Shape Representation: Origin- and Application-Dependent

- Acquired real-world objects:
  - Discrete sampling
  - Points, meshes

- Modeling “by hand”:
  - Higher-level representations, amendable to modification, control
  - Parametric surfaces, subdivision surfaces, implicits

- Procedural modeling
  - Algorithms, grammars
  - Primitives, Polygons, Application-dependent elements
Representation Considerations

- How should we represent geometry?
  - Needs to be stored in the computer
  - Creation of new shapes
    - Input metaphors, interfaces…
  - What operations do we apply?
    - Editing, simplification, smoothing, filtering, repair…
  - How to render it?
    - Rasterization, raytracing…
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

set of raw scans
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

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

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

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

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

Bezier curves, splines

\[ s(t) = \sum_{i=0}^{n} p_i B_i^n(t) \quad B_i^n(t) = \binom{n}{i} t^i (1 - t)^{n-i} \]
Parametric Curves

Bezier curves, splines

\[ s(t) = \sum_{i=0}^{n} p_i B^n_i(t) \quad B^n_i(t) = \binom{n}{i} t^i (1 - t)^{n-i} \]

Basis functions

Curve and control polygon
Parametric Curves

Bezizer curves, splines

\[ s(t) = \sum_{i=0}^{n} p_i B_i^n(t) \quad B_i^n(t) = \binom{n}{i} t^i (1 - t)^{n-i} \]

Basis functions

Curve and control polygon
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 Surfaces

- Curve swept by another curve
  \[ s(u, v) = \sum_{i,j} p_{i,j} B_i(u) B_j(v) \]

- Bezier surface:
  \[ s(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} p_{i,j} B_i^m(u) B_j^n(v) \]

- Also: subdivision surfaces
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

\( \{ x \in \mathbb{R}^m | f(x) > 0 \} \) **Outside**

\( \{ x \in \mathbb{R}^m | f(x) = 0 \} \) **Curve/Surface**

\( \{ x \in \mathbb{R}^m | f(x) < 0 \} \) **Inside**
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: $S = \{ x \in \mathbb{R}^3 | f(x) = 0 \}$

Zero level set of signed distance function
Implicit Curves and Surfaces

Implicit circle and sphere

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

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

\[ h = \max(f, -g) \]

<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!
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 $O(h^2)$

![Graph showing #faces vs. approximation error]

<table>
<thead>
<tr>
<th>#faces</th>
<th>Approximation Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>25%</td>
</tr>
<tr>
<td>6</td>
<td>6.5%</td>
</tr>
<tr>
<td>12</td>
<td>1.7%</td>
</tr>
<tr>
<td>24</td>
<td>0.4%</td>
</tr>
</tbody>
</table>
Polygonal Meshes

- Polygonal meshes are a good representation of objects.
- 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
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
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.
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.
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

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

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

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

\[
P = \{p_1, \ldots, p_n\}, \quad p_i \in \mathbb{R}^3
\]
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**)
  - $72v$ bytes for a mesh with $v$ vertices
- No connectivity information

<table>
<thead>
<tr>
<th>Triangles</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x0</td>
<td>y0</td>
<td>z0</td>
</tr>
<tr>
<td>1</td>
<td>x1</td>
<td>x1</td>
<td>z1</td>
</tr>
<tr>
<td>2</td>
<td>x2</td>
<td>y2</td>
<td>z2</td>
</tr>
<tr>
<td>3</td>
<td>x3</td>
<td>y3</td>
<td>z3</td>
</tr>
<tr>
<td>4</td>
<td>x4</td>
<td>y4</td>
<td>z4</td>
</tr>
<tr>
<td>5</td>
<td>x5</td>
<td>y5</td>
<td>z5</td>
</tr>
<tr>
<td>6</td>
<td>x6</td>
<td>y6</td>
<td>z6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Example:**

Coordinates for a triangle:

- x0, y0, z0
- x1, x1, z1
- x2, y2, z2
- x3, y3, z3
- x4, y4, z4
- x5, y5, z5
- x6, y6, z6

...
**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

| Vertices | | 
|---|---|---|---|
| $v_0$ | $x_0$ | $y_0$ | $z_0$ |
| $v_1$ | $x_1$ | $x_1$ | $z_1$ |
| $v_2$ | $x_2$ | $y_2$ | $z_2$ |
| $v_3$ | $x_3$ | $y_3$ | $z_3$ |
| $v_4$ | $x_4$ | $y_4$ | $z_4$ |
| $v_5$ | $x_5$ | $y_5$ | $z_5$ |
| $v_6$ | $x_6$ | $y_6$ | $z_6$ |
| ... | ... | ... | ... |
| ... | ... | ... | ... |

| Triangles | | 
|---|---|---|---|
| $t_0$ | $v_0$ | $v_1$ | $v_2$ |
| $t_1$ | $v_0$ | $v_1$ | $v_3$ |
| $t_2$ | $v_2$ | $v_4$ | $v_3$ |
| $t_3$ | $v_5$ | $v_2$ | $v_6$ |
| ... | ... | ... | ... |

*queue:* halfedge datastructure!
### Summary

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

![Parametric Examples](example1.png)  
![Implicit Examples](example2.png)  
![Discrete/Sampled Examples](example3.png)
Points $\rightarrow$ Implicit
Implicit $\rightarrow$ Mesh
Mesh $\rightarrow$ Points (next time!)

CONVERSIONS
Implicit Surface Reconstruction

POINTS $\rightarrow$ IMPLICIT
Define a function

\[ f : R^3 \rightarrow R \]

with value \(< 0\) outside the shape and \(> 0\) inside
Implicit Function Approach

Define a function

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

with value $< 0$ outside the shape and $> 0$ inside
Implicit Function Approach.

Define a function

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

with value < 0 outside the shape and > 0 inside

Extract the zero-set
Implicit Function Approach

Define a function

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

with value < 0 outside the shape and > 0 inside

Extract the zero-set

\[ \{ x : f(x) = 0 \} \]
Implicit Function Approach

Define a function

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

with value < 0 outside the shape and > 0 inside

Extract the zero-set

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

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

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

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

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

- Find smooth implicit $F$.
- Scattered data interpolation:
  - $F(p_i) = 0$
  - $F$ is smooth
  - Avoid trivial $F \equiv 0$

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

- Scattered data interpolation:
  - $F(p_i) = 0$
  - $F$ is smooth
  - Avoid trivial $F \equiv 0$

- Add off-surface constraints

$F(p_i + \varepsilon n_i) = \varepsilon$
$F(p_i - \varepsilon n_i) = -\varepsilon$

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

\textbf{RBF:} Weighted sum of shifted, smooth kernels

\[ F(\mathbf{x}) = \sum_{i=0}^{N-1} w_i \varphi(\| \mathbf{x} - \mathbf{c}_i \|) \quad N = 3n \]

Scalar weights \textit{Unknowns}

Smooth kernels (basis functions) centered at constrained points. For example:

\[ \varphi(r) = r^3 \]
Radial Basis Functions Interpolation
Radial Basis Functions Interpolation

c_i
Radial Basis Functions Interpolation

\[ \varphi_i(x) = \varphi \left( \| x - c_i \| \right) \]
Radial Basis Functions Interpolation

\[ \text{dist}(\mathbf{x}) = \sum_i w_i \varphi_i(\mathbf{x}) = \sum_i w_i \varphi(\|\mathbf{x} - \mathbf{c}_i\|) \]

Kernel centers: on- and off-surface points

\[ \varphi_i(\mathbf{x}) = \varphi(\|\mathbf{x} - \mathbf{c}_i\|) \]
Radial Basis Functions Interpolation

\[ dist(x) = \sum_{i} w_i \phi_i(x) = \sum_{i} w_i \phi(\|x - c_i\|) \]

Kernel centers: on- and off-surface points

How do we find the weights?

\[ \phi_i(x) = \phi(\|x - c_i\|) \]
Radial Basis Function Interpolation

Interpolate the constraints:

\[ \{c_{3i}, c_{3i+1}, c_{3i+2}\} = \{p_i, p_i + \varepsilon n_i, p_i - \varepsilon n_i\} \]

\[ \forall j = 0, \ldots, N - 1, \quad \sum_{i=0}^{N-1} w_i \varphi(\|c_j - c_i\|) = d_j \]

\[ F(p_i) = 0 \]
\[ F(p_i + \varepsilon n_i) = \varepsilon \]
\[ F(p_i - \varepsilon n_i) = -\varepsilon \]
Radial Basis Function Interpolation

Interpolate the constraints:
\[ \{ c_{3i}, c_{3i+1}, c_{3i+2} \} = \{ p_i, p_i + \varepsilon n_i, p_i - \varepsilon n_i \} \]

Symmetric linear system to get the weights:

\[
\begin{pmatrix}
\varphi(\|c_0 - c_0\|) & \cdots & \varphi(\|c_0 - c_{N-1}\|) \\
\vdots & \ddots & \vdots \\
\varphi(\|c_{N-1} - c_0\|) & \cdots & \varphi(\|c_{N-1} - c_{N-1}\|)
\end{pmatrix}
\begin{pmatrix}
w_0 \\
\vdots \\
w_{N-1}
\end{pmatrix}
= 
\begin{pmatrix}
d_0 \\
\vdots \\
d_{N-1}
\end{pmatrix}
\]

3n equations
3n variables
RBF Kernels

$\varphi(r) = r^3$

**Triharmonic:**
- Globally supported
- Leads to dense symmetric linear system
- $C^2$ smoothness
- Works well for highly irregular sampling
RBF Kernels

Polyharmonic spline
\[ \varphi(r) = r^k \log(r), \quad k = 2, 4, 6 \ldots \]
\[ \varphi(r) = r^k, \quad k = 1, 3, 5 \ldots \]

Multiquadric
\[ \varphi(r) = \sqrt{r^2 + \beta^2} \]

Gaussian
\[ \varphi(r) = e^{-\beta r^2} \]

B-Spline (compact support)
\[ \varphi(r) = \text{piecewise-polynomial}(r) \]
RBF Reconstruction Examples

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

Insufficient number/badly placed off-surface points

Properly chosen off-surface points

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

IMPLICIT $\rightarrow$ MESH
Wish to compute a manifold mesh of the level set

- $F(x) < 0$ \(\rightarrow\) inside
- $F(x) > 0$ \(\rightarrow\) outside
- $F(x) = 0$ \(\rightarrow\) surface
Sample the SDF
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
Computing the intersections:

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

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

Chernyaev, Marching Cubes 33,'95
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

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

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

Gabriel Peyre’s Fast Marching Toolbox
  https://www.mathworks.com/matlabcentral/fileexchange/6110-toolbox-fast-marching
  On-surface distances (more next time!)

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