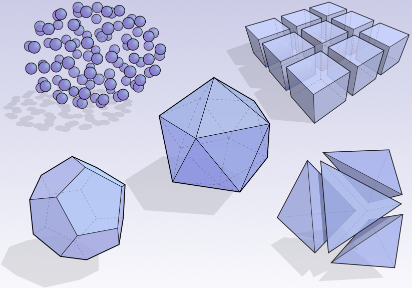
- Have spent a lot of time on triangle meshes…

Alternative Discretizations



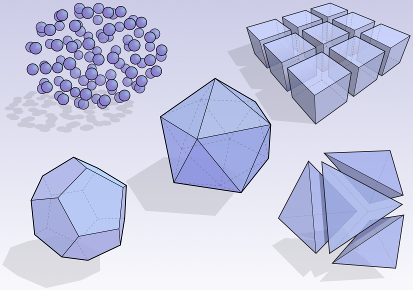
- Have spent a lot of time on triangle meshes…
- …plenty of other ways to describe a surface!

Alternative Discretizations



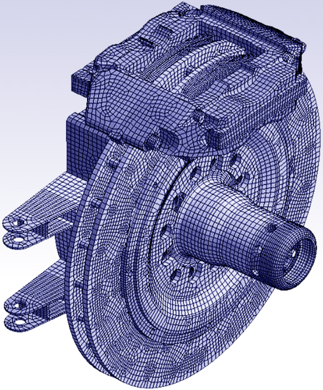
- Have spent a lot of time on triangle meshes…
- …plenty of other ways to describe a surface!
- E.g., *points* are increasingly popular (due to 3D scanning)

Alternative Discretizations



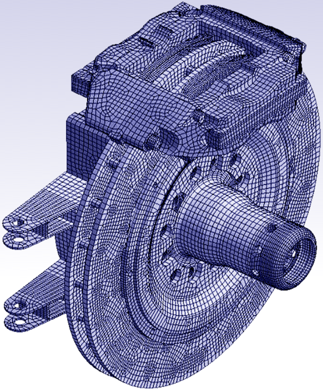
- Have spent a lot of time on triangle meshes…
- …plenty of other ways to describe a surface!
- E.g., *points* are increasingly popular (due to 3D scanning)
- Also: more accurate discretization on triangle meshes

Quad, Polygon Meshes



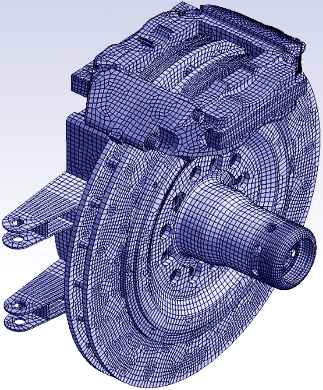
- *Quads* popular alternative to triangles. Why?

Quad, Polygon Meshes



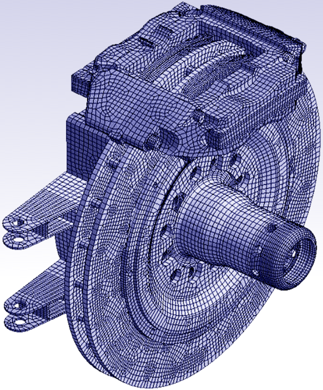
- *Quads* popular alternative to triangles. Why?
  - capture *principal curvatures* of a surface

Quad, Polygon Meshes



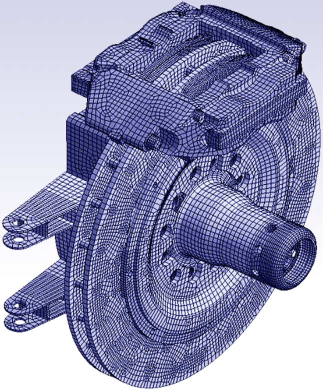
- *Quads* popular alternative to triangles. Why?
  - capture *principal curvatures* of a surface
  - nice bases can be built via *tensor products*

Quad, Polygon Meshes



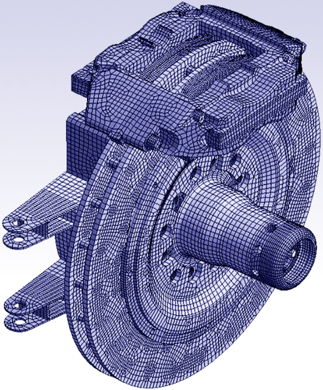
- *Quads* popular alternative to triangles. Why?
  - capture *principal curvatures* of a surface
  - nice bases can be built via *tensor products*
  - see [Bommes et al., 2013] for further discussion

Quad, Polygon Meshes



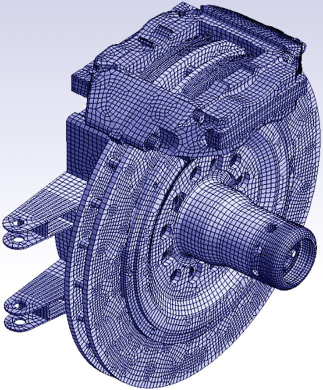
- *Quads* popular alternative to triangles. Why?
  - capture *principal curvatures* of a surface
  - nice bases can be built via *tensor products*
  - see [Bommes et al., 2013] for further discussion
- More generally: meshes with quads *and* triangles *and* . . .

Quad, Polygon Meshes



- *Quads* popular alternative to triangles. Why?
    - capture *principal curvatures* of a surface
    - nice bases can be built via *tensor products*
    - see [Bommes et al., 2013] for further discussion
- More generally: meshes with quads *and* triangles *and* . . .
- Nice discretization: [Alexa and Wardetzky, 2011]

Quad, Polygon Meshes



- *Quads* popular alternative to triangles. Why?
  - capture *principal curvatures* of a surface
  - nice bases can be built via *tensor products*
  - see [Bommes et al., 2013] for further discussion
- More generally: meshes with quads *and* triangles *and* ...
- Nice discretization: [Alexa and Wardetzky, 2011]
- Can then solve all the same problems (Laplace, Poisson, heat, ...)

Point Clouds

- Real data often *point cloud* with no connectivity (plus noise, holes…)

Point Clouds

- Real data often *point cloud* with no connectivity (plus noise, holes. . .)
- Can still build Laplace operator!

Point Clouds

- Real data often *point cloud* with no connectivity (plus noise, holes. . .)
- Can still build Laplace operator!
- Rough idea: use heat flow *to discretize* Δ

Point Clouds

- Real data often *point cloud* with no connectivity (plus noise, holes...)
- Can still build Laplace operator!
- Rough idea: use heat flow *to discretize* Δ
- $\frac{d}{dt}u = \Delta u \implies \Delta u \approx (u(T) - u(0))/T$

Point Clouds

- Real data often *point cloud* with no connectivity (plus noise, holes. . .)
- Can still build Laplace operator!
- Rough idea: use heat flow *to discretize* Δ
- $\frac{d}{dt}u = \Delta u \implies \Delta u \approx (u(T) - u(0))/T$
- How do we get $u(T)$? Convolve u with (Euclidean) heat kernel $\frac{1}{4\pi T}e^{r^2/4T}$

Point Clouds

- Real data often *point cloud* with no connectivity (plus noise, holes...)
- Can still build Laplace operator!
- Rough idea: use heat flow *to discretize* Δ
- $\frac{d}{dt}u = \Delta u \implies \Delta u \approx (u(T) - u(0))/T$
- How do we get $u(T)$? Convolve u with (Euclidean) heat kernel $\frac{1}{4\pi T}e^{r^2/4T}$
- Converges with more samples, T goes to zero (under certain conditions!)

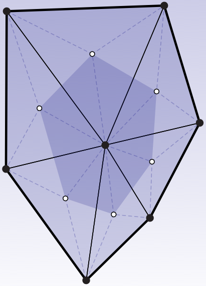
Point Clouds

- Real data often *point cloud* with no connectivity (plus noise, holes...)
- Can still build Laplace operator!
- Rough idea: use heat flow *to discretize* Δ
- $\frac{d}{dt}u = \Delta u \implies \Delta u \approx (u(T) - u(0))/T$
- How do we get $u(T)$? Convolve u with (Euclidean) heat kernel $\frac{1}{4\pi T}e^{r^2/4T}$
- Converges with more samples, T goes to zero (under certain conditions!)
- Details: [Belkin et al., 2009, Liu et al., 2012]

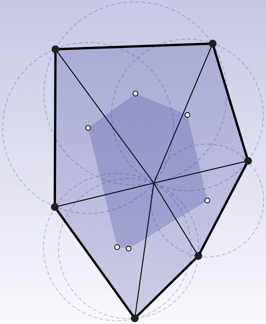
Point Clouds

- Real data often *point cloud* with no connectivity (plus noise, holes...)
- Can still build Laplace operator!
- Rough idea: use heat flow *to discretize* Δ
- $\frac{d}{dt}u = \Delta u \implies \Delta u \approx (u(T) - u(0))/T$
- How do we get $u(T)$? Convolve u with (Euclidean) heat kernel $\frac{1}{4\pi T}e^{r^2/4T}$
- Converges with more samples, T goes to zero (under certain conditions!)
- Details: [Belkin et al., 2009, Liu et al., 2012]
- From there, solve all the same problems! (Again.)

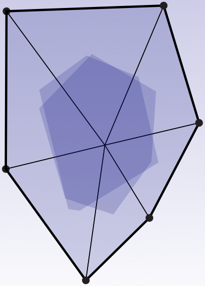
Dual Mesh



barycentric



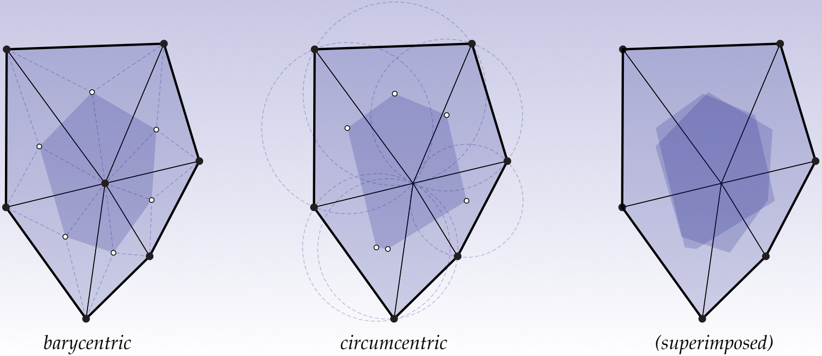
circumcentric



(superimposed)

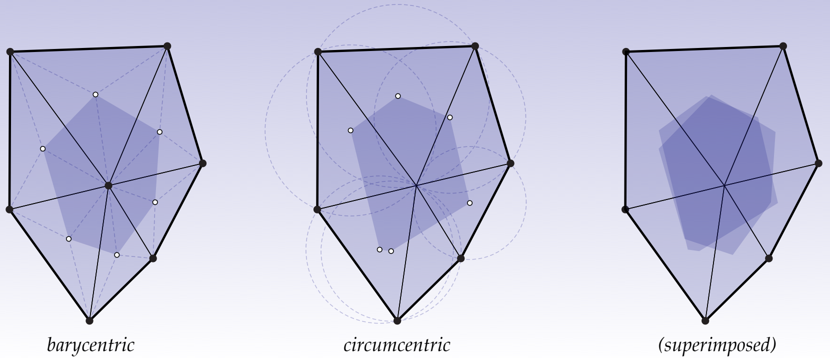
- Earlier saw Laplacian discretized via *dual mesh*

Dual Mesh



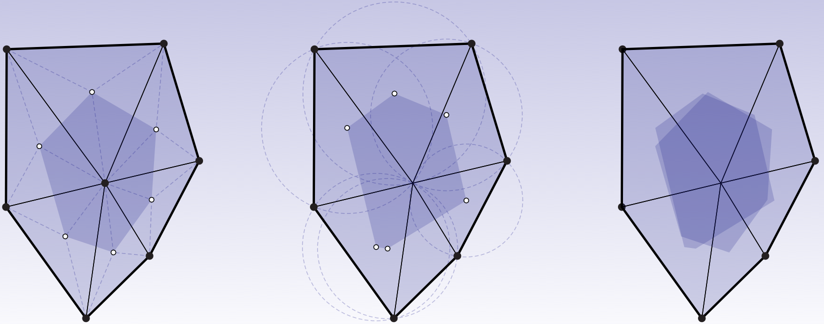
- Earlier saw Laplacian discretized via *dual mesh*
- Different duals lead to operators with different accuracy

Dual Mesh



- Earlier saw Laplacian discretized via *dual mesh*
- Different duals lead to operators with different accuracy
- Space of *orthogonal duals* explored by [Mullen et al., 2011]

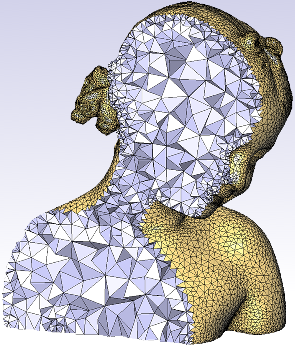
Dual Mesh



barycentric *circumcentric* *(superimposed)*

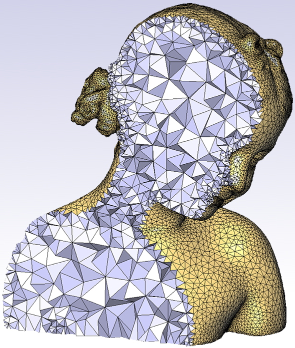
- Earlier saw Laplacian discretized via *dual mesh*
- Different duals lead to operators with different accuracy
- Space of *orthogonal duals* explored by [Mullen et al., 2011]
- Leads to many applications in geometry processing
  [de Goes et al., 2012, de Goes et al., 2013, de Goes et al., 2014]

Volumes / Tetrahedral Meshes



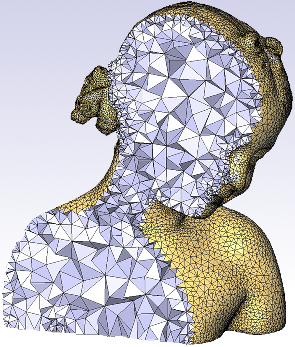
- Same problems (Poisson, Laplace, etc.) can also be solved on volumes

Volumes / Tetrahedral Meshes



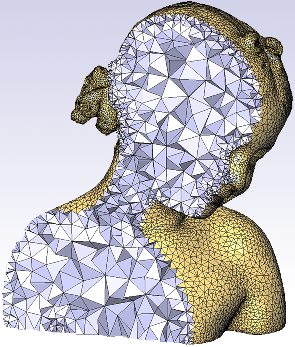
- Same problems (Poisson, Laplace, etc.) can also be solved on volumes
- Popular choice: *tetrahedral* meshes (graded, conform to boundary, . . .)

Volumes / Tetrahedral Meshes



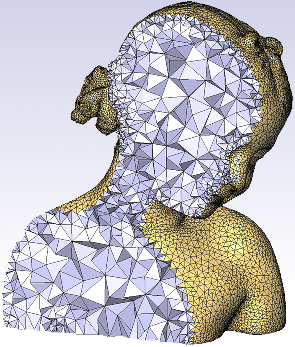
- Same problems (Poisson, Laplace, etc.) can also be solved on volumes
- Popular choice: *tetrahedral* meshes (graded, conform to boundary, . . .)
- Many ways to get Laplace matrix

Volumes / Tetrahedral Meshes



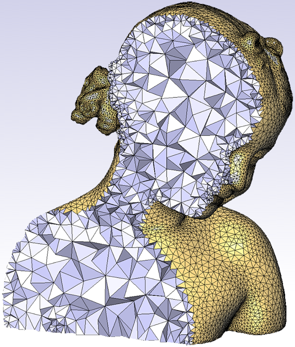
- Same problems (Poisson, Laplace, etc.) can also be solved on volumes
- Popular choice: *tetrahedral* meshes (graded, conform to boundary, ...)
- Many ways to get Laplace matrix
- One nice way: discrete exterior calculus (DEC) [Hirani, 2003, Desbrun et al., 2005]

Volumes / Tetrahedral Meshes



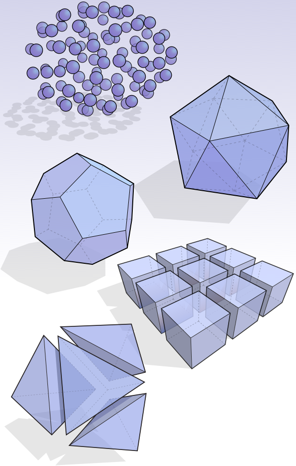
- Same problems (Poisson, Laplace, etc.) can also be solved on volumes
- Popular choice: *tetrahedral* meshes (graded, conform to boundary, ...)
- Many ways to get Laplace matrix
- One nice way: discrete exterior calculus (DEC) [Hirani, 2003, Desbrun et al., 2005]
- Just incidence matrices (e.g., which tets contain which triangles?) & primal / dual volumes (area, length, etc.).

Volumes / Tetrahedral Meshes



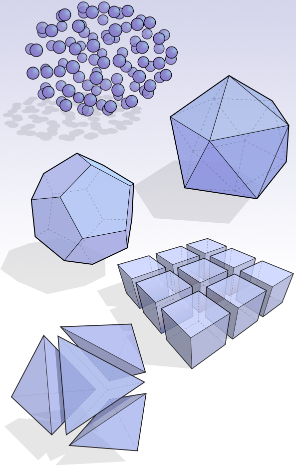
- Same problems (Poisson, Laplace, etc.) can also be solved on volumes
- Popular choice: *tetrahedral* meshes (graded, conform to boundary, ...)
- Many ways to get Laplace matrix
- One nice way: discrete exterior calculus (DEC) [Hirani, 2003, Desbrun et al., 2005]
- Just incidence matrices (e.g., which tets contain which triangles?) & primal / dual volumes (area, length, etc.).
- Added bonus: play with definition of dual to improve accuracy [Mullen et al., 2011].

...and More!



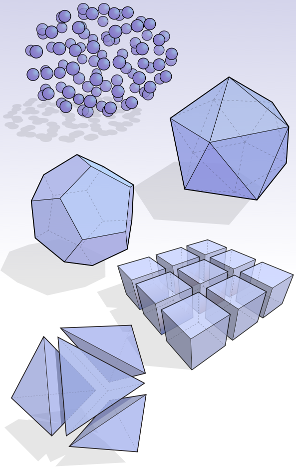
- Covered some standard discretizations

...and More!



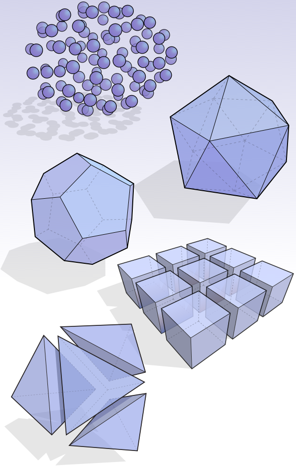
- Covered some standard discretizations
- Many possibilities (level sets, hex meshes…)

...and More!



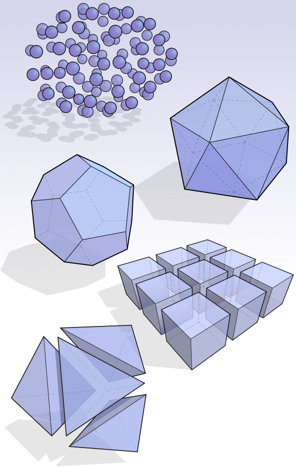
- Covered some standard discretizations
- Many possibilities (level sets, hex meshes…)
- Often enough to have *gradient* G and inner product W.

...and More!



- Covered some standard discretizations
- Many possibilities (level sets, hex meshes...)
- Often enough to have *gradient* G and inner product W.
- (weak!) Laplacian is then $\mathsf{C} = \mathsf{G}^{\mathsf{T}}\mathsf{W}\mathsf{G}$ (think Dirichlet energy)

...and More!



- Covered some standard discretizations
- Many possibilities (level sets, hex meshes...)
- Often enough to have *gradient* G and inner product W.
- (weak!) Laplacian is then $\mathsf{C} = \mathsf{G}^{\mathsf{T}}\mathsf{W}\mathsf{G}$ (think Dirichlet energy)
- Key message:
  build Laplace; do lots of cool stuff.

Applications

$$\{\text{simple pre-processing}\} \longrightarrow \Delta^{(-1)} \longrightarrow \{\text{simple post-processing}\}$$

"Our method boils down to 'backslash' in Matlab!"

Reminder: Model Equations

$$\boxed{\Delta f = 0}$$

Laplace equation

Linear solve

$$\boxed{\Delta f = g}$$

Poisson equation

Linear solve

$$\boxed{f_t = \Delta f}$$

Heat equation

ODE time-step

$$\boxed{\Delta \phi_i = \lambda_i \phi_i}$$

Vibration modes

Eigenproblem

Look here!

Reminder: Model Equations

$$\boxed{\Delta f = 0}$$

Laplace equation

Linear solve

$$\boxed{\Delta f = g}$$

Poisson equation

Linear solve

$$\boxed{f_t = \Delta f}$$

Heat equation

ODE time-step

$$\boxed{\Delta \phi_i = \lambda_i \phi_i}$$

Vibration modes

Eigenproblem

Reminder: Model Equations

$$\boxed{\Delta f = 0}$$

Laplace equation

Linear solve

$$\boxed{\Delta f = g}$$

Poisson equation

Linear solve

$$\boxed{f_t = \Delta f}$$

Heat equation

ODE time-step

$$\boxed{\Delta \phi_i = \lambda_i \phi_i}$$

Vibration modes

Eigenproblem

$\boxed{\Delta f = 0}$

Reminder: Variational Interpretation

$$\min_{f(x)} \int_\Sigma \|\nabla f(x)\|^2 \, dA$$

$\updownarrow$ <calculus>

$$\boxed{\Delta f(x) = 0}$$

$\boxed{\Delta f = 0}$

Reminder: Variational Interpretation

$$\min_{f(x)} \int_\Sigma \|\nabla f(x)\|^2 \, dA$$

$\updownarrow$ <calculus>

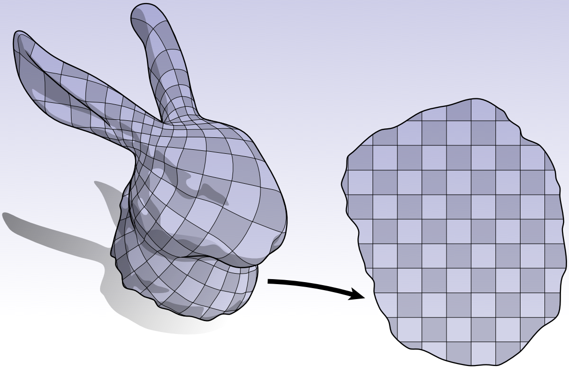
$$\boxed{\Delta f(x) = 0}$$

The (inverse) Laplacian **wants** to make functions smooth.

"Elliptic regularity"

$\boxed{\Delta f = 0}$

Application: Mesh Parameterization



Want **smooth** $f : M \to \mathbb{R}^2$.

$\boxed{\Delta f = 0}$

Variational Approach

$$\min_{f:M\rightarrow\mathbb{R}^2} \int \|\nabla f\|^2$$

Does this work?

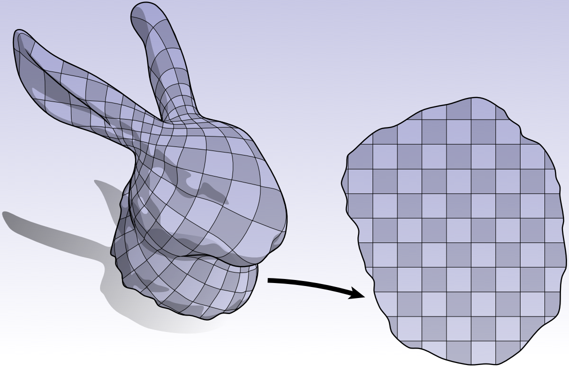
$$\min_{f:M\to\mathbb{R}^2} \int \|\nabla f\|^2$$

Does this work?

$$f(x) \equiv \text{const.}$$

$\boxed{\Delta f = 0}$

Harmonic Parameterization

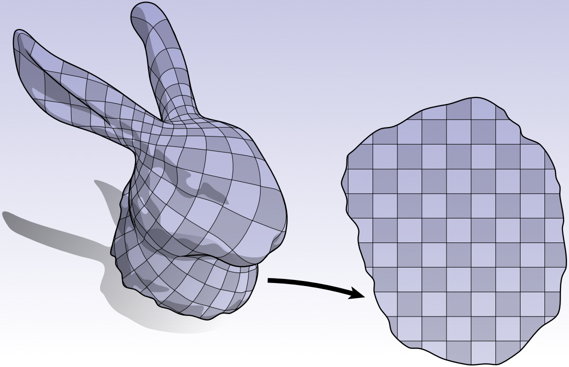


$$\min_{\substack{f:M\to\mathbb{R}^2 \\ f|_{\partial M}\ \text{fixed}}} \int \|\nabla f\|^2$$

[Eck et al., 1995]

$\boxed{\Delta f = 0}$

Harmonic Parameterization



$$\min_{\substack{f:M\to\mathbb{R}^2 \\ f|_{\partial M}\text{ fixed}}} \int \|\nabla f\|^2$$

[Eck et al., 1995]

$$\Delta f = 0 \text{ in } M\backslash\partial M, \text{ with } f|_{\partial M} \text{ fixed}$$

Reminder: Model Equations

$$\boxed{\Delta f = 0}$$

Laplace equation

Linear solve

$$\boxed{\Delta f = g}$$

Poisson equation

Linear solve

$$\boxed{f_t = \Delta f}$$

Heat equation

ODE time-step

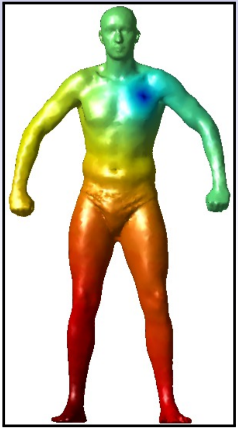
$$\boxed{\Delta \phi_i = \lambda_i \phi_i}$$

Vibration modes

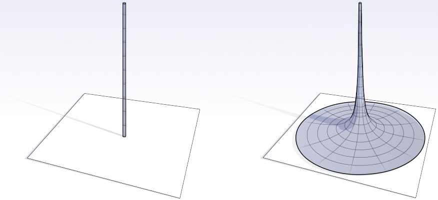
Eigenproblem

$$\boxed{\Delta f = g}$$

Recall: Green's Function



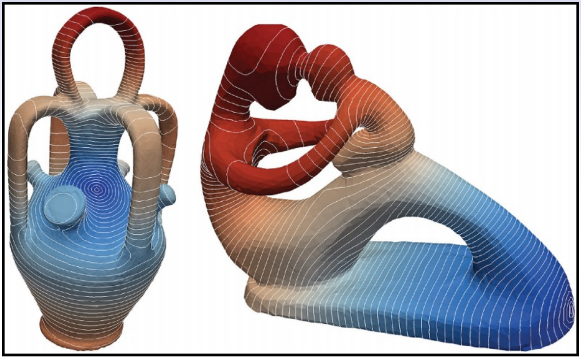
$$\Delta g_p = \delta_p \text{ for } p \in M$$



$\boxed{\Delta f = g}$

Application: Biharmonic Distances

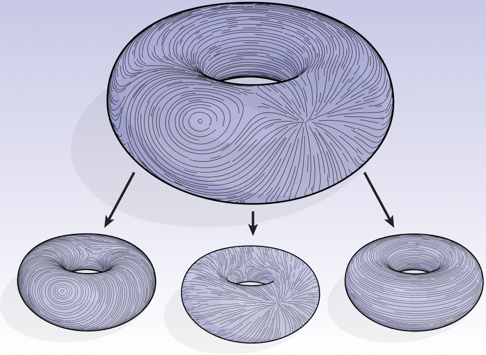
$$d_b(p,q) \equiv \|g_p - g_q\|_2$$



[Lipman et al., 2010], formula in [Solomon et al., 2014]

$\boxed{\Delta f = g}$

Hodge Decomposition



$$\vec{v}(x) = R^{90^\circ}\nabla g + \nabla f + \vec{h}(x)$$

- Divergence-free part: $R^{90^\circ}\nabla g$
- Curl-free part: ∇f
- Harmonic part: $\vec{h}(x)$ ($= \vec{0}$ if surface has no holes)

Computing the Curl-Free Part

$$\min_{f(x)} \int_\Sigma \|\nabla f(x) - \vec{v}(x)\|^2 \, dA$$

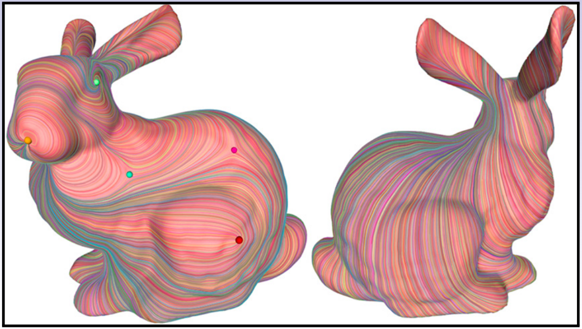
$\updownarrow$ `<calculus>`

$$\boxed{\Delta f(x) = \nabla \cdot \vec{v}(x)}$$

Get divergence-free part as $\vec{v}(x) - \nabla f(x)$ (when $\vec{h} \equiv \vec{0}$)

$\boxed{\Delta f = g}$

Application: Vector Field Design

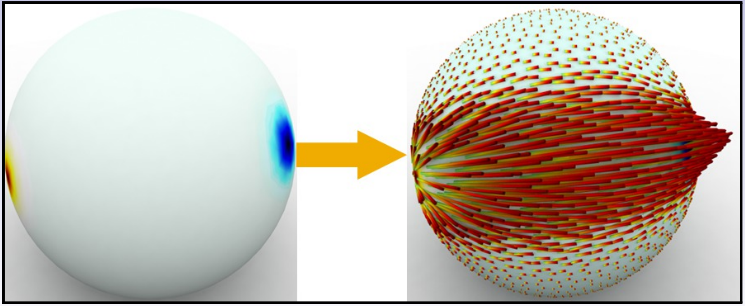


$$\Delta f = -\bar{K} \longrightarrow \vec{v}(x) = \nabla f(x)$$

[Crane et al., 2010, de Goes and Crane, 2010]

$\boxed{\Delta f = g}$

Application: Earth Mover's Distance



$$\min_{\vec{J}(x)} \int_M \|\vec{J}(x)\|$$

$$\text{such that } \vec{J} = R^{90^\circ}\nabla g + \nabla f + \vec{h}(x)$$

$$\Delta f = \rho_1 - \rho_0$$

[Solomon et al., 2014]

Reminder: Model Equations

$$\boxed{\Delta f = 0}$$

Laplace equation

Linear solve

$$\boxed{\Delta f = g}$$

Poisson equation

Linear solve

$$\boxed{f_t = \Delta f}$$

Heat equation

ODE time-step

$$\boxed{\Delta \phi_i = \lambda_i \phi_i}$$

Vibration modes

Eigenproblem

$\boxed{f_t = \Delta f}$

Generalizing Gaussian Blurs

Gradient descent on $\int \|\nabla f(x)\|^2 \, dx$:

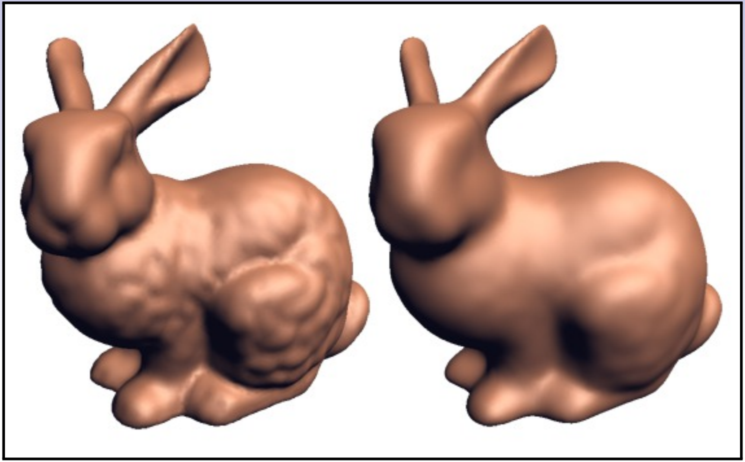
$$\frac{\partial f(x,t)}{\partial t} = \Delta_x f(x,t)$$

with $f(\cdot, 0) \equiv f_0(\cdot)$.

Image by M. Bottazzi

$\boxed{f_t = \Delta f}$

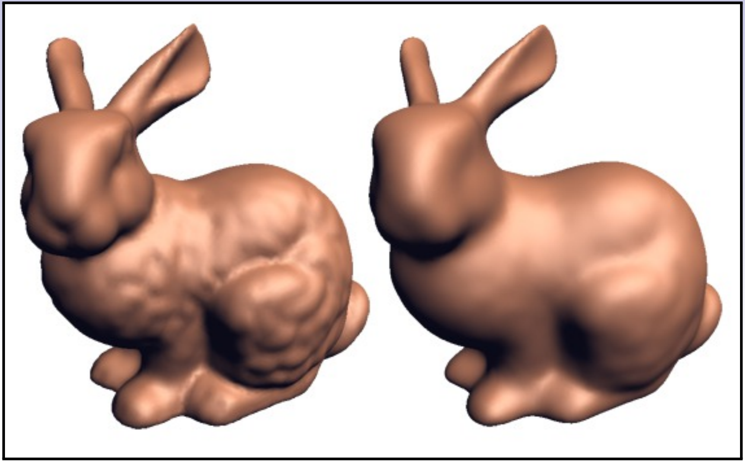
Application: Implicit Fairing



Idea: Take $f_0(x)$ to be the coordinate function.

$\boxed{f_t = \Delta f}$

Application: Implicit Fairing



Idea: Take $f_0(x)$ to be the coordinate function.

Detail: Δ changes over time.

[Desbrun et al., 1999]

$\boxed{\Delta f = g}$

Alternative: Screened Poisson Smoothing

Simplest incarnation of [Chuang and Kazhdan, 2011]:

$$\min_{f(x)} \alpha^2 \|f - f_0\|^2 + \|\nabla f\|^2$$

$$\updownarrow$$

$$(\alpha^2 I - \Delta) f = \alpha^2 f_0$$

(Semi-)Implicit Euler:

$$(I - hL)u_{k+1} = u_k$$

Screened Poisson:

$$(\alpha^2 I - \Delta)f = \alpha^2 f_0$$

$\boxed{f_t = \Delta f \to \Delta f = g}$

Interesting Connection

(Semi-)Implicit Euler:

$$(I - hL)u_{k+1} = u_k$$

Screened Poisson:

$$(\alpha^2 I - \Delta)f = \alpha^2 f_0$$

One time step of *implicit Euler* is *screened Poisson*.

$\boxed{f_t = \Delta f \to \Delta f = g}$

(Semi-)Implicit Euler:

$$(I - hL)u_{k+1} = u_k$$

Screened Poisson:

$$(\alpha^2 I - \Delta)f = \alpha^2 f_0$$

One time step of *implicit Euler* is *screened Poisson*.

Accidentally replaced one PDE with another!

Eikonal equation for geodesics:

$$\|\nabla \phi\|_2 = 1$$

$$\implies \text{Need } \textit{direction} \text{ of } \nabla \phi.$$

Eikonal equation for geodesics:

$$\|\nabla \phi\|_2 = 1$$

$$\implies \text{Need } \textit{direction} \text{ of } \nabla\phi.$$

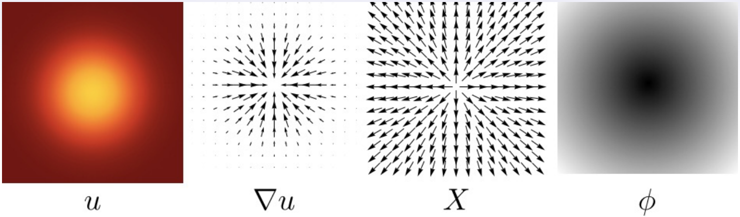
Idea:

Find u such that ∇u is *parallel* to geodesic.

$\boxed{f_t = \Delta f \text{ and } \Delta f = g}$

Application: The "Heat Method"

1. Integrate $u' = \nabla u$ (heat equation) to time $t \ll 1$.
2. Define vector field $X \equiv -\frac{\nabla u}{\|\nabla u\|_2}$.
3. Solve least-squares problem $\nabla \phi \approx X \iff \Delta \phi = \nabla \cdot X$.



Blazingly fast!

[Crane et al., 2013b]

Reminder: Model Equations

$$\boxed{\Delta f = 0}$$ *Laplace equation*

Linear solve

$$\boxed{\Delta f = g}$$ *Poisson equation*

Linear solve

$$\boxed{f_t = \Delta f}$$ *Heat equation*

ODE time-step

$$\boxed{\Delta \phi_i = \lambda_i \phi_i}$$ *Vibration modes*

Eigenproblem

$$\boxed{\Delta\phi_i = \lambda_i\phi_i}$$

Laplace-Beltrami Eigenfunctions

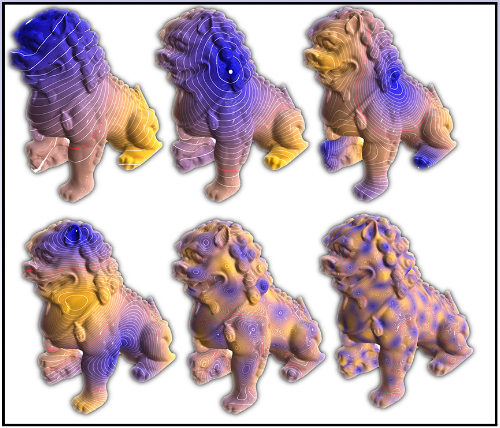


Image by B. Vallet and B. Lévy

Use eigenvalues and eigenfunctions to characterize shape.

$\boxed{\Delta\phi_i = \lambda_i\phi_i}$

Intrinsic Laplacian-Based Descriptors

All computable from eigenfunctions!

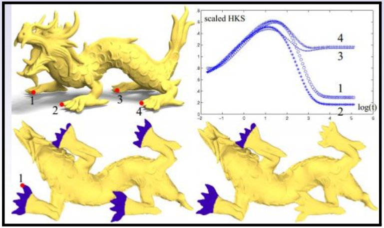
- $\text{HKS}(x;t) = \sum_i e^{\lambda_i t}\phi_i(x)^2$ [Sun et al., 2009]
- $\text{GPS}(x) = \left(\frac{\phi_1(x)}{\sqrt{-\lambda_1}}, \frac{\phi_2(x)}{\sqrt{-\lambda_2}}, \ldots\right)$ [Rustamov, 2007]
- $\text{WKS}(x;e) = C_e \sum_i \phi_i(x)^2 \exp\left(-\frac{1}{2\sigma^2}(e - \log(-\lambda_i))\right)$ [Aubry et al., 2011]

*Many others—or **learn** a function of eigenvalues!*
[Litman and Bronstein, 2014]

$\boxed{f_t = \Delta f}$

Example: Heat Kernel Signature

Heat diffusion encodes geometry for **all** times $t \geq 0$!



[Sun et al., 2009]

$$\mathrm{HKS}(x;t) \equiv k_t(x,x)$$

"Amount of heat diffused from x to itself over at time t."

- Signature of point x is a function of $t \geq 0$
- *Intrinsic* descriptor

$$\boxed{\Delta\phi_i = \lambda_i\phi_i}$$

HKS via Laplacian Eigenfunctions

$$\Delta\phi_i = \lambda_i\phi_i, f_0(x) = \sum_i a_i\phi_i(x)$$

$$\frac{\partial f(x,t)}{\partial t} = \Delta f \text{ with } f(x,0) \equiv f_0(x)$$

$\boxed{\Delta\phi_i = \lambda_i \phi_i}$

HKS via Laplacian Eigenfunctions

$$\Delta\phi_i = \lambda_i\phi_i, f_0(x) = \sum_i a_i \phi_i(x)$$

$$\frac{\partial f(x,t)}{\partial t} = \Delta f \text{ with } f(x,0) \equiv f_0(x)$$

$$\implies f(x,t) = \sum_i a_i e^{\lambda_i t} \phi_i(x)$$

$\boxed{\Delta\phi_i = \lambda_i\phi_i}$

HKS via Laplacian Eigenfunctions

$$\Delta\phi_i = \lambda_i\phi_i, f_0(x) = \sum_i a_i\phi_i(x)$$

$$\frac{\partial f(x,t)}{\partial t} = \Delta f \text{ with } f(x,0) \equiv f_0(x)$$

$$\implies f(x,t) = \sum_i a_i e^{\lambda_i t}\phi_i(x)$$

$$\implies \text{HKS}(x;t) \equiv k_t(x,x) = \sum_i e^{\lambda_i t}\phi_i(x)^2$$

$$\boxed{\Delta\phi_i = \lambda_i\phi_i}$$

Application: Shape Retrieval

Solve problems like ***shape similarity search****.*

"Shape DNA" [Reuter et al., 2006]:
Identify a shape by its vector of Laplacian eigenvalues

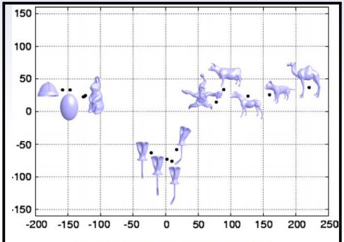
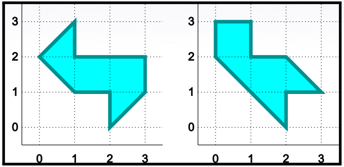
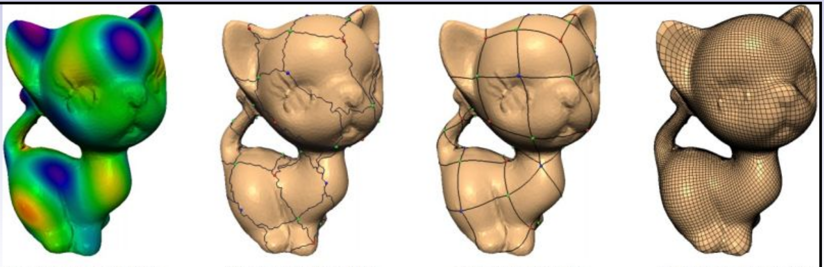


Fig. 19. 2d MDS plot of mesh Shape-DNAs.



$$\boxed{\Delta\phi_i = \lambda_i\phi_i}$$

Different Application: Quadrangulation



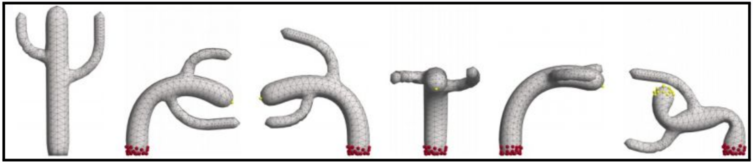
(a) Laplacian eigenfunction (b) Morse-Smale complex (c) Optimized complex (d) Semi-regular remeshing

Connect critical points (well-spaced) of ϕ_i
in *Morse-Smale complex*.

[Dong et al., 2006]

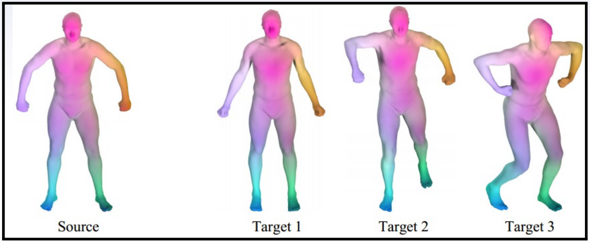
Other Ideas I

- **Mesh editing:** Displacement of vertices and parameters of a deformation should be *smooth* functions along a surface [Sorkine et al., 2004, Sorkine and Alexa, 2007] (and many others)



Other Ideas II

- **Surface reconstruction:** Poisson equation helps distinguish inside and outside [Kazhdan et al., 2006]
- **Regularization for mapping:** To compute $\phi : M_1 \to M_2$, ask that $\phi \circ \Delta_1 \approx \Delta_2 \circ \phi$ [Ovsjanikov et al., 2012]



http://ddg.cs.columbia.edu/SGP2014/LaplaceBeltrami.pdf

Alexa, M. and Wardetzky, M. (2011).
Discrete laplacians on general polygonal meshes.
ACM Trans. Graph., 30(4).

Alon, N., Karp, R., Peleg, D., and West, D. (1995).
A graph-theoretic game and its application to the k-server problem.
SIAM Journal on Computing, 24:78–100.

Aubry, M., Schlickewei, U., and Cremers, D. (2011).
The wave kernel signature: A quantum mechanical approach to shape analysis.
In *Proc. ICCV Workshops*, pages 1626–1633.

Belkin, M., Sun, J., and Wang, Y. (2009).
Constructing laplace operator from point clouds in rd.
In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 1031–1040, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Bobenko, A. I. and Springborn, B. A. (2005).
A discrete Laplace-Beltrami operator for simplicial surfaces.
ArXiv Mathematics e-prints.

Bommes, D., Lévy, B., Pietroni, N., Puppo, E., Silva, C., Tarini, M., and Zorin, D. (2013).
Quad-mesh generation and processing: A survey.
Computer Graphics Forum, 32(6):51–76.

Bronstein, A. M., Bronstein, M. M., Guibas, L. J., and Ovsjanikov, M. (2011).
Shape Google: Geometric words and expressions for invariant shape retrieval.
ACM Trans. Graph., 30(1):1:1–1:20.

Chuang, M. and Kazhdan, M. (2011).
Interactive and anisotropic geometry processing using the screened Poisson equation.
ACM Trans. Graph., 30(4):57:1–57:10.

Crane, K., de Goes, F., Desbrun, M., and Schröder, P. (2013a).
Digital geometry processing with discrete exterior calculus.

In *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, pages 7:1–7:126, New York, NY, USA. ACM.

Crane, K., Desbrun, M., and SchrÃűder, P. (2010).
Trivial connections on discrete surfaces.
Computer Graphics Forum, 29(5):1525–1533.

Crane, K., Weischedel, C., and Wardetzky, M. (2013b).
Geodesics in heat: A new approach to computing distance based on heat flow.
ACM Trans. Graph., 32.

de Goes, F., Alliez, P., Owhadi, H., and Desbrun, M. (2013).
On the equilibrium of simplicial masonry structures.
ACM Trans. Graph., 32(4):93:1–93:10.

de Goes, F., Breeden, K., Ostromoukhov, V., and Desbrun, M. (2012).
Blue noise through optimal transport.
ACM Trans. Graph., 31.

de Goes, F. and Crane, K. (2010).
Trivial connections on discrete surfaces revisited: A simplified algorithm for simply-connected surfaces.

de Goes, F., Liu, B., Budninskiy, M., Tong, Y., and Desbrun, M. (2014).
Discrete 2-tensor fields on triangulations.
Symposium on Geometry Processing.

Desbrun, M., Kanso, E., and Tong, Y. (2005).
Discrete differential forms for computational modeling.
In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA. ACM.

Desbrun, M., Meyer, M., Schröder, P., and Barr, A. H. (1999).
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, pages 317–324, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Dong, S., Bremer, P.-T., Garland, M., Pascucci, V., and Hart, J. C. (2006).

Spectral surface quadrangulation.
ACM Trans. Graph., 25(3):1057–1066.

Duffin, R. (1959).
Distributed and lumped networks.
Journal of Mathematics and Mechanics, 8:793–826.

Dunyach, M., Vanderhaeghe, D., Barthe, L., and Botsch, M. (2013).
Adaptive Remeshing for Real-Time Mesh Deformation.
In *Proceedings of Eurographics Short Papers*, pages 29–32.

Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., and Stuetzle, W. (1995).
Multiresolution analysis of arbitrary meshes.
In *Proc. SIGGRAPH*, pages 173–182.

Gillman, A. and Martinsson, P.-G. (2013).
A direct solver with O(N) complexity for variable coefficient elliptic PDEs discretized via a high-order composite spectral collocation method.
SIAM Journal on Scientific Computation.

Hirani, A. (2003).
Discrete exterior calculus.

Kazhdan, M., Bolitho, M., and Hoppe, H. (2006).
Poisson surface reconstruction.
In *Proc. SGP*, pages 61–70. Eurographics Association.

Koutis, I., Miller, G., and Peng, R. (2011).
A nearly $m \log n$ time solver for sdd linear systems.
pages 590–598.

Krishnan, D., Fattal, R., and Szeliski, R. (2013).
Efficient preconditioning of laplacian matrices for computer graphics.
ACM Trans. Graph., 32(4):142:1–142:15.

Lavoué, G. (2011).

Bag of words and local spectral descriptor for 3d partial shape retrieval.
In *Proc. 3DOR*, pages 41–48. Eurographics Association.

Li, X. and Godil, A. (2009).
Exploring the bag-of-words method for 3d shape retrieval.
In *Proc. ICIP*, pages 437–440.

Lipman, Y., Rustamov, R. M., and Funkhouser, T. A. (2010).
Biharmonic distance.
ACM Trans. Graph., 29(3):27:1–27:11.

Litman, R. and Bronstein, A. M. (2014).
Learning spectral descriptors for deformable shape correspondence.
PAMI, 36(1):171–180.

Liu, Y., Prabhakaran, B., and Guo, X. (2012).
Point-based manifold harmonics.
IEEE Trans. Vis. Comput. Graph., 18(10):1693–1703.

MacNeal, R. (1949).
The solution of partial differential equations by means of electrical networks.

Mullen, P., Memari, P., de Goes, F., and Desbrun, M. (2011).
Hot: Hodge-optimized triangulations.
ACM Trans. Graph., 30(4):103:1–103:12.

Ovsjanikov, M., Ben-Chen, M., Solomon, J., Butscher, A., and Guibas, L. (2012).
Functional maps: A flexible representation of maps between shapes.
ACM Trans. Graph., 31(4):30:1–30:11.

Pinkall, U. and Polthier, K. (1993).
Computing discrete minimal surfaces and their conjugates.
Experimental Mathematics, 2:15–36.

Reuter, M., Wolter, F.-E., and Peinecke, N. (2006).

LaplaceâĂŞBeltrami spectra as 'shape-dna' of surfaces and solids.
Computer-Aided Design, 38(4):342–366.

Rustamov, R. M. (2007).
Laplace-Beltrami eigenfunctions for deformation invariant shape representation.
In *Proc. SGP*, pages 225–233. Eurographics Association.

Solomon, J., Rustamov, R., Guibas, L., and Butscher, A. (2014).
Earth mover's distances on discrete surfaces.
In *Proc. SIGGRAPH, to appear*.

Sorkine, O. and Alexa, M. (2007).
As-rigid-as-possible surface modeling.
In *Proc. SGP*, pages 109–116. Eurographics Association.

Sorkine, O., Cohen-Or, D., Lipman, Y., Alexa, M., Rössl, C., and Seidel, H.-P. (2004).
Laplacian surface editing.
In *Proc. SGP*, pages 175–184. ACM.

Spielman, D. and Teng, S.-H. (2004).
Nearly linear time algorithms for graph partitioning, graph sparsification, and solving linear systems.
pages 81–90.

Sun, J., Ovsjanikov, M., and Guibas, L. (2009).
A concise and provably informative multi-scale signature based on heat diffusion.
In *Proc. SGP*, pages 1383–1392. Eurographics Association.

Vaidya, P. (1991).
Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners.
Workshop Talk at the IMA Workshop on Graph Theory and Sparse Matrix Computation.

Wardetzky, M., Mathur, S., Kälberer, F., and Grinspun, E. (2007).
Discrete laplace operators: No free lunch.

In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, pages 33–37, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Wojtan, C., Müller-Fischer, M., and Brochu, T. (2011).
Liquid simulation with mesh-based surface tracking.
In *ACM SIGGRAPH 2011 Courses*, SIGGRAPH '11, pages 8:1–8:84, New York, NY, USA. ACM.