Seed Finding as a Spatial Search Problem

Using *k*-d trees to accelerate seed finding for high pile-up

Stephen Nicholas Swatman

Monday, September 26, 2022

A denotational view of seeding

Seeding is a function $f : [\mathbb{R}^3] \rightarrow [(\mathbb{R}^3)^3]$ which takes a list of points, and returns a list of triplets of those points.

$$f(S) = \{(b, m, t) \in S^3 \mid p(b, m), p(m, t), q(b, m, t)\} \tag{1}$$

Operationally, this corresponds to a triple loop, $\Theta(|S|^3)$.

A denotational view of seeding

Alternatively, focus on the middle point to match the operational semantics of the current ACTS approach:

$$f(S) = \bigcup_{m \in S} \{(b, m, t) \mid b, t \in S, p(b, m), p(m, t), q(b, m, t)\} \tag{2}$$

A denotational view of seeding

In orthodox seeding, bin spacepoints s.t. if a and b are in adjacent bins, then $p(a, b)$ and $p(b, a)$. However:

- Binning is pessimistic
- Binning is static
- Binning is irregular

In this talk: an *alternative* seeding approach based on range search.

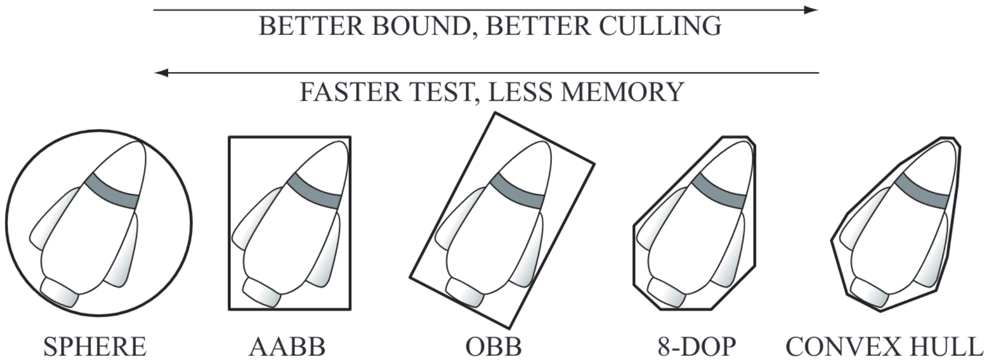
A denotational view of seeding

Instead of considering all combinations of points, we can some oracle volumes that contain all valid candidates:

$$f(S) = \bigcup_{m \in S} \{(b, m, t) \mid b, t \in S, b \in V_m^{\downarrow}, t \in V_m^{\uparrow}, q(b, m, t)\} \tag{3}$$

Does this really help us? Determining whether a point lies in an arbitrary volume is no easier than determining whether a set of predicates is true!

The hierachy of volumes



Source: Mei, Gang. 2014 "RealModel: A System for Modeling and Visualizing Sedimentary Rocks."

Spatial search problems

A very simply defined volumes is the axis-aligned (*orthogonal*) hyperrectangle! Can be defined in $\mathbb{R}^n$ as:

$$V = \bigtimes_{i=1}^{n} [v_n^{\min}, v_n^{\max}] \tag{4}$$

With extremely simple membership checking:

$$\vec{p} \in V \iff \bigwedge_{i=1}^{n} \pi_n(\vec{p}) \in [v_n^{\min}, v_n^{\max}] \tag{5}$$

Predicate conversion

Solution will need to be to derive an axis-aligned superset of V analytically. Consider that the duplet predicate $p(\vec{x}, \vec{y})$ is actually a conjunction of several smaller predicates:

$$\begin{aligned} p(\vec{x}, \vec{y}) = y_r \leqslant x_r + \Delta r_{\mathrm{max}} \\ \wedge\ z_{\mathrm{min}} \leqslant \mathrm{intercept}_z(\overleftrightarrow{xy})_z \leqslant z_{\mathrm{max}} \\ \wedge\ \eta_{\mathrm{min}} \leqslant \eta(\overleftrightarrow{xy}) \leqslant \eta_{\mathrm{max}} \\ \wedge\ y_\phi \leqslant x_\phi + \Delta\phi_{\mathrm{max}} \end{aligned} \tag{6}$$

Predicate conversion

If we can convert each of these predicates to one or more predicates of the form:

$$p_i(\vec{x},\vec{y}) \Rightarrow \left(p'_i(\vec{x},\vec{y}) = g_{\min}(x) \leqslant \pi_i(y) \leqslant g_{\max}(x)\right) \tag{7}$$

Then this trivially defines an axis-aligned bounding box...

$$g_{\min}(x) \leqslant \pi_i(y) \leqslant g_{\max}(x) \iff \pi_i(y) \in [g_{\min}(x), g_{\max}(x)] \tag{8}$$

Predicate conversion

Predicates are ideally equivalent, but weakened predicates are also acceptable! Just need to filter the spacepoints after with the corresponding equivalent predicate.

If we can find an equivalent predicate p' to our predicate p:

$$\forall x : p(x) \iff p'(x) \tag{9}$$

But if p' is a weaker version of p:

$$\forall x : p(x) \implies p'(x) \tag{10}$$

r limits

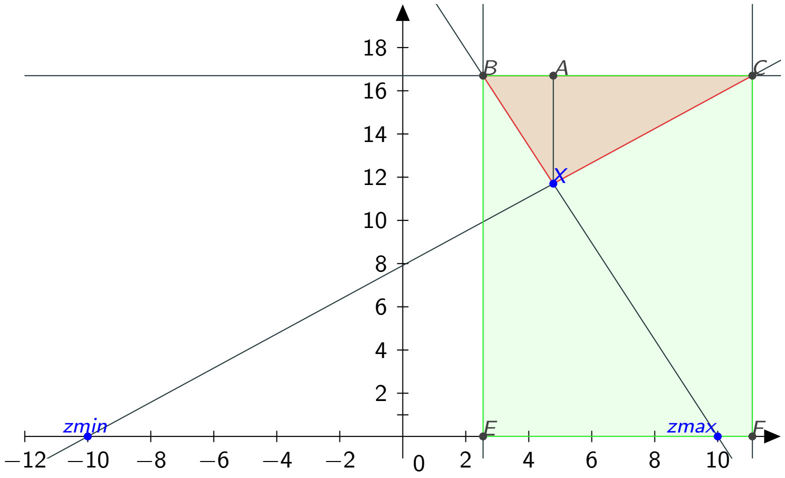
First predicate is practically in orthogonal form already:

$$p_1(\vec{x}, \vec{y}) = y_r \leqslant x_r + \Delta r_{\max} \iff x_r \leqslant y_r \leqslant x_r + \Delta r_{\max} \tag{11}$$

Thus we can conclude:

$$\begin{aligned} g_{\min}(\vec{x}) &= x_r \\ g_{\max}(\vec{x}) &= x_r + \Delta r_{\max} \\ i &= r \end{aligned} \tag{12}$$

Approximating the z-intercept



Approximating the z-intercept

This gives the following weakened version of predicate p_2:

$$p'_2(\vec{x}, \vec{y}) = z_{\max} - \frac{r_{\max}(z_{\max} - \vec{x}_z)}{\vec{x}_r} \leqslant \vec{y}_z \leqslant z_{\min} + \frac{r_{\max}(\vec{x}_z - z_{\min})}{\vec{x}_r} \tag{13}$$

Thus:

$$\begin{aligned} g_{\min}(\vec{x}) &= z_{\max} - \frac{r_{\max}(z_{\max} - x_z)}{x_r} \\ g_{\max}(\vec{x}) &= z_{\min} + \frac{r_{\max}(x_z - z_{\min})}{x_r} \\ i &= z \end{aligned} \tag{14}$$

Dependencies between predicates

If you look closely, the previous predicate relies on some r_{max} value: if one predicate constricts the r range, then it will by proxy constrict the z range!

Remaining predicates are outside of the scope of this talk... But good approximations found!

Dynamic cuts

So far, we have seen *static* cuts, which use very little information about the bound spacepoints, but this doesn't need to be the case:

$$\Delta r_{\mathrm{max}}(\vec{x}) = \begin{cases} 40, & \text{if } x_r \leqslant 60 \\ 70, & \text{if } 60 < x_r \leqslant 220 \\ 100, & \text{otherwise} \end{cases} \tag{15}$$

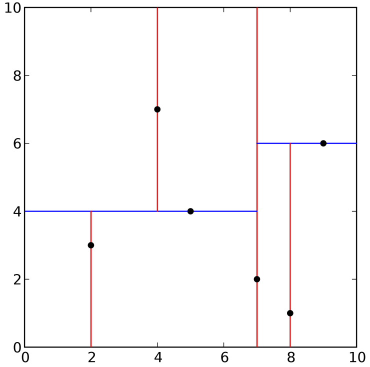
This sort of dynamic search is hard to model in orthodox seeding, but trivial to model in orthogonal seeding.

k-d trees

Use *k*-d trees from the field of computer graphics: *k*-dimensional generalisation of a binary search tree.

Each inner node picks an arbitrary dimension and splits it as evenly as possible, then recurse.

k-d trees



Source: Wikipedia (CC BY-SA 3.0)

k-d trees

Constructing a k-d tree is an $\mathcal{O}(n \log n)$ operation.

Performing a range search has worst case complexity $\mathcal{O}(n^{1-\frac{1}{k}} + n)$, for $k = 3$ that means $\mathcal{O}(n^{\frac{2}{3}} + n)$.

Average time to perform a range search is closer to $\mathcal{O}(\log n)$.

k-d trees

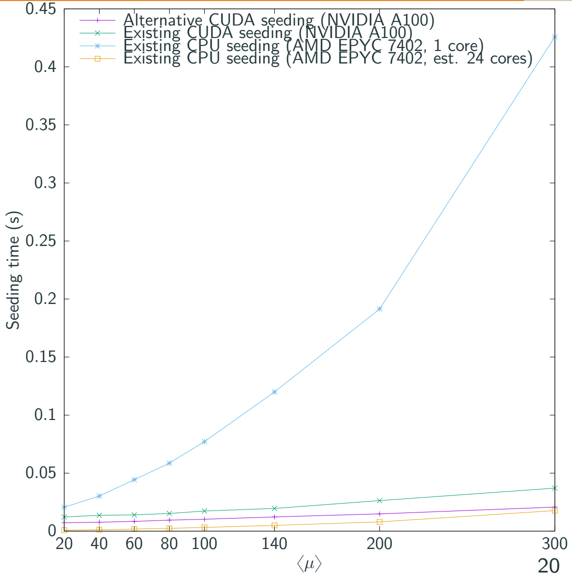
Added an implementation of a generalized *k*-d tree to ACTS, with a few design considerations:

- Dynamic median finding: Use exact median for small data sets, approximate a median for larger data sets.
- Higher order functional interface: Use higher order functions to perform operations on the tree, allowing us to remove the overhead of constructing output vectors, and write cleaner code.

Very naive simplified GPU implementation in *traccc*.

Thinking of co-processors

k-d trees are GPU-friendly, and naive implementation of this seeding exists in *traccc*.



Conclusions and future work

Seed finding can be reduced to a spatial search problem. Provides alternative to orthodox seeding with some benefits:

- Dynamic search ranges based on e.g. detector region
- Regular GPU-friendly structure
- More accurate search spaces

Currently implemented in main-line *Acts* (CPU) and *traccc* (GPU).

Questions or comments outside this session: stephen.nicholas.swatman@cern.ch!