Point Cloud Collision Detection

- Modern acquisition methods lead to modern object representations.
- Efficient rendering (splatting & ray-tracing).
- Only little work on interaction.

Goals
- Fast collision detection between point clouds.
- No polygonal reconstruction.

Surface Definition

- Approximate surface by implicit function
  \[ S = \{ x : f(x) = 0, x \in \mathbb{R}^3 \} \]
- Define \( n(x) \) by weighted least squares.
- Weight
  \[ \theta(x, p) = e^{-\frac{d(x, p)^2}{h^2}} \]
  \[ d(x, p) = \text{some distance measure} \]
- \( f(x) = n(x) \cdot (a(x) - x) \)
- Which distance measure to use?
**Geometric proximity graph:**
- nodes = points
- edges = "neighboring" points

- Approximate geodesic distance by shortest path.

- Properties:
  - Nice surface
  - Efficient evaluation
  - Implicit function throughout space
  - Surface with boundaries allowed

[Klein & Zachmann, 2004]  

**Contributions**

- Novel, fast intersection computation for point clouds
- Utilizes proximity graph
- Runtime $O(\log \log N)$, if constant number of intersection points is sufficient.
- Quality/resolution of output is adjustable.
Problem Statement

- Given two point clouds A and B (or subsets thereof),
  - decide if there is an intersection
  - construct a sampling of

\[ Z = \{ x \mid f_A(x) = f_B(x) = 0 \}. \]

Overview

1. Bracket intersections by pairs of points.
2. Find approximate intersection point (AIP) by interpolation search.
3. Refine AIP by (randomized) sampling.
1. Root Bracketing

- Goal:
  - The pairs should evenly sample the surface.
  - The two points should not be too far apart.
  - Do it without explicit spatial data structure!
- Task: construct \( n \) pairs of points (to be root brackets)

- Thought experiment:
  - Assume surface is covered by \( \alpha \) surfels.
  - Cover each surfel with at least one point from \( A \) (candidate points for root brackets)
  - For each point: try to find another point from \( A \) lying on the other side of \( B \). (completing the brackets)

Covering the Surfels

Avoid spatial data structure \( \rightarrow \) pursue probabilistic approach: occupy all \( \alpha \) surfels with high probability!

- Assumption: \( A \) is uniformly sampled.
- Lemma from paper \( \rightarrow \)
  draw \( O(\alpha \ln \alpha) \) random and independent points from \( A \cap \text{Vol}(A \cap B) \).
  Proof: see paper.

Premise: number of intersection points should be bounded by a constant.

Consequence: choose \( \alpha \) constant, or choose \( \alpha \) depending on surfels size and surface area
Completing the Brackets

• Use $f_B(p_i) - f_B(p_j) < C$ as an indicator.
• Test only points $p_j$ that
  - belong to the randomly chosen points
  - are close to each other
• Solution: SIG

Finding brackets:
$O(a \cdot \log^* d)$, where $d = \text{max. out-degree}$;
average-ase: $O(1)$

2. Interpolation Search

• Find $\hat{p} \in A$ along shortest path $\overline{p_i \hat{p}}$ in the geometric proximity graph, such that $|f_B(\hat{p})|$ is minimal.
• Utilize interpolation search! $\Rightarrow O(\log \log m)$, $m = \# \text{ elements}$
**Interpolation Search**

- Assumptions:
  - Shortest path are precomputed and stored in LUT.
  - $f_B$ is monotone along shortest path.

```
P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11
|_l = 1                      |_r = 11
```

- Interpolation parameter: $x = l + \left\lfloor \frac{-f_B(P_i)}{f_B(P_r) - f_B(P_l)}(r - l) \right\rfloor$

- Large point clouds:
  - Memory consumption could be too high.
    → compute paths on-the-fly.
  - In practice: runtime still behaves sublinear.

---

### 3. Precise Intersection Points

- Refine approximate intersection point.
  → Details: see paper...

Runtime: $O(a \ln a)$
Complexity Considerations

- constant number of brackets, as \( \alpha \) is constant
- Interpolation search: \( O(\alpha \ln a \log \log m) = O(\log \log N) \)
  (\( m = \) length of paths, is not constant!)
- Precise intersection points: \( O(\alpha \ln a) = O(1) \).
- \( f(x) \) can be evaluated in \( O(1) \).

Overall runtime: \( O(\log \log N) \)

Benchmark Scenario

- Objects are scaled uniformly \( \rightarrow \) cube size \( 2^5 \)
- Perform a full tumbling turn by a fixed, large number (5000) of small steps.
- Average collision detection time for a complete revolution.
**Minimal Bracket Density**

- If number of surfels is too small → influencing spheres in the graph are too large → likelihood increases that \( n(x) \) flips its sign without \( x \) changing sides.

- Use boolean collision queries to measure error.

![Graph showing error percentage over distance](image)

**Complexity**

- Theoretical complexity: \( O(\log \log N) \).

- Experimental complexity:

![Graph showing average time per million seconds](image)
**Timings**

- Benchmarking old vs. new method:
  - Old (RST) = brute-force sampling [EG’04]
  - iSearch = new

![Graph showing time vs. distance for RST (old) and iSearch (new)](image)

- 28,000 points

---

**Conclusion**

- **Technique:**
  - utilizes a proximity graph for collision detection and surface definition.
  - needs no BV hierarchies and no spatial partitioning data structure.
  - any BV hierarchy can be augmented by new technique to increase performance.

- **Runtime:**
  - fast (approximate) collision detection
  - overall runtime: $O(\log \log N)$ in average case.
  - speedup of factor 5–10 compared to "old" technique.

- **Quality/resolution of output (intersection points) can be adjusted**
  ($\rightarrow$ surfel radius)
Future Work

- Deformable point clouds, SIG can be updated in $O(\log^3 N)$.
- More rigorous estimation of minimal bracket density.
- Consistency of $n(x)$.
- Out-of-core collision detection.

Thank you!

Jan Klein
Heinz Nixdorf Institute and
Institute of Computer Science
University of Paderborn, Germany
janklein@uni-paderborn.de

Dr. Gabriel Zachmann
Dept. of Computer Graphics and
Virtual Reality
University of Bonn, Germany
zach@cs.uni-bonn.de