Computational Aspects of Digital Fabrication

186.857, 2016SS, VU, 2.0h, 3.0ECTS

Przemyslaw Musialski
Vienna University of Technology, Austria
Geometry Processing and Geometry Modeling in the Context of 3D Printing
Meshes
Object Representations

**Raw Data**
- Range Image
- Point Cloud
- Polygon Soup

**Surfaces**
- **Polygonal Mesh**
  - Subdivision
  - Parametric

**Solids**
- Implicit
- Voxels
- CSG

**High-Level Structures**
- Scene Graph
- Semantic Parts
- Application specific
Polygonal Meshes

• Boundary representations of objects
Meshes as Approximations of Smooth Surfaces

• Piecewise linear approximation
  – Error is $O(h^2)$
  – Taylor’s theorem

3/20/2018
Computational Aspects of Digital Fabrication: Geometry Processing
Meshes as Approximations of Smooth Surfaces

- Piecewise linear approximation
  - Error is $O(h^2)$
  - Taylor’s theorem
Meshes as Approximations of Smooth Surfaces

- Piecewise linear approximation
  - Error is $O(h^2)$
  - Taylor’s theorem
  - $h$-refinement vs. $p$-refinement
  - simple objects vs. complex objects

![Diagram showing approximation error percentages for different polygonal shapes.](image)

- 3 sides: 25%
- 6 sides: 6.5%
- 12 sides: 1.7%
- 24 sides: 0.4%
Polyhedral meshes are a good representation

- approximation $O(h^2)$
- arbitrary topology
- piecewise smooth surfaces
- adaptive refinement
- efficient rendering
Polygon

- Vertices: $v_0, v_1, \ldots, v_{n-1}$
- Edges: $\{(v_0, v_1), \ldots, (v_{n-2}, v_{n-1})\}$
- Closed: $v_0 = v_{n-1}$
- Planar: all vertices on a plane
- Simple: not self-intersecting
A finite set $M$ of closed, simple polygons $Q_i$ is a polygonal mesh.

The intersection of two polygons in $M$ is either empty, a vertex, or an edge.

$M = \langle V, E, F \rangle$

- vertices
- edges
- faces
A finite set $M$ of closed, simple polygons $Q_i$ is a **polygonal mesh**

- The intersection of two polygons in $M$ is either empty, a vertex, or an edge
- Every **edge** belongs to at least one polygon
A finite set \( M \) of closed, simple polygons \( Q_i \) is a **polygonal mesh**

- The intersection of two polygons in \( M \) is either empty, a vertex, or an edge
- Every **edge** belongs to at least one polygon
- Each \( Q_i \) defines a **face** of the polygonal mesh
A finite set $M$ of closed, simple polygons $Q_i$ is a **polygonal mesh**

- The intersection of two polygons in $M$ is either empty, a vertex, or an edge
- Every **edge** belongs to at least one polygon
- Each $Q_i$ defines a **face** of the polygonal mesh
A finite set $M$ of closed, simple polygons $Q_i$ is a **polygonal mesh**.

The intersection of two polygons in $M$ is either empty, a vertex, or an edge.

Every **edge** belongs to at least one polygon.

Each $Q_i$ defines a **face** of the polygonal mesh.
A finite set $M$ of closed, simple polygons $Q_i$ is a **polygonal mesh**

- The intersection of two polygons in $M$ is either empty, a vertex, or an edge
- Every **edge** belongs to at least one polygon
- Each $Q_i$ defines a **face** of the polygonal mesh
• Vertex **degree** or **valence** = number of incident edges
Polygonal Mesh

- Vertex **degree** or **valence** = number of incident edges
**Boundary**: the set of all edges that belong to only one polygon
- Either empty or forms **closed loops**
- If empty, then the polygonal mesh is **closed**
Triangle Meshes

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

\[ V = \{v_1, \ldots, v_n\} \]

\[ E = \{e_1, \ldots, e_k\}, \quad e_i \in V \times V \]

\[ F = \{f_1, \ldots, f_m\}, \quad f_i \in V \times V \times V \]

\[ P = \{p_1, \ldots, p_n\}, \quad p_i \in \mathbb{R}^3 \]
Manifolds

- A surface is a closed **2-manifold** if it is everywhere locally homeomorphic to a disk
For every point $x$ in $M$, there is an open ball $B_x(r)$ of radius $r > 0$ centered at $x$ such that $M \cap B_x$ is homeomorphic to an open disk.

$$B_x(r) = \{ y \in \mathbb{R}^3 \text{ s.t. } \|y - x\| < r \}$$
Manifold with boundary: a vicinity of each boundary point is homeomorphic to a half-disk.
• For each case, decide if it is a 2-manifold (possibly with boundary) or not. If not, explain why not.
Democratic Manifolds 😊

- Bonus cases

Case 6

Case 7

Case 8
Manifolds

• In a manifold mesh, there are at most 2 faces sharing an edge
  – Boundary edges: have one incident face
  – Inner edges have two incident faces

• A manifold vertex has 1 connected ring of faces around it, or 1 connected half-ring (boundary)
Manifolds

• If closed, a manifold divides the space into inside and outside
• A closed manifold polygonal mesh is called polyhedron
• Every face of a polygonal mesh is orientable
  – Clockwise vs. counterclockwise order of face vertices
  – Defines sign/direction of the surface normal
Orientation

• Consistent orientation of neighboring faces:
A polygonal mesh is orientable, if the incident faces to every edge can be consistently oriented.

- If the faces are consistently oriented for every edge, the mesh is oriented.

Notes

- Every non-orientable closed mesh embedded in \( \mathbb{R}^3 \) intersects itself.
- The surface of a polyhedron is always orientable.
• **Genus**: Half the maximal number of closed paths that do not disconnect the graph.
  – Informally, the number of holes or handles.

![Genus 0](Image)
![Genus 1](Image)
![Genus 2](Image)
![Genus 3](Image)
Global Topology of Meshes

- **Genus**: Half the maximal number of closed paths that do not disconnect the graph.
  - Informally, the number of holes or handles.
• **Genus:** Half the maximal number of closed paths that do not disconnect the graph.
  – Informally, the number of holes or handles.
Global Topology of Meshes

- **Genus**: Half the maximal number of closed paths that do not disconnect the graph.
  - Informally, the number of holes or handles.

Genus 0  Genus 1

?
Theorem (Euler): The sum

\[ \chi(M) = v - e + f \]

is constant for a given surface topology, no matter which mesh we choose.

- \( v \) = number of vertices
- \( e \) = number of edges
- \( f \) = number of faces

*) \( \chi \) : Chi is pronounced /ˈkiː/
Theorem (Euler): The sum

\[ \chi(M) = v - e + f \]

is constant for a given surface topology, no matter which mesh we choose.

\[ \chi(\text{sphere}) = 2 \]
\[ \chi(\text{torus}) = 0 \]
\[ \chi(\text{disk}) = ? \]
Euler-Poincaré Formula

- For orientable meshes:

\[ v - e + f = 2(c - g) - b = \chi(M) \]

- \( c = \) number of connected components
- \( g = \) genus
- \( b = \) number of boundary loops

\[ \chi(\text{sphere}) = 2 \]
\[ \chi(\text{torus}) = 0 \]
\[ \chi(\text{circle}) = ? \]
• Let’s count the edges and faces in a closed \textbf{triangle mesh}:
  
  – Ratio of vertices to faces: $f \sim 2v$
    
    • $2 = v - e + f = v - 3/2f + f$
    • $2 + f / 2 = v$

  – Ratio of edges to vertices: $e \sim 3v$

  – Ratio of edges to faces: $e = 3/2f$
    
    • each edge belongs to exactly 2 triangles
    • each triangle has exactly 3 edges

  – \textbf{Average degree of a vertex: 6}
• **Triangle mesh**: average valence = 6
• **Quad mesh**: average valence = 4

• **Regular mesh**: all faces have the same number of edges and all vertex degrees are equal

• **Quasi-regular mesh**: a lot of **vertices** have degree 6 (4). Sometimes also refers to mostly equilateral faces.
Regularity

- Quasi-regular
  - Most vertices have valence 6
Regularity

- **Semi-regular mesh**: connectivity is a result of $N > 0$ subdivision steps
• **Semi-regular mesh:** connectivity is a result of $N > 0$ subdivision steps
Regularity

- **Semi-regular mesh:**
  - semi-regular (B): basically a set of regular patches
  - valence semi-regular (C): most vertices are of regular valence. B is always C, but not vice versa
Triangulation

- Polygonal mesh where every face is a triangle
- Simplifies data structures
- Simplifies rendering
- Simplifies algorithms
- Each face planar and convex
- Any polygon can be triangulated
Triangulation

- Polygonal mesh where every face is a triangle
- Simplifies data structures
- Simplifies rendering
- Simplifies algorithms
- Each face planar and convex
- Any polygon can be triangulated
Polygons vs. Triangle Meshes

- Triangles are flat and convex
  - Easy rasterization, normals
  - Uniformity (same # of vertices)
  - Optimized data-structures
  - 3-way symmetry is less natural

- General polygons are flexible
  - Quads have natural symmetry

- Can be non-planar, non-convex
  - Difficult for graphics hardware

- Varying number of vertices
  - More general data-structures
Data Structures

• What should be stored?
  – Geometry: 3D coordinates
  – Connectivity
    • Adjacency relationships
  – Attributes
    • Normal, color, texture coordinates
    • Per vertex, face, edge
What should be supported?

- Rendering
- Geometry queries
  - What are the vertices of face #2?
  - Is vertex A adjacent to vertex H?
  - Which faces are adjacent to face #1?

Modifications
- Remove/add a vertex/face
- Vertex split, edge collapse
Data Structures

• How good is a data structure?
  – Time to construct
  – Time to answer a query
  – Time to perform an operation
  – Space complexity
  – Redundancy

• Criteria for design
  – Expected number of vertices
  – Available memory
  – Required operations
  – Distribution of operations
Triangle List

- STL format (used in CAD)
- Storage
  - Face: 3 positions
  - 4 bytes per coordinate
  - 36 bytes per face
    - Euler: $f = 2v$
    - $72 \times v$ bytes for a mesh with $v$ vertices
- No connectivity information
- This is a “triangle soup”

<table>
<thead>
<tr>
<th>Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Indexed Face Set

- Used in formats
  - OBJ, OFF, VRML
- Storage
  - Vertex: position
  - Face: vertex indices
  - 12 bytes per vertex
  - 12 bytes per face
  - $36 \times v$ bytes for the mesh (~half of triangle list)

- No *explicit* neighborhood info

- Well suitable for rendering!

---

<table>
<thead>
<tr>
<th>Vertices</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v0</td>
<td>x0</td>
</tr>
<tr>
<td>v1</td>
<td>x1</td>
</tr>
<tr>
<td>v2</td>
<td>x2</td>
</tr>
<tr>
<td>v3</td>
<td>x3</td>
</tr>
<tr>
<td>v4</td>
<td>x4</td>
</tr>
<tr>
<td>v5</td>
<td>x5</td>
</tr>
<tr>
<td>v6</td>
<td>x6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Triangles</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>v0</td>
</tr>
<tr>
<td>t1</td>
<td>v0</td>
</tr>
<tr>
<td>t2</td>
<td>v2</td>
</tr>
<tr>
<td>t3</td>
<td>v5</td>
</tr>
</tbody>
</table>

3/20/2018
Computational Aspects of Digital Fabrication: Geometry Processing
• Information about neighbors is not explicit
  – Finding neighboring vertices/edges/faces costs $O(v)$ time!
  – Local mesh modifications cost $O(v)$

  – Breadth-first search costs $O(k*v)$
    where $k = \#$ found vertices
Neighborhood Relations

- All possible neighborhood relationships:
  1. Vertex – Vertex $VV$
  2. Vertex – Edge $VE$
  3. Vertex – Face $VF$
  4. Edge – Vertex $EV$
  5. Edge – Edge $EE$
  6. Edge – Face $EF$
  7. Face – Vertex $FV$
  8. Face – Edge $FE$
  9. Face – Face $FF$

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

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

- Introduce orientation into data structure
  - Oriented edges
- Vertex
  - Position
  - 1 outgoing halfedge index
- Halfedge
  - 1 vertex its points (index)
  - 1 incident face index
  - 3 next, prev, twin halfedge indices
- Face
  - 1 adjacent halfedge index
- Easy traversal, full connectivity
Halfedge data structure

• One-ring traversal
  – Start at vertex
Halfedge data structure

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

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

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

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

• **Cons:**
  – Heavy – requires storing and managing extra pointers (or indices)
  – Not as trivial as Indexed Face Set for rendering with OpenGL / Vertex Buffer Objects
Halfedge Libraries

• CGAL
  – www.cgal.org
  – Computational geometry

• OpenMesh
  – www.openmesh.org
  – Mesh processing

• We will not implement a half-edge data structure in the course:
  – the framework provides a basic half-edge implementation in C# based on OpenMesh
Discrete Differential Geometry
Surfaces
Differential Geometry Basics

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

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

continuous 1-1 mapping
Differential Geometry Basics

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

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

Tangents, normals, curvatures
Distances, curve angles, topology
• Continuous surface

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

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

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

• Lines on the surface when keeping one parameter fixed

\[ \gamma_{u_0}(v) = p(u_0, v) \]
\[ \gamma_{u_0}(u) = p(u, v_0) \]
Surface Normals

- Surface normal:

\[ n(u, v) = \frac{p_u \times p_v}{\| p_u \times p_v \|} \]

- Assuming *regular* parameterization, i.e.,

\[ p_u \times p_v \neq 0 \]
Normal Curvature

The normal curvature $n(u, v)$ at a point $p$ on a surface is given by:

$$ n(u, v) = \frac{p_u \times p_v}{\|p_u \times p_v\|} $$

Direction $t$ in the tangent plane (if $p_u$ and $p_v$ are orthogonal):

$$ t = \cos \varphi \frac{p_u}{\|p_u\|} + \sin \varphi \frac{p_v}{\|p_v\|} $$

Tangent plane
The curve $\gamma$ is the intersection of the surface with the plane through $n$ and $t$.

**Normal curvature:**

$$\kappa_n(\varphi) = \kappa(\gamma(p))$$

Direction $t$ in the tangent plane (if $p_u$ and $p_v$ are orthogonal):

$$t = \cos \varphi \frac{p_u}{\|p_u\|} + \sin \varphi \frac{p_v}{\|p_v\|}$$
Surface Curvatures

- **Principal curvatures**
  - Maximal curvature
  - Minimal curvature

\[ \kappa_1 = \kappa_{\text{max}} = \max_{\varphi} \kappa_n(\varphi) \]

\[ \kappa_2 = \kappa_{\text{min}} = \min_{\varphi} \kappa_n(\varphi) \]

Direction \( t \) in the tangent plane (if \( p_u \) and \( p_v \) are orthogonal):

\[ t = \cos \varphi \frac{p_u}{||p_u||} + \sin \varphi \frac{p_v}{||p_v||} \]
Euler’s Theorem: Plans of principal curvature are orthogonal and independent of parameterization.

\[ \kappa(\varphi) = \kappa_1 \cos^2 \varphi + \kappa_2 \sin^2 \varphi, \quad \varphi = \text{angle with } \mathbf{t}_1 \]

Direction \( \mathbf{t} \) in the tangent plane (if \( \mathbf{p}_u \) and \( \mathbf{p}_v \) are orthogonal):

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

Tangent plane
Principal Directions

- Principal directions: tangent vectors corresponding to $\varphi_{\text{max}}$ and $\varphi_{\text{min}}$
Principal Directions

3/20/2018
Computational Aspects of Digital Fabrication: Geometry Processing
Surface Curvatures

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

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

**Gaussian curvature:**
\[
K = \kappa_1 \cdot \kappa_2
\]
Mean Curvature

- Intuition for mean curvature:
  - integrate the curvature around the point

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

• Classify surface by Gaussian curvature $K$

• A point $p$ on the surface is called
  – Elliptic, if $K > 0$
  – Parabolic, if $K = 0$
  – Hyperbolic, if $K < 0$
  – Umbilical, if $\kappa_1 = \kappa_2$

• Developable surface
  iff $K = 0$
Local Surface Shape By Curvatures

**Isotropic:**
all directions are principal directions

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

- $K = 0$
  - planar

**Anisotropic:**
2 distinct principal directions

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

- $K = 0$
  - $\kappa_2 = 0$
    - parabolic

- $K < 0$
  - $\kappa_1 > 0$
    - hyperbolic
Visual Inspection

Mean curvature $H$  
Gaussian curvature $K$
• Gradient Operator $\nabla$:

$$\nabla f (x, y, z) = \text{grad} f = \left( \frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y} \quad \frac{\partial f}{\partial z} \right)^T$$

- the gradient is the vector of partial derivatives of $f$ at each point
- it is a (continuous) vector field
- it is a \textit{differential operator}
• Divergence Operator $\nabla \cdot$ or $\text{div}$

$$\nabla \cdot \mathbf{f} = \text{div} \mathbf{f} = \left( \frac{\partial f_x}{\partial x} + \frac{\partial f_y}{\partial y} + \frac{\partial f_z}{\partial z} \right)$$

- divergence operator applied to vector fields
- measures how much the vectors are diverging or converging at any point
Laplace Operator

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

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

\[ \text{grad} f = \nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right) \quad \text{div} \mathbf{F} = \nabla \cdot \mathbf{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z} \]

Laplace operator

gradient operator

2nd partial derivatives

function in Euclidean space

divergence operator

Cartesian coordinates
Laplace-Beltrami Operator

- Extension of Laplace to functions on manifolds

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

\[ \Delta_{\mathcal{M}} f = \text{div}_{\mathcal{M}} \nabla_{\mathcal{M}} f \]
Laplace-Beltrami Operator

- Laplace-Beltrami Operator for coordinate functions:
  \[ p(u, v) = (x(u, v), y(u, v), z(u, v))^T \]

\[ \Delta_M p = \text{div}_M \nabla_M p = -2H n \quad \in \mathbb{R}^3 \]
Laplace-Beltrami Operator

- Laplace-Beltrami Operator for coordinate functions:

\[ p(u, v) = (x(u, v), y(u, v), z(u, v))^T \]

\[ \Delta_M p = \text{div}_M \nabla_M p = -2H n \in \mathbb{R}^3 \]
Laplace-Beltrami Operator

- Laplace-Beltrami Operator for coordinate functions:

\[ p(u, v) = (x(u, v), y(u, v), z(u, v))^T \]

\[ \Delta_M p = -2H n \]
Differential Geometry on Meshes

• Assumption: meshes are piecewise linear approximations of smooth surfaces

• Can try fitting a smooth surface locally (say, a polynomial) and find differential quantities analytically

• But: it is often too slow for interactive setting and error prone
Discrete Differential Operators

• Approach: approximate differential properties at point \( v \) as spatial average over local mesh neighborhood \( N(v) \) where typically
  - \( v \) = mesh vertex
  - \( N_k(v) = k \)-ring neighborhood
Discrete Laplace-Beltrami

\[ \Delta_M p = -2Hn \]

- Uniform discretization: \( L(v) \) or \( \Delta v \)

\[
L_u(v_i) = \frac{1}{|N(i)|} \sum_{j \in N(i)} (v_j - v_i) = \left( \frac{1}{d_i} \sum_{j \in N(i)} v_j \right) - v_i
\]

- Depends only on connectivity
  = simple and efficient

- Bad approximation for irregular triangulations
Discrete Laplace-Beltrami

\[ \Delta_M p = -2H n \]

Intuition for uniform discretization

\[ H = \frac{1}{2\pi} \int_0^{2\pi} \kappa(\varphi) d\varphi \quad \kappa n = \gamma'' \]

\[ -2H n = -2 \left( \frac{1}{2\pi} \int_0^{2\pi} \kappa(\varphi) d\varphi \right) n = -\frac{1}{\pi} \int_0^{2\pi} \kappa(\varphi) n d\varphi = -\frac{1}{\pi} \int_0^{2\pi} \gamma'' d\varphi \]
Discrete Laplace-Beltrami

\[ \Delta_M \mathbf{p} = -2H \mathbf{n} \]

Intuition for uniform discretization

\[ H = \frac{1}{2 \pi} \int_0^{2\pi} \kappa(\varphi) d\varphi \]

\[ \kappa \mathbf{n} = \gamma'' \]

\[ \gamma'' \approx \frac{1}{h} \left( \frac{\mathbf{v}_{i+1} - \mathbf{v}_i}{h} - \frac{\mathbf{v}_i - \mathbf{v}_{i-1}}{h} \right) = -\frac{2}{h^2} \left( \frac{1}{2}(\mathbf{v}_{i-1} + \mathbf{v}_{i+1}) - \mathbf{v}_i \right) \]
Discrete Laplace-Beltrami

\[ \Delta_M p = -2H n \]

Intuition for uniform discretization

\[ H = \frac{1}{2\pi} \int_0^{2\pi} \kappa(\varphi) \, d\varphi \]

\[ \frac{1}{2}(v_{j1} + v_{j4}) - v_i + \]
\[ \frac{1}{2}(v_{j2} + v_{j5}) - v_i + \]
\[ \frac{1}{2}(v_{j3} + v_{j6}) - v_i = \]
\[ \frac{1}{2} \sum_{j \in \mathcal{N}(i)} v_j - 3v_i = 3 \left( \frac{1}{6} \sum_{j \in \mathcal{N}(i)} v_j - v_i \right) \]
Discrete Laplace-Beltrami

• Cotangent formula

\[ L_c(v_i) = \frac{1}{A_i} \sum_{j \in \mathcal{N}(i)} \frac{1}{2} (\cot \alpha_{ij} + \cot \beta_{ij})(v_j - v_i) \]
Unfold the triangle flap onto the plane (without distortion)
Voronoi Vertex Area

\[ c_j = \begin{cases} 
\text{circumcenter of } \triangle(v_i, v_j, v_{j+1}) & \text{if } \theta < \pi/2 \\
\text{midpoint of edge } (v_j, v_{j+1}) & \text{if } \theta \geq \pi/2 
\end{cases} \]

\[ A_i = \sum_j \text{Area } (\triangle(v_i, c_j, c_{j+1})) \]
Discrete Laplace-Beltrami

- Cotangent formula

\[ L_c(v_i) = \frac{1}{A_i} \sum_{j \in \mathcal{N}(i)} \frac{1}{2} (\cot \alpha_{ij} + \cot \beta_{ij})(v_j - v_i) \]

- Accounts for mesh geometry

- Potentially negative/infinite weights
Discrete Laplace-Beltrami

- Cotangent formula

\[ L_c(v_i) = \frac{1}{A_i} \sum_{j \in \mathcal{N}(i)} \frac{1}{2} (\cot \alpha_{ij} + \cot \beta_{ij}) (v_j - v_i) \]

- Can be derived using linear Finite Elements
- Nice property: gives zero for planar 1-rings!
Triangle Areas Cheat Sheet

\[ \cot \theta = \frac{u \cdot v}{||u \times v||} \]

\[ A = \frac{1}{2} uv \sin(\alpha + \beta) \]

\[ W = \frac{1}{8} (u^2 \cot \alpha + v^2 \cot \beta) \]

\[ D = \frac{1}{4} \ell^2 (\cot \alpha + \cot \beta) \]

\[ V = \frac{1}{4} \ell^2 \cot \alpha \]

Keenan Crane (keenan@cs.caltech.edu)
Discrete Laplace-Beltrami

- **Uniform Laplacian** $L_u(v_i)$
- **Cotangent Laplacian** $L_c(v_i)$
- **Mean curvature normal**
Discrete Laplace-Beltrami

- **Uniform Laplacian** $L_u(v_i)$
- **Cotangent Laplacian** $L_c(v_i)$
- **Mean curvature normal**
- **For nearly equal edge lengths:**
  - Uniform $\approx$ Cotangent
Discrete Laplace-Beltrami

- **Uniform Laplacian** $L_u(v_i)$
- **Cotangent Laplacian** $L_c(v_i)$
- **Mean curvature normal**
- **For nearly equal edge lengths**
  - *Uniform* $\approx$ *Cotangent*

Cotan Laplacian allows computing discrete normal
Surface Curvatures

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

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

Gaussian curvature: \[ K = \kappa_1 \cdot \kappa_2 \]
Discrete Curvatures

**Principal curvatures:**

\[ \kappa_1 = H + \sqrt{H^2 - K} \quad \kappa_2 = H - \sqrt{H^2 - K} \]

**Mean curvature:**

\[ H(v_i) = \frac{\| L_c(v_i) \|}{2} \]

**Gaussian curvature:**

\[ K(v_i) = \frac{1}{A_i} (2\pi - \sum_j \theta_j) \]
Example: Discrete Mean Curvature
Links and Literature

• M. Meyer, M. Desbrun, P. Schroeder, A. Barr
Recap: Linear Algebra and Optimization

Przemyslaw Musialski
Institute of Computer Graphics and Algorithms
Vienna University of Technology, Austria
Linear Equations

- A system of linear equations

\[
\begin{align*}
1x + 2y &= 3 \\
4x + 5y &= 6
\end{align*}
\]

\[
\begin{align*}
1x + 2y &= 3 \\
4x + 8y &= 6
\end{align*}
\]

\[
\begin{align*}
1x + 2y &= 3 \\
4x + 8y &= 12
\end{align*}
\]

One solution \((x, y) = (-1, 2)\)

Parallel: No solution

Whole line of solutions
A system of linear equations

\[ 2x - y = 1 \]
\[ x + y = 5. \]

(a) Lines meet at \( x = 2, y = 3 \)
A system of linear equations

\[
\begin{align*}
2x - y &= 1 \\
 x + y &= 5.
\end{align*}
\]

\[
x \begin{bmatrix} 2 \\ 1 \end{bmatrix} + y \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \end{bmatrix}
\]

(a) Lines meet at \( x = 2, \ y = 3 \)

(b) Columns combine with 2 and 3
• **Row form**: intersection of three planes

\[
\begin{align*}
2u + v + w &= 5 \\
4u - 6v &= -2 \\
-2u + 7v + 2w &= 9.
\end{align*}
\]
Three Planes: Column Picture

- **Column form**: linear combination of three vectors

\[
\begin{bmatrix}
2 \\
4 \\
-2
\end{bmatrix}u + \begin{bmatrix}
1 \\
-6 \\
7
\end{bmatrix}v + \begin{bmatrix}
1 \\
0 \\
2
\end{bmatrix}w = \begin{bmatrix}
5 \\
-2 \\
9
\end{bmatrix}
\]

\[
\begin{bmatrix}
5 \\
1 \\
9
\end{bmatrix} = \text{linear combination equals } b
\]

\[
\begin{bmatrix}
2 \\
0 \\
4
\end{bmatrix} = 2 \begin{bmatrix}
1 \\
0 \\
2
\end{bmatrix}
\]

2 (column 3)

columns 1 + 2
• **Column form**: linear combination of columns

\[
\begin{bmatrix}
2 \\
4 \\
-2
\end{bmatrix}
+ \begin{bmatrix}
1 \\
-6 \\
7
\end{bmatrix}
+ 2 \begin{bmatrix}
1 \\
0 \\
2
\end{bmatrix}
= \begin{bmatrix}
5 \\
-2 \\
9
\end{bmatrix}
\]

\[
\begin{bmatrix}
\frac{5}{1} \\
\frac{9}{1}
\end{bmatrix}
= \text{linear combination equals } b
\]

\[
\begin{bmatrix}
\frac{2}{4} \\
\frac{-2}{7}
\end{bmatrix}
+ \begin{bmatrix}
\frac{1}{6} \\
\frac{-5}{7}
\end{bmatrix}
= \begin{bmatrix}
\frac{3}{2} \\
\frac{-2}{5}
\end{bmatrix}
\]

columns 1 + 2

\[
\begin{bmatrix}
\frac{2}{4} \\
\frac{-2}{4}
\end{bmatrix}
= 2 \begin{bmatrix}
\frac{1}{2} \\
\frac{0}{2}
\end{bmatrix}
\]

2 (column 3)
Matrix Form

- **Row form**

\[
\begin{align*}
2u + v + w &= 5 \\
4u - 6v &= -2 \\
-2u + 7v + 2w &= 9.
\end{align*}
\]

- **Column form**

\[
u \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} + v \begin{bmatrix} 1 \\ -6 \\ 7 \end{bmatrix} + w \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}
\]

- **Matrix form**

\[
Ax = b 
\begin{bmatrix}
2 & 1 & 1 \\
4 & -6 & 0 \\
-2 & 7 & 2
\end{bmatrix}
\begin{bmatrix}
u \\
v \\
w
\end{bmatrix} = 
\begin{bmatrix}
5 \\
-2 \\
9
\end{bmatrix}
\]
Matrix Multiplication

• Row times column

\[ AB = \begin{bmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  a_{21} & a_{22} & a_{23} & a_{24} \\
  a_{31} & a_{32} & a_{33} & a_{34} \\
\end{bmatrix} \begin{bmatrix}
  b_{11} \\
  b_{21} \\
  b_{31} \\
  b_{41} \\
\end{bmatrix} = \begin{bmatrix}
  * \\
  * \\
  (AB)_{32} \\
\end{bmatrix} \]
Row Space and Column Space

• Row-Vectors of the matrix form the **Row Space** \( C(A^T) \)

• Column-Vectors of the matrix form the **Column Space** \( C(A) \)

• also called Domain

\[
\begin{bmatrix}
- & - & - \\
- & - & - \\
\end{bmatrix}
\]

• also called Range

\[
\begin{bmatrix}
| & | & | \\
| & | & | \\
\end{bmatrix}
\]
A solution of the system $Ax = b$ exists iff the vector $b$ lies in the column-space $C(A)$.

\[
u \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} + v \begin{bmatrix} 1 \\ -6 \\ 7 \end{bmatrix} + w \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}
\]

\[
\begin{bmatrix} 5 \\ -1 \\ -9 \end{bmatrix} = \text{linear combination equals } b
\]

\[
\begin{bmatrix} \frac{2}{4} \\ -2 \end{bmatrix} + \begin{bmatrix} \frac{1}{6} \\ 7 \end{bmatrix} = \begin{bmatrix} \frac{3}{2} \\ \frac{-2}{5} \end{bmatrix}
\]

columns 1 + 2
A solution of the system $Ax = b$ exists iff the vector $b$ lies in the **column-space** $C(A)$.

\[
\begin{bmatrix}
1 & 2 \\
4 & 1 \\
-2 & -6
\end{bmatrix} + 1 \begin{bmatrix}
1 \\
-6 \\
7
\end{bmatrix} + 2 \begin{bmatrix}
1 \\
0 \\
2
\end{bmatrix} = \begin{bmatrix}
5 \\
-2 \\
9
\end{bmatrix}
\]

\[
\begin{bmatrix}
5 \\
-1 \\
-9
\end{bmatrix} = \text{linear combination equals } b
\]

\[
\begin{bmatrix}
2 \\
0 \\
4
\end{bmatrix} = 2 \begin{bmatrix}
1 \\
0 \\
2
\end{bmatrix}
\]

2 (column 3)

\[
\begin{bmatrix}
2 \\
4 \\
-2
\end{bmatrix} + \begin{bmatrix}
1 \\
-6 \\
7
\end{bmatrix} = \begin{bmatrix}
3 \\
-2 \\
5
\end{bmatrix}
\]

columns 1 + 2
Solving the System of Equations

• Iff there exist a unique solution the matrix $A$ has an inverse $A^{-1}$

• A system of $n$ equations with $n$ unknown can be solved with e.g. Gaussian elimination

Original system

\[
\begin{align*}
2u + v + w &= 5 \\
4u - 6v &= -2 \\
-2u + 7v + 2w &= 9.
\end{align*}
\]

Equivalent system

\[
\begin{align*}
2u + v + w &= 5 \\
-8v - 2w &= -12 \\
8v + 3w &= 14.
\end{align*}
\]

Triangular system

\[
\begin{align*}
2u + v + w &= 5 \\
-8v - 2w &= -12 \\
1w &= 2.
\end{align*}
\]
Solving the System of Equations

- Iff there exist a unique solution the matrix $A$ has a inverse $A^{-1}$

- A system of $n$ equations with $n$ unknown can be solved with e.g. Gaussian elimination

\[
\begin{bmatrix}
2 & 1 & 1 & 5 \\
4 & -6 & 0 & -2 \\
-2 & 7 & 2 & 9
\end{bmatrix} \rightarrow 
\begin{bmatrix}
2 & 1 & 1 & 5 \\
0 & -8 & -2 & -12 \\
0 & 8 & 3 & 14
\end{bmatrix} \rightarrow 
\begin{bmatrix}
2 & 1 & 1 & 5 \\
0 & -8 & -2 & -12 \\
0 & 0 & 1 & 2
\end{bmatrix}
\]

- Perform **back-substitution**!

- **Watch out for singular cases!!!**
What if the system $Ax = b$ has more equations than unknowns?

i.e. the matrix $A$ is **thin**

$$
\begin{bmatrix}
1 & 0 \\
5 & 4 \\
2 & 4
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix}
=
\begin{bmatrix}
b_1 \\
b_2 \\
b_3
\end{bmatrix}
$$

vector $b$ must lie in the column space of $A$

there might be no such vector $\rightarrow$ no solution!
Combination of columns equals $b$

$$\begin{bmatrix} 1 & 0 \\ 5 & 4 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$
Least Squares

- If there is no such combination of columns there is no exact solution
- We can still find an optimal approximate solution $\hat{x}$
- it is the perpendicular projection of $b$ on $\mathcal{C}(A)$
Least Squares

• Solve approximately means find the point $p = A\hat{x}$ in $\mathcal{C}(A)$ that has the minimal residual $e$

• i.e. smallest Euclidian distance to $b$

$$\|e\| = \|Ax - b\|$$

• this means, we are looking for the optimal $\hat{x}$ that satisfies:

$$f_0 = \min_{\hat{x}} \|Ax - b\|^2$$

Objective Function
Find Minimum?

\[
\min_{\hat{x}} \|Ax - b\|^2
\]

- Expand

\[
\|Ax - b\|^2 =
\]
• Expand

\[ \|Ax - b\|^2 = (Ax - b)^T (Ax - b) = (Ax)^T (Ax) - 2(Ax)^T b + b^T b = x^T A^T A x - 2x^T A^T b + b^T b \]

• Compute gradient

\[ \nabla_x \|e\|^2 = (A^T A + A^T A)x - 2A^T b = 2A^T Ax - 2A^T b = 0 \]

\[ \Rightarrow A^T A x = A^T b \]

Normal Equations
Pseudo-Inverse

\[ A^T A x = A^T b \]

Normal Equations

\[ \hat{x} = \left( A^T A \right)^{-1} A^T b \]

\[ \hat{x} = A^+ b \]

Pseudo-Inverse
Moore-Penrose Inverse

\[ A^+ A = I \]
Constrained Least-Squares

- often a solution is subject to constraints

\[ \min \|Ax - b\|^2 \quad \text{s.t.} \quad Bx = d \]

- Hard constraints
  - constraints that are satisfied exactly

- Soft constraints
  - constraints that are not satisfied exactly, only “as good as possible” with respect to some weight
Method of Weighting: Soft Constraints

\[
\min_{\hat{x}} ||A\hat{x} - b||^2 \quad \text{s.t.} \quad B\hat{x} = d
\]

\[
\begin{bmatrix}
A \\
\lambda B
\end{bmatrix}
\hat{x} =
\begin{bmatrix}
b \\
\lambda d
\end{bmatrix}
\]

- use the scalar \( \lambda \) to weight the constraints
- too large or too small \( \lambda \) lambda might lead to numerical instability (large condition number)
Hard Constraints

- Lagrange Multipliers:
  - Hard constraints
  - solve: \( \min_{\hat{x}} \| A\hat{x} - b \|^2 \) \( \text{s.t.} \quad Bx = d \)

- KKT-Conditions

\[
\begin{bmatrix}
A^T A & B^T \\
B & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix} =
\begin{bmatrix}
A^T b \\
d
\end{bmatrix}
\]

- if the matrix is invertible, \( x \) is the approximate solution, \( y \) are the Lagrange multipliers
- \( A, B \) must have full-column rank (i.e. \( r=n \)), i.e., the column-vectors must be linearly independent
What to do if $A$ does not have full-column rank, i.e., the column-space is not linearly independent?

- usually there will be no unique (approximate solution) but a space of solutions
- in certain cases Tchikanov-Regularization can be used:

$$\lim_{\mu \to 0} (A^T A + \mu I)^{-1} A^T = A^+$$

- regularized pseudo-inverse for small scalars $\mu$
- method should be used with caution
Least-Squares

\[ f = \min_x \|Ax - b\|_2^2 \]

- solving least-squares problems
  - analytical solution: \( x^* = (A^T A)^{-1} A^T b \)
  - reliable and efficient algorithms and libraries exist
  - mature technique / well understood

- using least-squares
  - least-squares problems are relatively easy to recognize
  - least-squares is quite flexible:
    - weighting
    - regularization
    - soft and/or hard constraints, equality & inequality constraints
Mathematical Optimization

- general (mathematical) optimization problem
  - minimize \( f_0(x) \)
  - subject to \( g_i(x) \leq b_i, \quad i = 1, \ldots, m \)
- where
  - \( x = (x_1, \ldots, x_n) \) : optimization variables
  - \( f_0 : \mathbb{R}^n \rightarrow \mathbb{R} \) : objective function
  - \( g_i : \mathbb{R}^n \rightarrow \mathbb{R}, \quad i = 1, \ldots, m \) : constraint functions

- **optimal solution** \( x^* \) has smallest value of \( f_0(x) \) among all
- that satisfy the constraints \( g_i(x) \)
Thank You

Questions