Agenda

• Machine Learning on Extrinsic Geometry (3 weeks)
  • Overview of 3D Representations
  • Geometric foundation
• Machine Learning on Different 3D Representations
  • Volumetric
  • Multi-view
  • Point cloud
  • Parametric
Shape Representation: Origin- and Application-Dependent

• Acquired real-world objects:

• Modeling “by hand”:

• Procedural modeling

• …
Shape Representations

- Points
- Polygonal meshes
Shape Representations

- Parametric surfaces
- Implicit functions
- Subdivision surfaces
POINTS
Output of Acquisition
Points

• Standard 3D data from a variety of sources
  • Often results from scanners
  • Potentially noisy

• Depth imaging (e.g. by triangulation)
• Registration of multiple images
Points

- Points = unordered set of 3-tuples
- Often converted to other reps
  - Meshes, implicits, parametric surfaces
  - Easier to process, edit and/or render
- Efficient point processing / modeling requires spatial partitioning data structure
  - Eg. to figure out neighborhoods

shading needs normals!
PARAMETRIC CURVES AND SURFACES
Parametric Representation

- Range of a function \( f : X \rightarrow Y, X \subseteq \mathbb{R}^m, Y \subseteq \mathbb{R}^n \)

- Planar curve: \( m = 1, n = 2 \)
  \[ s(t) = (x(t), y(t)) \]

- Space curve: \( m = 1, n = 3 \)
  \[ s(t) = (x(t), y(t), z(t)) \]
Parametric Representation

• Range of a function \( f : X \rightarrow Y, X \subseteq \mathbb{R}^m, Y \subseteq \mathbb{R}^n \)

• Surface in 3D: \( m = 2, n = 3 \)

\[
s(u, v) = (x(u, v), y(u, v), z(u, v))
\]
Parametric Curves

• Example: Explicit curve/circle in 2D

\[ p : \mathbb{R} \to \mathbb{R}^2 \]

\[ t \mapsto p(t) = (x(t), y(t)) \]

\[ p(t) = r (\cos(t), \sin(t)) \]

\[ t \in [0, 2\pi) \]
Parametric Surfaces

- Sphere in 3D

\[ s : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \]

\[ s(u, v) = r \left( \cos(u) \cos(v), \sin(u) \cos(v), \sin(v) \right) \]

\[ (u, v) \in [0, 2\pi) \times [-\pi/2, \pi/2] \]
Parametric Curves and Surfaces

• Advantages
  • Easy to generate points on the curve/surface
  • Separates x/y/z components

• Disadvantages
  • Hard to determine inside/outside
  • Hard to determine if a point is on the curve/surface
  • Hard to express more complex curves/surfaces!
  ➔ cue: piecewise parametric surfaces (eg. mesh)
IMPLICIT CURVES AND SURFACES
Implicit Curves and Surfaces

- Kernel of a scalar function \( f : \mathbb{R}^m \rightarrow \mathbb{R} \)
- Curve in 2D: \( S = \{ x \in \mathbb{R}^2 | f(x) = 0 \} \)
- Surface in 3D: \( S = \{ x \in \mathbb{R}^3 | f(x) = 0 \} \)

- Space partitioning

\[
\begin{align*}
\{ x \in \mathbb{R}^m | f(x) > 0 \} & \quad \text{Outside} \\
\{ x \in \mathbb{R}^m | f(x) = 0 \} & \quad \text{Curve/Surface} \\
\{ x \in \mathbb{R}^m | f(x) < 0 \} & \quad \text{Inside}
\end{align*}
\]
Implicit Curves and Surfaces

- Kernel of a scalar function $f : \mathbb{R}^m \rightarrow \mathbb{R}$
  - Curve in 2D: $S = \{x \in \mathbb{R}^2 \mid f(x) = 0\}$
  - Surface in 3D: $S = \{x \in \mathbb{R}^3 \mid f(x) = 0\}$

- Zero level set of signed distance function

![Image of a horse with different regions indicated by >=0, >0, =0, <0 indices.]
Implicit Curves and Surfaces

- Implicit circle and sphere

\[ f(x, y) = x^2 + y^2 - r^2 \]

- Implicit sphere

\[ f(x, y, z) = x^2 + y^2 + z^2 - r^2 \]
Boolean Set Operations

• Union: \[ \bigcup_i f_i(x) = \min f_i(x) \]

• Intersection: \[ \bigcap_i f_i(x) = \max f_i(x) \]
Boolean Set Operations

- Positive = outside, negative = inside
- Boolean subtraction:

<table>
<thead>
<tr>
<th></th>
<th>$f &gt; 0$</th>
<th>$f &lt; 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g &gt; 0$</td>
<td>$h &gt; 0$</td>
<td>$h &lt; 0$</td>
</tr>
<tr>
<td>$g &lt; 0$</td>
<td>$h &gt; 0$</td>
<td>$h &gt; 0$</td>
</tr>
</tbody>
</table>

- Much easier than for parametric surfaces!

$h = \max(f, -g)$
Implicit Curves and Surfaces

• Advantages
  • Easy to determine inside/outside
  • Easy to determine if a point is on the curve/surface

• Disadvantages
  • Hard to generate points on the curve/surface
  • Does not lend itself to (real-time) rendering
A related representation

- Binary volumetric grids
- Can be produced by thresholding the distance function, or from the scanned points directly
POLYGONAL MESHES
Polygonal Meshes

• Boundary representations of objects
Meshes as Approximations of Smooth Surfaces

- Piecewise linear approximation
- Error is $O(h^2)$
Polygonal Meshes

- Polygonal meshes are a good representation
  - approximation $O(h^2)$
  - arbitrary topology
  - 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
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

\[ M = \langle V, E, F \rangle \]

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.
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
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
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
Polygonal Mesh

- Vertex **degree** or **valence** = number of incident edges
Polygonal Mesh

- Vertex degree or valence = number of incident edges
Polygonal Mesh

- 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
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
Triangle Meshes

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

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

• What should be stored?
  • Geometry: 3D coordinates
  • Connectivity
    • Adjacency relationships
  • Attributes
    • Normal, color, texture coordinates
    • Per vertex, face, edge
Simple Data Structures: Triangle List

- STL format (used in CAD)
- Storage
  - Face: 3 positions
  - 4 bytes per coordinate
  - 36 bytes per face
    - on average: $f = 2v$ (**euler)
    - 72*v bytes for a mesh with v vertices
- No connectivity information

<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>
Simple Data Structures: Indexed Face Set

- Used in formats
  - OBJ, OFF, WRL
- Storage
  - Vertex: position
  - Face: vertex indices
  - 12 bytes per vertex
  - 12 bytes per face
  - $36 \times v$ bytes for the mesh
- No explicit neighborhood info

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0 (x_0, y_0, z_0)</td>
<td>t0 (v_0, v_1, v_2)</td>
</tr>
<tr>
<td>v1 (x_1, x_1, z_1)</td>
<td>t1 (v_0, v_1, v_3)</td>
</tr>
<tr>
<td>v2 (x_2, y_2, z_2)</td>
<td>t2 (v_2, v_4, v_3)</td>
</tr>
<tr>
<td>v3 (x_3, y_3, z_3)</td>
<td>t3 (v_5, v_2, v_6)</td>
</tr>
<tr>
<td>v4 (x_4, y_4, z_4)</td>
<td>(\ldots)</td>
</tr>
<tr>
<td>v5 (x_5, y_5, z_5)</td>
<td>(\ldots)</td>
</tr>
<tr>
<td>v6 (x_6, y_6, z_6)</td>
<td>(\ldots)</td>
</tr>
<tr>
<td>(\ldots)</td>
<td>(\ldots)</td>
</tr>
<tr>
<td>(\ldots)</td>
<td>(\ldots)</td>
</tr>
</tbody>
</table>

queue: halfedge data structure!
# Summary

<table>
<thead>
<tr>
<th>Parametric</th>
<th>Implicit</th>
<th>Discrete/Sampled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Splines, tensor-product</td>
<td>Distance fields</td>
<td>Meshes</td>
</tr>
<tr>
<td>surfaces</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subdivision surfaces</td>
<td>Metaballs/blobs</td>
<td>Point set surfaces</td>
</tr>
</tbody>
</table>

- Splines, tensor-product surfaces
- Subdivision surfaces
- Distance fields
- Metaballs/blobs
- Meshes
- Point set surfaces
CONVERSIONS

Implicit $\rightarrow$ Mesh
Mesh $\rightarrow$ Points (next time!)
IMPLICIT $\rightarrow$ MESH

Marching Cubes
Extracting the Surface

- Wish to compute a manifold mesh of the level set

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

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

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

Converting from implicit to explicit representations.

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

Create a triangle mesh that approximates the surface.

Lorensen and Cline, SIGGRAPH ‘87
Marching Squares (2D)

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

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

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

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

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

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

Computing the intersections:

- Edges with a sign switch contain intersections.

\[ f(x_1) < 0, f(x_2) > 0 \Rightarrow f(x_1 + t(x_2 - x_1)) = 0 \]

for some \( 0 \leq t \leq 1 \)

- Simplest way to compute \( t \): assume \( f \) is linear between \( x_1 \) and \( x_2 \):

\[ t = \frac{f(x_1)}{f(x_2) - f(x_1)} \]
Marching Squares (2D)

Connecting the intersections:

- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
Connecting the intersections:

• Grand principle: treat each cell separately!
• Enumerate all possible inside/outside combinations.
• Group those leading to the same intersections
Connecting the intersections:

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

Connecting the intersections:

Ambiguous cases:

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

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

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

explore ambiguity to avoid holes!
Marching Cubes (3D)

Main Strengths:

- Very multi-purpose.
- Extremely fast and parallelizable.
- Relatively simple to implement.
- Virtually parameter-free

Main Weaknesses:

- Can create badly shaped (skinny) triangles.
- Many special cases (implemented as big lookup tables).
- No sharp features.
Recap: Points $\rightarrow$ Implicit $\rightarrow$ Mesh

Next Time: Mesh $\rightarrow$ Point Cloud!
Software

  - MATLAB-style (flat) C++ library, based on indexed face set structure
- OpenMesh [www.openmesh.org](http://www.openmesh.org)
  - Mesh processing, based on half-edge data structure
- CGAL [www.cgal.org](http://www.cgal.org)
  - Computational geometry
  - Viewing and processing meshes
Software

- Alec Jacobson’s GP toolbox
  - [https://github.com/alecjacobson/gptoolbox](https://github.com/alecjacobson/gptoolbox)
  - MATLAB, various mesh and matrix routines
- Gabriel Peyre’s Fast Marching Toolbox
  - On-surface distances (more next time!)
- OpenFlipper [https://www.openflipper.org/](https://www.openflipper.org/)
  - Various GP algorithms + Viewer
MESH -> POINT CLOUD

Sampling
From Surface to Point Cloud - Why?

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

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

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

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

the problem:
- sampling the mesh
Farthest Point Sampling

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

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

1. Create an initial sample point set $S$
   - Image corners + additional random point.
2. Find the point which is the farthest from all point in $S$

   \[
   d(p, S) = \max_{q \in A} (d(q, S))
   \]

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

3. Insert the point to $S$ and update the distances
4. While more points are needed, iterate
Farthest Point Sampling

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

FPS on surfaces

• What’s an appropriate distance?
On-Surface Distances

- Geodesics: Straightest and *locally shortest* curves

- Euclidean distances vs. geodesic distances

- Isolines: Euclidean vs. Geodesic
Discrete Geodesics

• Recall: a mesh is a graph!
• Approximate geodesics as paths along edges

$v_0 =$ initial vertex
$d_i =$ current distance to vertex $i$
$S =$ vertices with known optimal distance

# initialize
$d_0 = 0$
$d_i = \inf \text{ for } d \text{ in } d_i$
$S = \{\}$

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

Can be asymmetric - no matter how fine the mesh!

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

Can be asymmetric - no matter how fine the mesh!

- Dijkstra as front propagation
Fast Marching Geodesics

• A better approximation: allow fronts to cross triangles!

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

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

slides credits: Daniele Panozzo
Differential Geometry Basics

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

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

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

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

manifold point

continuous 1-1 mapping

can use this mapping to calculate things!
Differential Geometry Basics

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

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

- manifold point
- continuous 1-1 mapping

- non-manifold point
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
Differential Geometry Basics

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

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

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

isolines - geodesic

another important example: curvature!
Curves

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

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

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

- A line through two points on the curve.
Secant

- A line through two points on the curve.
Tangent

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

- Secant: $p(t) - p(s)$
- Tangent: $p'(t) = (x'(t), y'(t), \ldots)^T$
- If $t$ is arc-length:
  $$\|p'(t)\| = 1$$

Recall

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

does not mean curve “geodesic”
Circle of Curvature

- Consider the circle passing through three points on the curve…
Circle of Curvature

• …the limiting circle as three points come together.
Tangent, normal, radius of curvature

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

Curvature

\( \kappa = \frac{1}{r} \)
Curvature is scale dependent

\[ \kappa = \frac{1}{r} \]
Curvature and Normal

• Assuming $t$ is arc-length parameter:

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

normal to the curve
Discrete Planar Curves
Tangents, Normals

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

- For vertices, we have many options
Tangents, Normals

• Can choose to average the adjacent edge normals

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

- Weight by edge lengths

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

- Sum of edge lengths

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

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

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

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

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

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

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

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

same as the turning angle between the edges
Curvature of a Discrete Curve

- Zero along the edges
- Turning angle at the vertices
  $\alpha_1, \alpha_2 > 0, \alpha_3 < 0$

$\alpha_1$, $\alpha_2 > 0$, $\alpha_3 < 0$