Geometric Queries
Geometric Queries—Motivation
Motivating Example:
Signal Degradation in Geometry Processing

downsampling  upsampling

Q: How can we do a better job of preserving the original signal?
Recovering Fidelity via Closest Point Projection

Idea: after resampling, project each vertex onto original mesh
Closest Point Queries

■ Q: Given a point, in space (e.g., a new sample point), how do we find the closest point on a given surface?
■ Q: Does implicit/explicit representation make this easier?
■ Q: Does our choice of mesh data structure make a difference?
■ Q: What’s the cost of the naïve algorithm?
■ Q: How do we find the distance to a single triangle anyway?
■ So many questions!
Many types of geometric queries

- Already identified need for “closest point” query
- Plenty of other things we might like to know:
  - Do two triangles intersect?
  - Are we inside or outside an object?
  - Does one object contain another?
  - ... 
- Data structures we’ve seen so far not really designed for this...
- Need some new ideas!
- TODAY: come up with simple (read: slow) algorithms.
- NEXT TIME: intelligent ways to accelerate geometric queries.
Warm up: closest point on point

- Goal is to find the point on a mesh closest to a given point.
- *Much* simpler question: given a query point \((p_1,p_2)\), how do we find the closest point on the point \((a_1,a_2)\)?

Bonus question: what’s the distance?
Slightly harder: closest point on line

- Now suppose I have a line $N^T x = c$, where $N$ is the unit normal.
- How do I find the point closest to my query point $p$?

Many ways to do it:

\[ N^T (p + tN) = c \]

\[ \iff N^T p + tN^T N = c \]

\[ \iff t = c - N^T p \]

\[ \Rightarrow p + tN = p + (c - N^T p)N \]
Harder: closest point on line segment

- Two cases: endpoint or interior
- Already have basic components:
  - point-to-point
  - point-to-line
- Algorithm?
  - find closest point on line
  - check if it’s between endpoints
  - if not, take closest endpoint
- How do we know if it’s between endpoints?
  - write closest point on line as \( a + t(b-a) \)
  - if \( t \) is between 0 and 1, it’s inside the segment!
Even harder: closest point on triangle

- What are all the possibilities for the closest point?
- Almost just minimum distance to three segments:

Q: What about a point inside the triangle?
Closest point on triangle in 3D

- Not so different from 2D case
- Algorithm?
  - project onto plane of triangle
  - use half-space tests to classify point (vs. half plane)
  - if inside the triangle, we’re done!
  - otherwise, find closest point on associated vertex or edge
- By the way, how do we find closest point on plane?
- Same expression as closest point on a line!

E.g., \( p + ( c - N^T p ) N \)
Closest point on triangle *mesh* in 3D?

- Conceptually easy:
  - loop over all triangles
  - compute closest point to current triangle
  - keep globally closest point

- Q: What’s the cost?
- What if we have *billions* of faces?
- NEXT TIME: Better data structures!
Closest point to implicit surface?

- If we change our representation of geometry, algorithms can change completely.
- E.g., how might we compute the closest point on an implicit surface described via its distance function?

One idea:
- start at the query point
- compute gradient of distance (using, e.g., finite differences)
- take a little step (decrease distance)
- repeat until we’re at the surface (zero distance)

Better yet: just store closest point for each grid cell! (speed/memory trade off)
Different query: ray-mesh intersection

- A “ray” is an oriented line starting at a point
- Think about a ray of light traveling from the sun
- Want to know where a ray pierces a surface
- Why?
  - GEOMETRY: inside-outside test
  - RENDERING: visibility, ray tracing
  - ANIMATION: collision detection
- Might pierce surface in many places!
Ray equation

- Can express ray as

\[ r(t) = o + td \]
Intersecting a ray with an implicit surface

- Recall implicit surfaces: all points $x$ such that $f(x) = 0$
- Q: How do we find points where a ray pierces this surface?
- Well, we know all points along the ray: $r(t) = o + td$
- Idea: replace “$x$” with “$r$” in 1st equation, and solve for $t$
- Example: unit sphere

$$f(x) = |x|^2 - 1$$

$$
\Rightarrow f(r(t)) = |o + td|^2 - 1
$$

$$|d|^2 t^2 + 2(o \cdot d) t + |o|^2 - 1 = 0$$

$$
t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
$$

Why two solutions?
Ray-plane intersection

- Suppose we have a plane $N^T x = c$
  - $N$ - unit normal
  - $c$ - offset
- How do we find intersection with ray $r(t) = o + td$?
- *Key idea*: again, replace the point $x$ with the ray equation $t$:
  \[N^T r(t) = c\]
- Now solve for $t$:
  \[N^T (o + td) = c \quad \Rightarrow \quad t = \frac{c - N^T o}{N^T d}\]
- And plug $t$ back into ray equation:
  \[r(t) = o + \frac{c - N^T o}{N^T d} d\]
Ray-triangle intersection

- Triangle is in a plane...
- Not much more to say!
  - Compute ray-plane intersection
  - Q: What do we do now?
  - A: Why not compute barycentric coordinates of hit point?
  - If barycentric coordinates are all positive, point in triangle
- Actually, a lot more to say... if you care about performance!
Why care about performance?

Pixar’s “Coco” — about 50 hours per frame (@24 frames/sec)
High-performance ray tracing

Intel Embree

NVIDIA OptiX

Hardware-accelerated ray tracing (RTX)
High-performance ray tracing

NVIDIA’s “Project Sol” — real time ray tracing demo (RTX)
One more query: mesh-mesh intersection

- **GEOMETRY**: How do we know if a mesh intersects itself?
- **ANIMATION**: How do we know if a collision occurred?
Warm up: point-point intersection

Q: How do we know if \( p \) intersects \( a \)?

A: ...check if they’re the same point!

Sadly, life is not always so easy.
Slightly harder: point-line intersection

- Q: How do we know if a point intersects a given line?
- A: ...plug it into the line equation!

I promise, life isn’t always so easy.
Finally interesting: line-line intersection

- Two lines: \( ax=b \) and \( cx=d \)
- Q: How do we find the intersection?
- A: See if there is a simultaneous solution
- Leads to linear system:

\[
\begin{bmatrix}
  a_1 & a_2 \\
  c_1 & c_2 \\
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
\end{bmatrix}
= 
\begin{bmatrix}
  b \\
  d \\
\end{bmatrix}
\]
Degenerate line-line intersection?

- What if lines are almost parallel?
- Small change in normal can lead to big change in intersection!
- Instability very common, very important with geometric predicates. Demands special care (e.g., analysis of matrix).

See for example Shewchuk, “Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates”
Triangle-Triangle Intersection?

- Lots of ways to do it
- Basic idea:
  - Q: Any ideas?
  - One way: reduce to edge-triangle intersection
  - Check if each line passes through plane
  - Then do interval test
- What if triangle is moving?
  - Important case for animation
  - Can think of triangles as prisms in time
  - Turns dynamic problem \((nD + \text{time})\) into purely geometric problem in \((n+1)\)-dimensions
Up Next: Spatial Acceleration Data Structures

- Testing every element is slow!
- E.g., linearly scanning through a list vs. binary search
- Can apply this same kind of thinking to geometric queries