NEARPT3 — Nearest Point Query in E3 with a Uniform Grid

W. Randolph Franklin

Rensselaer Polytechnic Institute

geom@wrfranklin.org

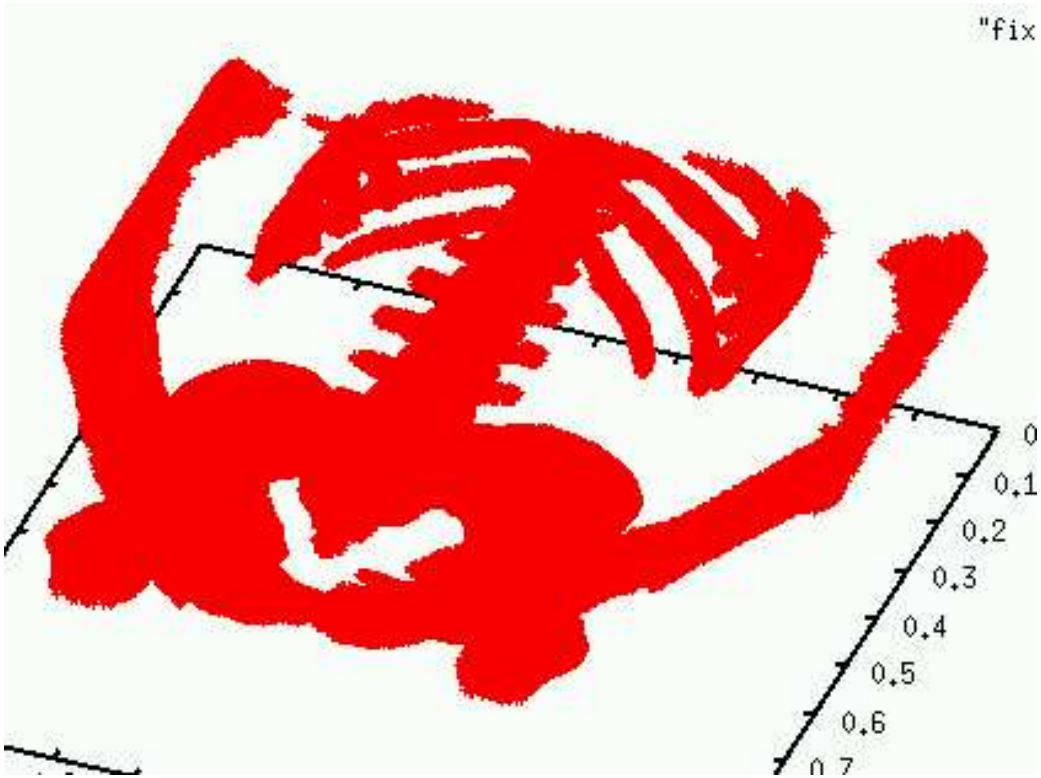
http://wrfranklin.org/

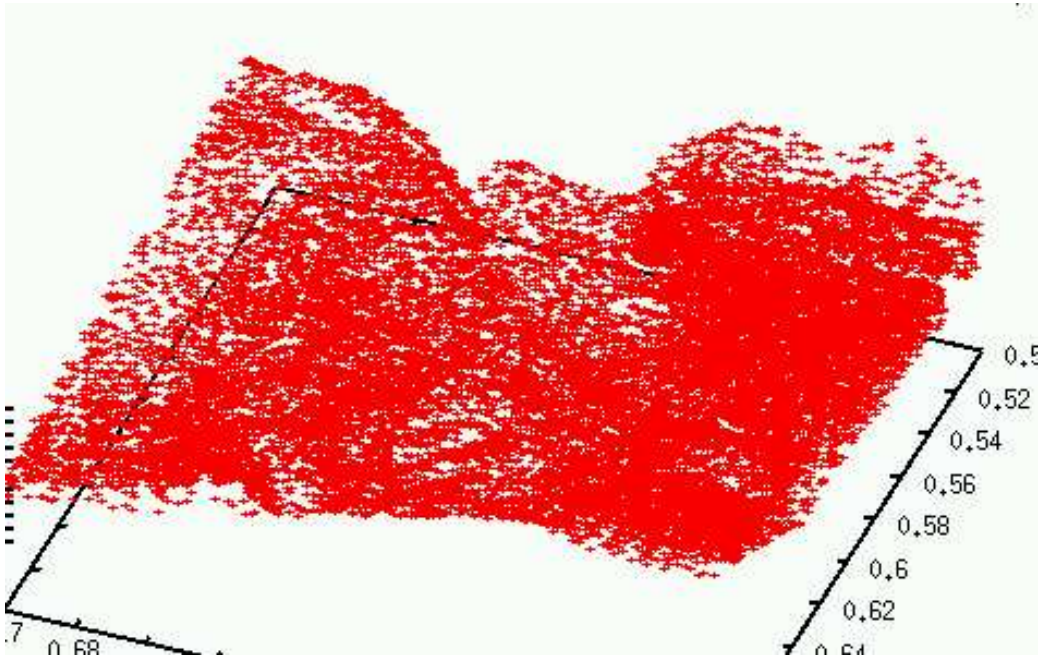
November 20, 2004

—What?—

- Consider N fixed points, p, drawn from some distribution, $\mathcal{D}$, in E^3.
- Preprocess them.
- Select query points, q, also from $\mathcal{D}$.
- For each q find the closest p.
- Optimize for $N \approx 10^7$.
- Optimize for real examples from, e.g., Georgia Tech Large Geometric Models Archive.
- Bone6 example: $10^{5.45}$ each fixed points and queries. Total preprocessing and query time: 28 seconds on this laptop.

—Bone6—





Note the nonuniformity.

—Prior Art—

Data structures:

- **Voronoi diagram.** $T_p = \Omega(N \log N)$ **to** $O(N^2)$ **in time and space.** $T_q = \theta(\log N)$.
- **Range tree.** $T_p = \theta(N \log N)$. $T_q = \theta(\log N)$.

Implementations:

- **Approximate Nearest Neighbors (ANN).**
- **CGAL.**

—Broader Implication—

- Simple data structures often work.
- Data dependent: Worst case is worse, but real data sets are quite good.
- Larger datasets will now fit in real memory
- ... and be processed much more quickly.
- External data structures are less necessary.

—General Idea—

1. Superimpose a uniform grid in E^3 on the data.
2. Insert the fixed points.
3. Locate the queries and spiral out.

—The Three Stages of the Computation—

Antepreprocessing w/o data. Compute-bound work that is independent of the data.

Preprocessing of the fixed points.

Querying to find closest fixed point to each query.

—Antepreprocessing (w/o data)—

1. Sort the cells of a grid in E^3 by distance of the closest corner from $\mathcal{O}$.
2. For each cell, c, find its *stop cell*, the last cell in the list whose closest point is closer than the *farthest* point of c.
3. Why factor out this step? It's surprisingly slow.

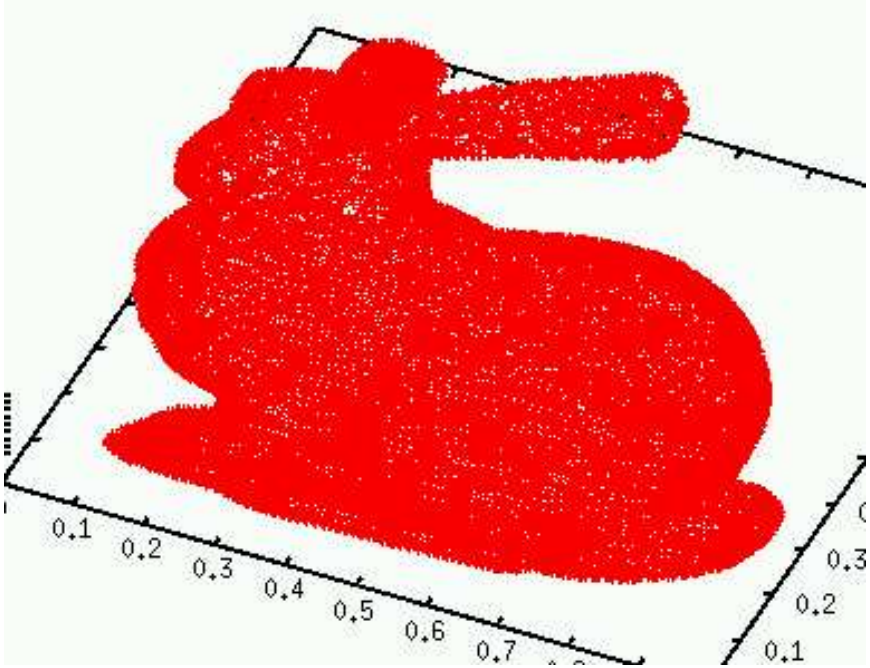
—Preprocessing (of the fixed points)—

1. Choose $\mathcal{G}$, the grid resolution.
2. Filter the points, counting the number of points in each cell.
3. Allocate a ragged array to store the points in the cells.
4. Read the points again, inserting into the cells.

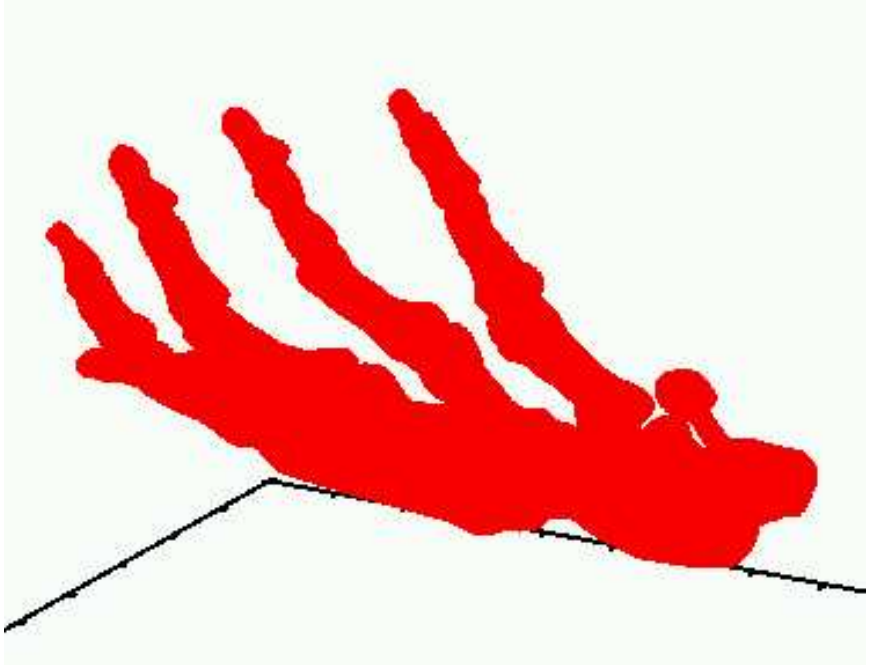
—Querying—

1. Locate the query point.
2. Spiral out, following the antepreprocessed cell list.

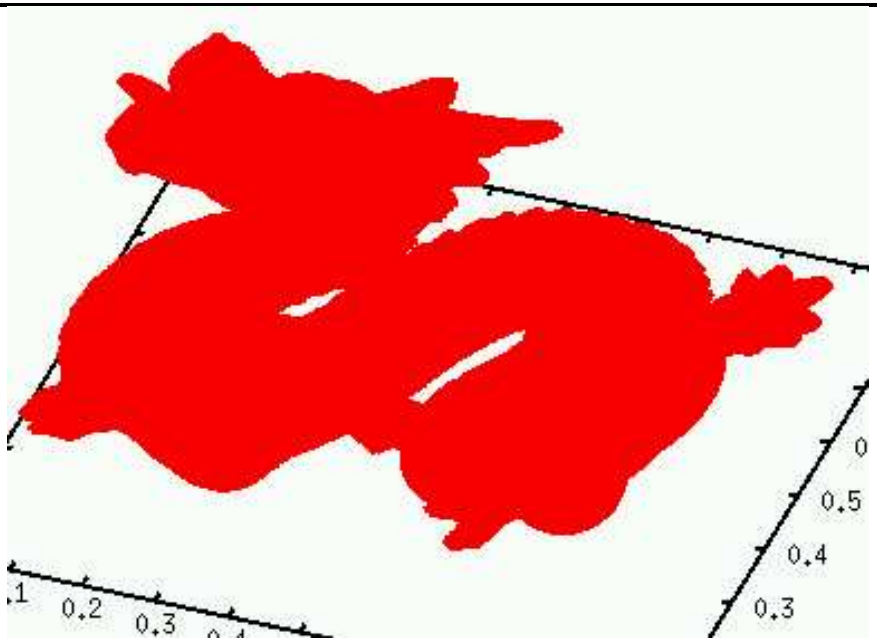
—Next Slide: Other Test Data—



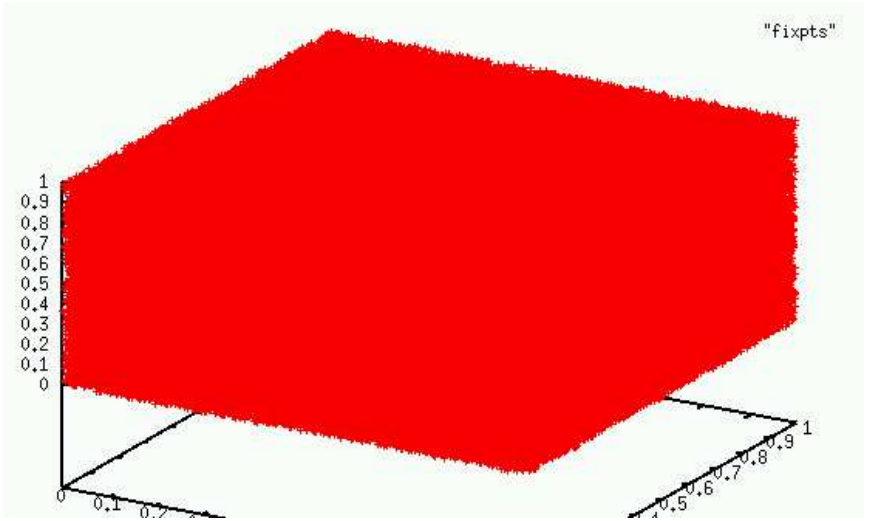
Bunny



Hand



Dragon



Uniform

—Results—

Data set	Source	# fixed points	# queries	CPU time, secs
Bunny	GIT	17973	17974	1.9
Hand	GIT	163661	163662	16
Dragon	GIT	218882	218883	21
Bone6	GIT	284818	284818	28
Uniform random	generated	10^6	10^6	128

—Extensions & Restrictions—

1. Approximate nearest neighbors obvious.
2. Nearest neighbors in E^2.

—Restriction to E^2—

1. **Compare to** APPROXIMATE NEAREST NEIGHBORS **(ANN) and to CGAL 3.0.1's** NEAREST_NEIGHBOR_SEARCHING**.**
2. **No I/O here. (Randomly generated input. Output discarded.)**
3. **Compiler choice affects speed somewhat, but not space.**

# Fixed, # Queries	Program	Time	Storage
10^6	NEARPT2	**9.4**	**46MB**
	CGAL NNS	**41**	**120MB**
	ANN	**41**	**128MB**
10^7	NEARPT2	**98**	**458MB**
	CGAL NNS	—	—
	ANN	—	—

—Future—

1. Reduce query time.
2. Process UNC Complete Powerplant (about 3M unique vertices).
3. Larger datasets anyone?