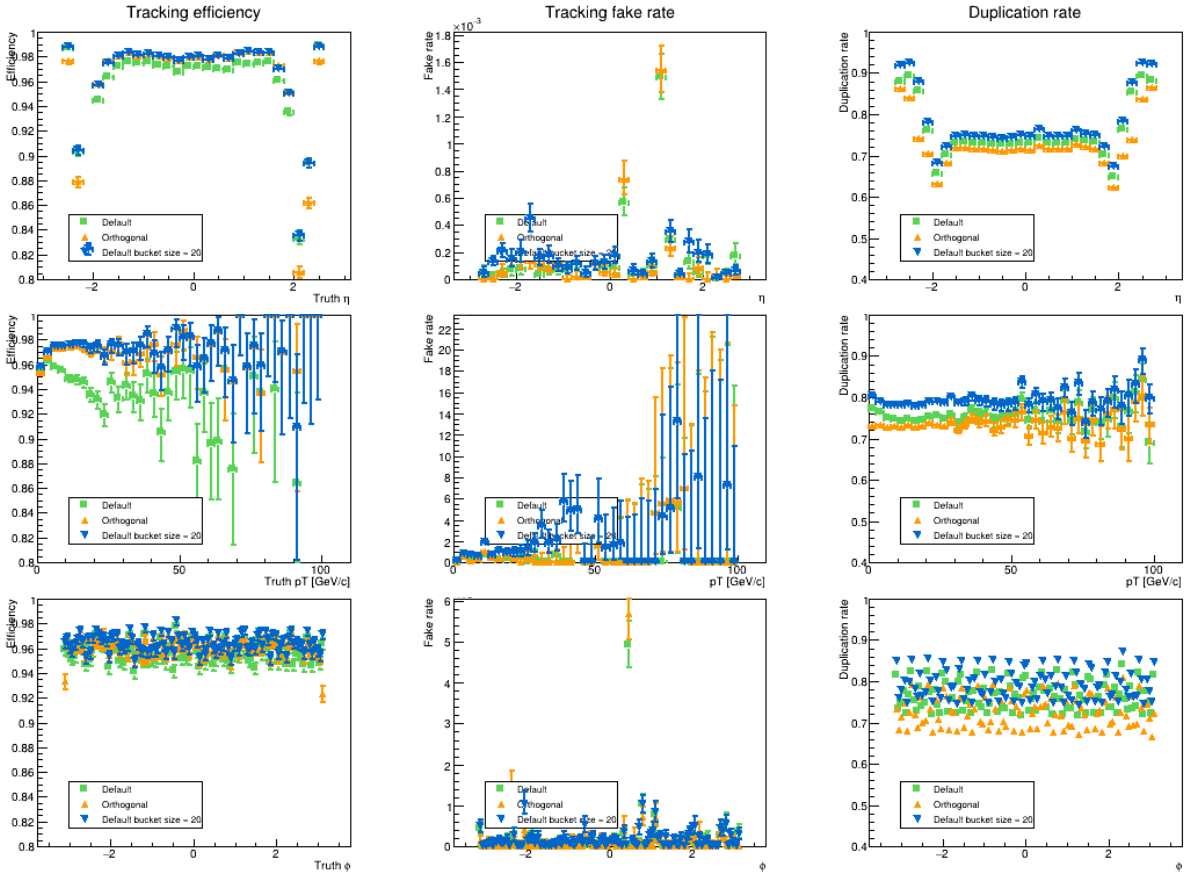
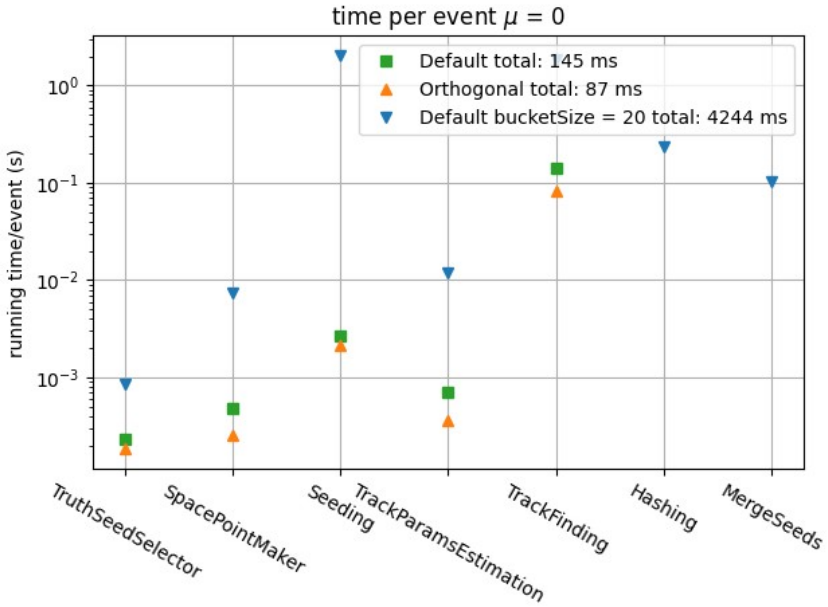
Tracking with Hashing

Current state

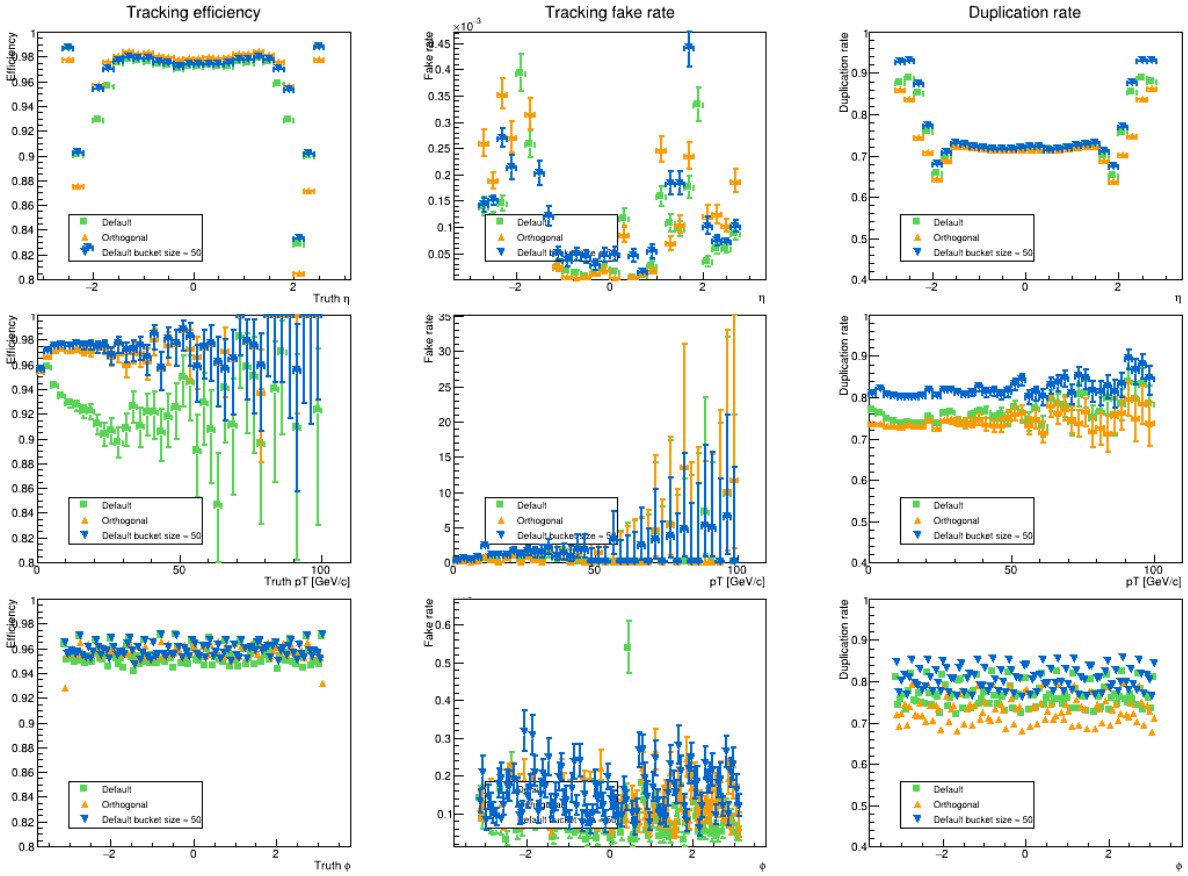
Jeremy Couthures

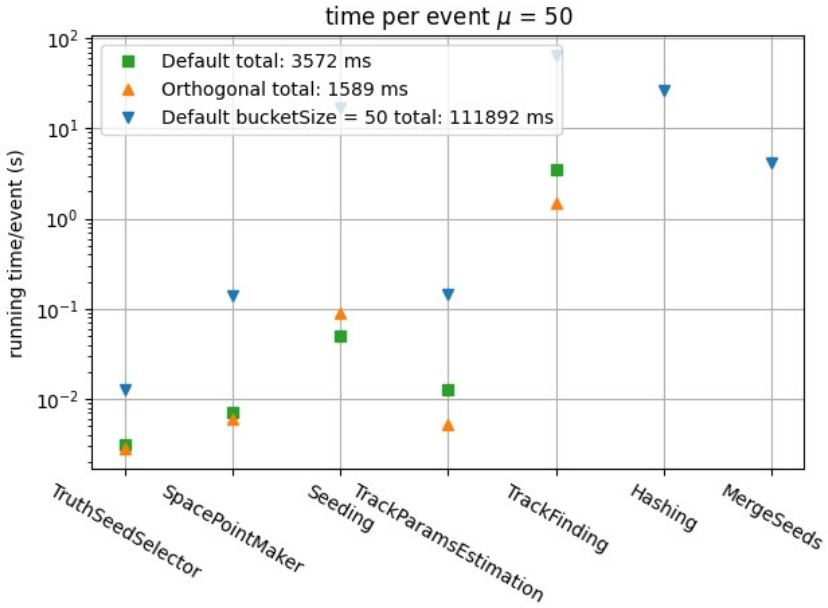
Mu = 0 TrackML


Tracking efficiency
Tracking fake rate
Duplication rate
Efficiency
Fake rate
Duplication rate
Default
Orthogonal
Default bucket size = 20
Truth η
η
Truth pT [GeV/c]
pT [GeV/c]
Truth ϕ
ϕ


time per event μ = 0
running time/event (s)
Default total: 145 ms
Orthogonal total: 87 ms
Default bucketSize = 20 total: 4244 ms
TruthSeedSelector
SpacePointMaker
Seeding
TrackParamsEstimation
TrackFinding
Hashing
MergeSeeds

Mu = 50 TrackML





Orthogonal Seeding

stephenswat commented on Jul 30, 2021 • edited ▾ Member ⋯

As I said in #901, I have been playing around with seed finding a little bit lately. Last weekend, I mentioned an idea for a new (?) kind of seed finding algorithm based on range search datastructures, and this is the very, very first semi-working implementation of it, just before the weekend.

The idea behind this algorithm is relatively simple. In traditional seedfinding, we check a whole lot of candidate spacepoints to see whether they meet some condition. If you look at this differently, each spacepoint defines a volume in the z-r-φ space, which contains any spacepoints it can form a doublet with. What if we reversed this logic? What if we defined this volume first, and then just extract the spacepoints inside of that space? That way, we can vastly reduce the number of spacepoints we need to look at.

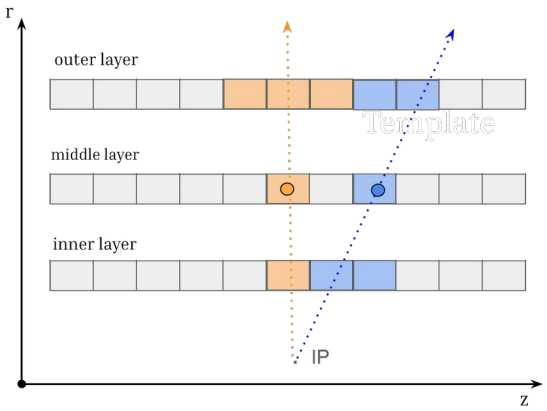
How do we do this quickly? With *k*-d trees. These data structures are cheap to build, and they give us very fast orthogonal range searches. In other words, we can very quickly look up which of our spacepoints lie within an axis-aligned orthognal n-dimensional hyperrectangle. In this case, which spacepoints lie within a z-r-φ box.

So, the core idea of this seedfinder is to define as many of our seedfinding constraints in orthogonal fashion. That way, we can make our candidate hyperrectangle smaller and smaller. The tighter the constraints we can place, the better. Then, we look up the relevant spacepoints, and we can avoid looking at any others. That also means this solution requires no binning whatsoever.

Orthogonal Seeding

Default

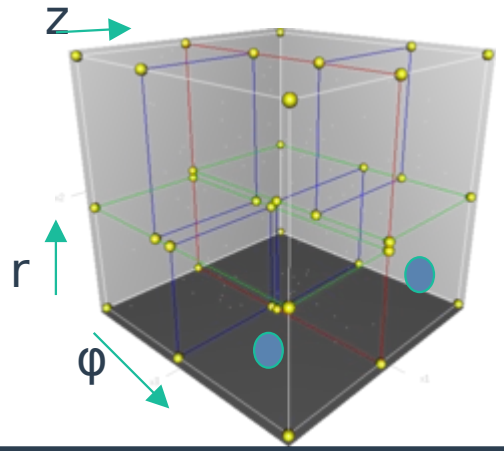
z-φ bins



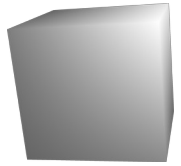
Look for lines

Orthogonal

K-d trees



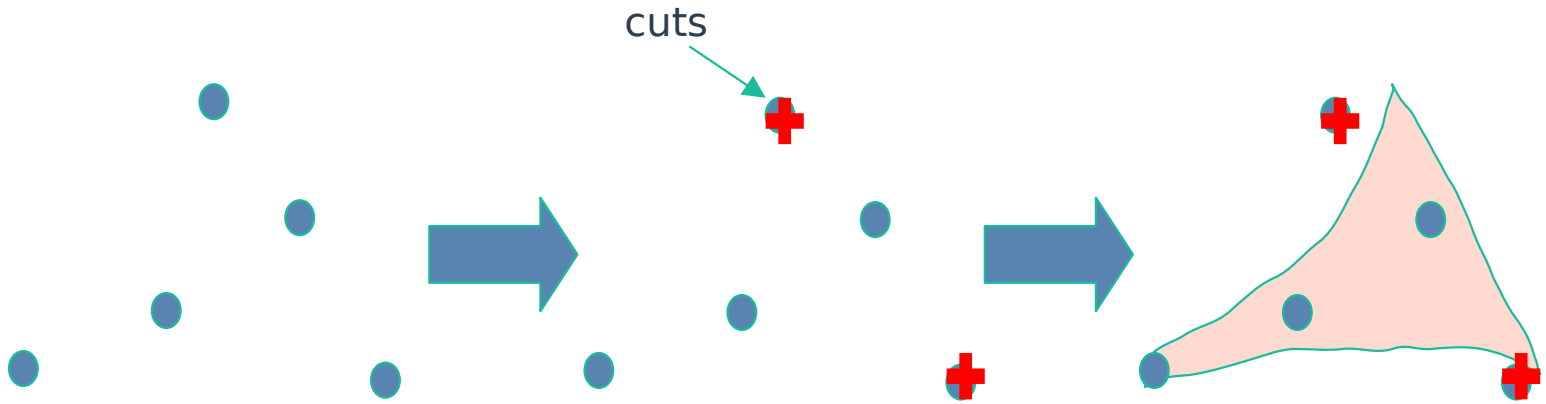
z-r-φ box



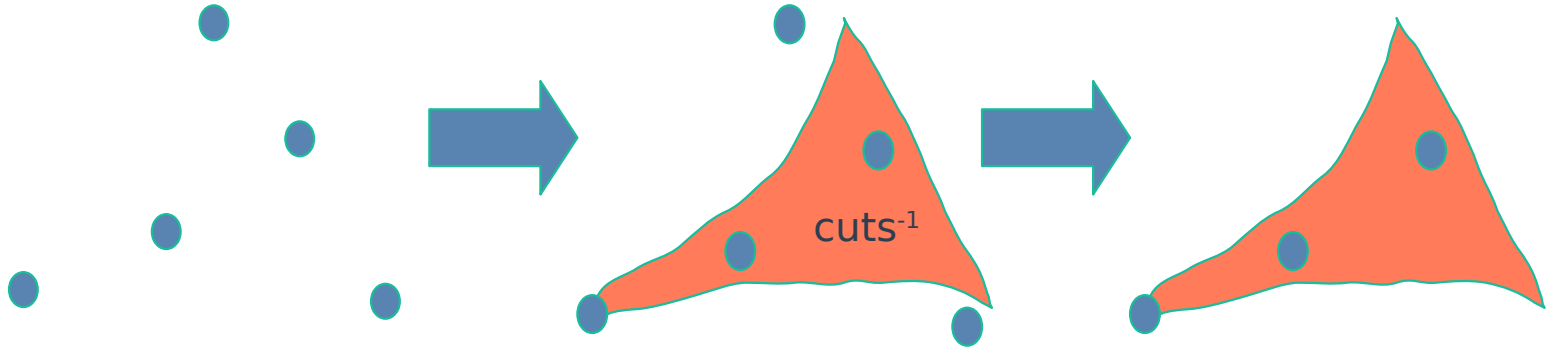
Look for volume

Orthogonal Seeding

Default

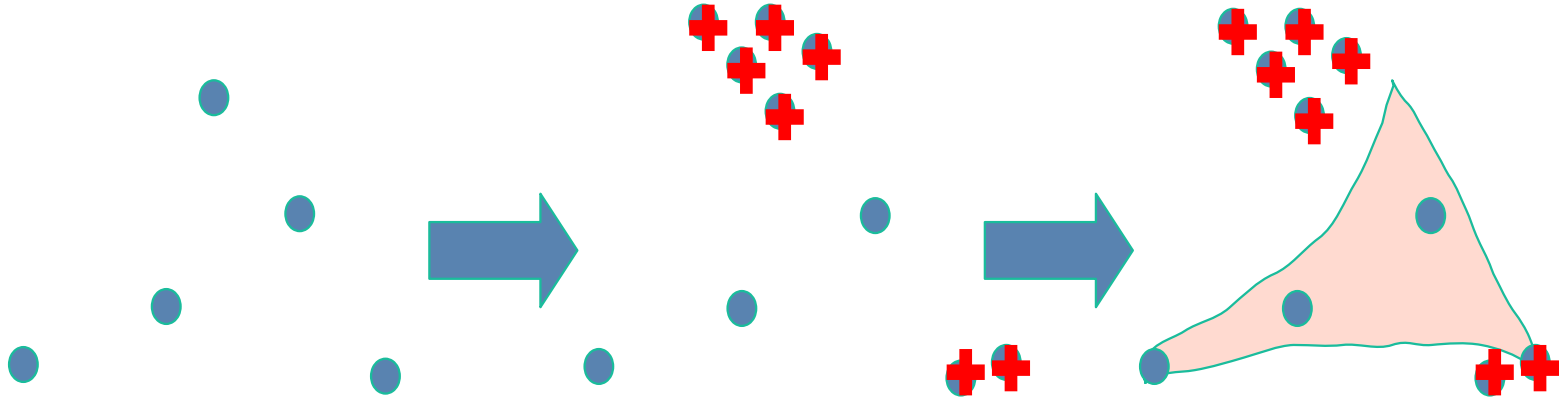


Orthogonal

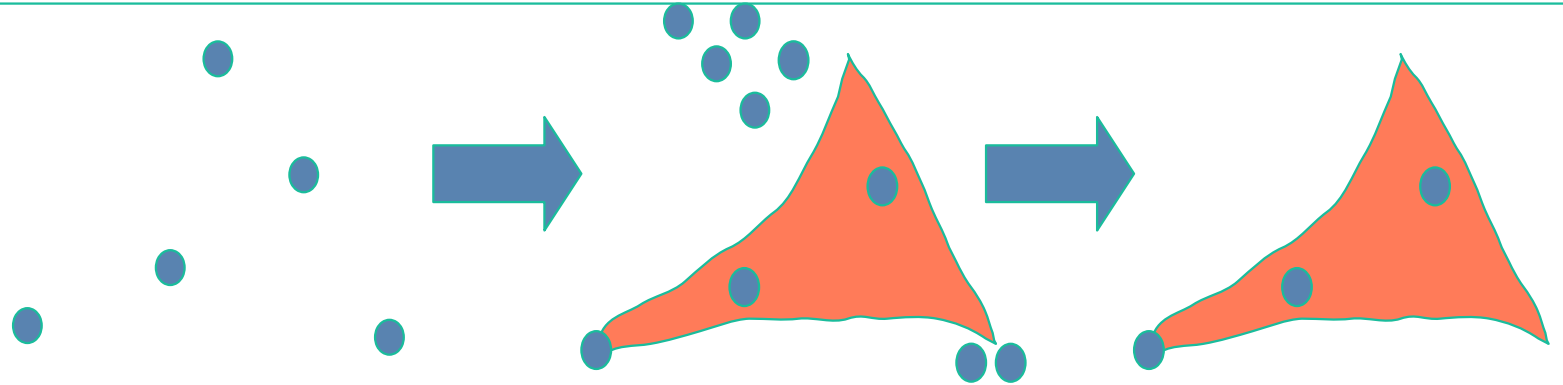


Orthogonal Seeding

Default



Orthogonal



Conclusion

- **Efficacités hashing/orthogonal plus élevées que le Default ; hashing, plus de duplicates**
- **Hashing prend beaucoup plus de temps ~ x30**

  Orthogonal prend ~2x moins de temps que Default

  La suite ?
- **Mu = 100**
- **Bucket intelligent (moins de buckets à passer au seeding)**