• Point clouds: why do we use them?
  • Many scanning devices produce them
  • Simplest graphics primitive to argue about
  • Can render directly (do we?)
  • Can manipulate directly (if desired)
• What are other geometry representations?
  • Splines?
  • Meshes?
  • Implicit surfaces?
• Describe a number of discrete representations used to encode geometric objects for modeling and simulation purposes
  • Meshes
  • Implicit surfaces
  • Point clouds
• Explain the features that make one representation better than another for certain tasks (e.g. meshes vs. implicit surfaces)
  • Static vs. dynamic topology (connectivity)
  • “Shape memory” and deformation drift
  • Regular, structured storage
  • Efficiency of geometric queries
Geometry representations

Triangular surface meshes (not volumetric)
Explicit meshes

Example:
A quadrilateral mesh

\[
\begin{array}{c}
(3, 3) \\
(-3, 3) \\
(-1, 1) \\
(1, 1) \\
(-1, -1) \\
(1, -1) \\
(-3, -3) \\
(3, -3)
\end{array}
\]
### Explicit meshes

### Vertex data structure

<table>
<thead>
<tr>
<th>Vertex ID</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>(1, 1)</td>
</tr>
<tr>
<td>P₂</td>
<td>(1, -1)</td>
</tr>
<tr>
<td>P₃</td>
<td>(-1, -1)</td>
</tr>
<tr>
<td>P₄</td>
<td>(-1, 1)</td>
</tr>
<tr>
<td>P₅</td>
<td>(3, 3)</td>
</tr>
<tr>
<td>P₆</td>
<td>(3, -3)</td>
</tr>
<tr>
<td>P₇</td>
<td>(-3, -3)</td>
</tr>
<tr>
<td>P₈</td>
<td>(-3, 3)</td>
</tr>
</tbody>
</table>
### Explicit meshes

#### Mesh data structure

<table>
<thead>
<tr>
<th>Element (Quad) ID</th>
<th>Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q₁</td>
<td>(P₁, P₂, P₃, P₄)</td>
</tr>
<tr>
<td>Q₂</td>
<td>(P₁, P₅, P₆, P₂)</td>
</tr>
<tr>
<td>Q₃</td>
<td>(P₂, P₆, P₇, P₃)</td>
</tr>
<tr>
<td>Q₄</td>
<td>(P₃, P₇, P₈, P₄)</td>
</tr>
<tr>
<td>Q₅</td>
<td>(P₄, P₈, P₅, P₁)</td>
</tr>
</tbody>
</table>
• Motivation

✓ Accelerated geometric queries for problems such as:

⇒ Is a point \((x^*,y^*)\) inside the object?

⇒ Is a point \((x^*,y^*)\) within a distance of \(d^*\) from the object surface?

⇒ What is the point on the surface which is closest to the query point \((x^*,y^*)\)?
Implicit surfaces - levelsets

- Motivation

✓ Easy modeling of motions that involve topological change, e.g. shapes splitting or merging

✓ Such operations are difficult to encode with meshes, since they don’t “split” or “merge” unless we force them to
Implicit surfaces - levelsets

- Familiar representations address some of these demands:

  ✓ e.g. Analytic equations

  ➡ For an ellipsis:

  \[ \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \]

  ➡ Easy inside/outside tests

  \[ \frac{x_*^2}{a^2} + \frac{y_*^2}{b^2} < 1 \iff (x_*, y_*) \text{ is inside} \]
Familiar representations address some of these demands:

✓ Describe a closed region via its boundary; split and reconnect when necessary

This may be tractable in isolated cases, but very cumbersome and impractical for more complicated cases, and with 3-dimensional surfaces.
• Represent a curve in 2D (or, a surface in 3D) as the zero isocontour of a (continuous) function, i.e.

\[ \mathcal{C} = \{ (x, y) \in \mathbb{R}^2 : \phi(x, y) = 0 \} \]

e.g.

circle \( x^2 + y^2 = R^2 \) \( \equiv \{ (x, y) : \phi(x, y) = 0 \} \)

where \( \phi(x, y) = x^2 + y^2 - R^2 \)
Implicit surfaces - levelsets

• This representation may seem redundant (we store information everywhere, just to capture a curve), but it conveys important benefits:

  ➡ Containment queries

  Is \((x_*, y_*)\) inside \(C\)? \(\Leftrightarrow \phi(x_*, y_*) < 0\)

  ➡ Composability

\[
\begin{align*}
\phi_1(x, y) & \text{ encodes } \Omega_1 \\
\phi_2(x, y) & \text{ encodes } \Omega_2 \\
\end{align*}
\] \(\Rightarrow\) \[
\max(\phi_1, \phi_2) \text{ encodes } \Omega_1 \cap \Omega_2
\]

  ➡ We model both shape & topology change by simply varying the level set function
Implicit surfaces - levelsets
Contouring - marching cubes