Progress adding 3D imaging to the Wire-Cell Toolkit

Process

- Follow Xin's WCP imaging summary slides
- Follow Xin's high level "imaging flow" →
- Stare at WCP code, but no copy-paste allowed! ☺
- Add some new ideas and optimizations.
- Development steps:
  1. implement basic operations as "util" code.
  2. Design WCT data and component interfaces.
  3. Iterate as realism is added (ie, test on full sim/data, dead wire support).

Core, high level operations

- Slicing (done)
- Striping (done, but in the end, not needed)
- Tiling (done)
- Clustering (done)
- Solving (in progress)

“done”, here, means code works on idealized tests.

I’ll next go through each one.

Slicing

*Cut up a readout **frame** of signals into a queue of time **slices** with each slice holding a set of **active channels** and the sum of their waveform sample values in the time **span** of the slice.*

- A **slice** data object has:

ident	some identifying **number**
start	when in **time** the slice begins.
span	the **duration** of the time interval covered.
activity	a **map** from channel to a value (signal charge)
frame	a back-pointer to the original **frame**.

- Algorithm: the obvious thing.

Striping

> *Collect all **channels** in a **slice** with a **value** above a **threshold** along with their **connected wires** such that the wires are **contiguous** in their plane.*

- In DUNE a stripe may "wrap" around the APA, both in terms of the wire conductors and the list of channels.
- In the end, this wasn't needed for **tiling** (next).
- However, it has a "cute" solution:

1. Build a **graph** with each **channel** above threshold connected to its 1, 2 or 3 **wires** and in that set connect each wire to its **neighbor** in the plane.
2. Call `boost::connected_components()`, done!

The returned value is effectively the list of "stripes" for the slice.

Tiling

*Tiling identifies **spatial regions in a slice** which are **likely consistent** with containing **ionization activity** based on knowing **wire geometry** and **channels above threshold**.*

- Although, more accurately we **remove** regions which are **definitely not consistent** with....
- Tiling is the core starting point for 3D imaging, and inherently very **combinatoric** so optimizations are very welcome.
- And a nice one has been found....

Ray Grid Optimization

> *Exploit* ***uniform wire angle and pitch*** *to make wire geometry calculations cheap.*

The concepts are actually rather simple but they need some detailed slides....

Ray Grid

Ray: (as used here) is a **line segment** defined by **two 3D endpoints** and a **direction**.

- Each real wire (segment) has an associated ray.
- Define two parallel rays, **ray0** and **ray1** offset by $\pm\frac{1}{2}$ pitch on each side of **wire 0** for each wire plane.
- This ray pair may define a 2D coordinate system:
  - ○ origin: center of **ray0** (projected to $x = 0$)
  - $\rightarrow$ $\hat{z}$ axis along $\overrightarrow{pitch}$
  - $\uparrow$ $\hat{y}$ axis along $\overrightarrow{ray}$
  - $\otimes$ $\hat{x}$ by RHR (anti-drift direction)

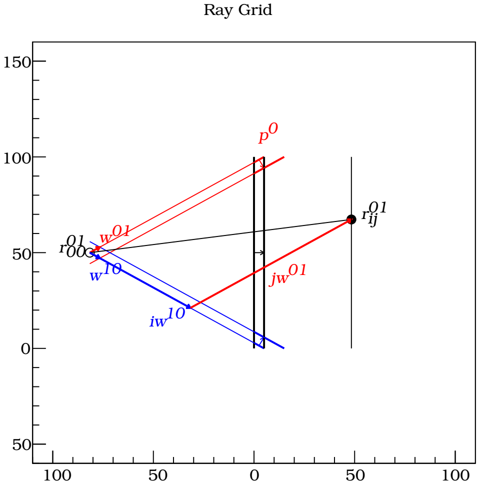
$\leftrightarrow$ Also defines a **1-D linear grid** along plane pitch direction.

$\Rightarrow$ Specifying a **ray index** specifies a **pitch location**.

$\Rightarrow$ Valid **ray indices** $\in [0 ... N_{wires}]$ (inclusive and for each plane)

Thus, each ~~wire plane~~ **layer** has a **ray grid**.
I'll say why I change names to "layer" in a bit.

3-layer example of Ray Grid vectors



ray0/ray1 pairs for **U**,**V**,**W** planes. Other vectors described next.

Two Ray Grids

> *Combining two 1-D ray grids form a* ***regular 2D grid****,*
> *may provide a* ***non-orthogonal coordinate system****.*

Each ray grid is one "layer".

- p^l relative **pitch vector** for layer l.
- c^l the **origin point** (center of **ray0**) for layer l.
- r_{ij}^{lm} a **crossing point** of ray i from layer l and ray j from layer m.
- w^{lm} relative **displacement vector** for layer l connecting crossing points of neighboring layer-m rays on a ray of layer l.

(revisit diagram on previous slide)

The Key Optimization

Given vectors p^l, c^l and tensor w^{lm} for **two layers** and one explicitly calculated crossing point r^{lm}_{00} the tensor of **all other crossing points** r^{lm}_{ij} is trivial:

$$r^{lm}_{ij} = r^{lm}_{00} + jw^{lm} + iw^{ml}$$

- r^{lm}_{00} and w^{lm} can be calculated with simple vector arithmetic
- **For N layers**: must calculate for **every pair of layers**.
  - This is $\mathcal{O}(\frac{1}{2}N^2)$ but $N = 5$ so we don't care.
    - ? (why 5 and not 3? it's coming!)

2-Layer crossing points in a 3^{rd} layer

Core tiling operation: given a ***crossing point*** r_{ij}^{lm} *what is its* ***pitch location*** *in a* 3^{rd}*, layer* $n \notin \{l, m\}$.

The tensor P of all such pitch locations:

$$P_{ij}^{lmn} = (r_{ij}^{lm} - c^n) \cdot \hat{p}^n$$

Where $\hat{p}^n$ is unit vector in pitch direction of layer n.
Expanding r_{ij}^{lm} from last slide,

$$P_{ij}^{lmn} = r_{00}^{lm} \cdot \hat{p}^n + jw^{lm} \cdot \hat{p}^n + iw^{ml} \cdot \hat{p}^n - c^n \cdot \hat{p}^n$$

Or more simply,

$$P_{ij}^{lmn} = ja^{lmn} + ia^{mln} + b^{lmn}, \ (l \neq m \neq n)$$

The tensors a and b are scalar valued, a is not symmetric under a transpose of l and m and b is. Both have undefined diagonals

Crossing point containment

Okay, the operation that is **really** needed is:[1]

> *What rays in layer n bound a given crossing point r_{ij}^{lm} of layers l and m?*

We simply normalize P_{ij}^{lmn} by the pitch and truncate to get the **index of the ray** which is **at or just below** (in pitch) the **crossing point**:

$$I_{ij}^{lmn} \equiv floor(P_{ij}^{lmn}/p^n)$$

$$(l \neq m \neq n)$$

[1] In WCP language: "is a *blob corner* in a *merged wire*?"

Ray Grid Optimization Summary

We can now **very quickly** find:

- any crossing point of two rays (in different ray grids),
- the pitch location in a 3^{rd} layer of this crossing point,
- the index of the nearby ray and thus
- the corresponding wire and the corresponding channel.

These are the building blocks. Now, on to tiling....

Tiling prelude: Activity and why 5 layers

> ***Activity array****: a 1-D array, defined on a* ***ray grid****, with elements indicating possible "activity" somewhere near the ray*[2]*.*

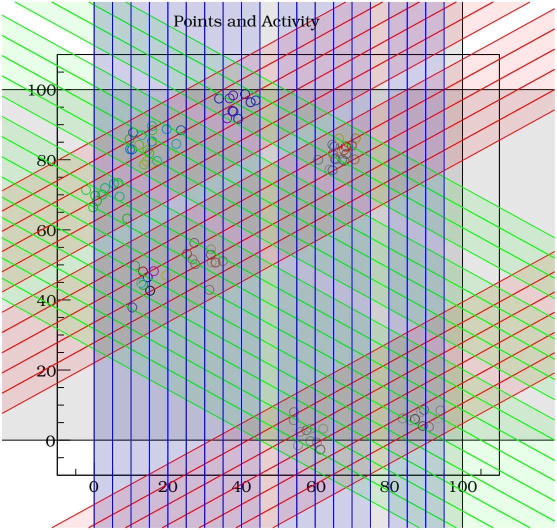
- Gives $\mathcal{O}(1)$ lookup of a wire/channel given a pitch.
- For each plane, U, V and W the **activity array** is simply the channel **charge values** in the time slice **ordered/limited** by the **wires in the plane**.
- Add to these, **two special layers**, each of a single "wire" and "channel" which are **always active** and with a "pitch" equal to the height or width, respectively, of the sensitive area of the anode.

Tiling thus solves a $2 + N$ layer problem where $N = 3$ for pD/DUNE/MB but can be naturally extended to $N = 4+$.

[2] Each element is a "fired wire" in WCP language

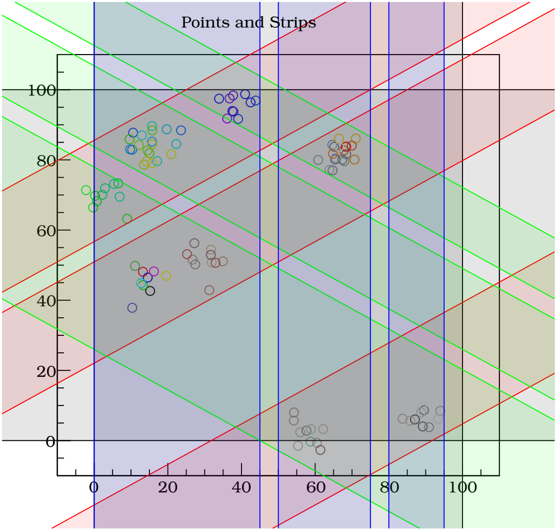
Activity arrays illustrated with toy simulation

- Make random clusters of “electrons” in one slice.
- For each, find nearest wire from each plane and assign it +1 hit.
- Channels sum all hits on attached wires.
- Non-zero activity array elements colored by their plane. (threshold: $n_{hit} > 0$)
- The 2 special horiz/vert boundary layers are in gray.



Activity array to Strips

- Scan each activity array to find contiguous regions above threshold.
- Collect set $\{s^l_{ii'}\}$ of strips in layer l bound by ray i and ray i'.



Strips to Blobs

Blob: a contiguous region of mutual intersections of strips from all layers.

Procedure (ignoring some optimizations):

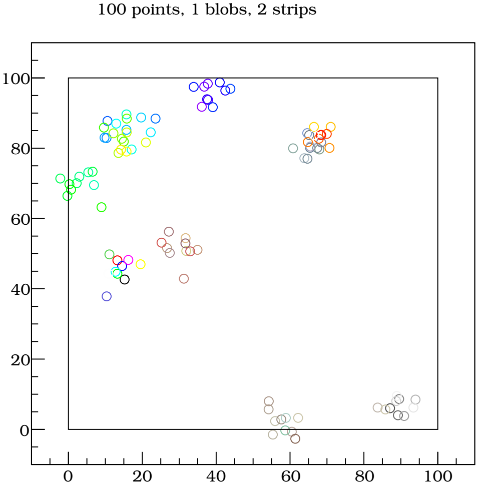
1. Given layers 0, 1: find set of all crossing points $\{r_{ij}^{01}\}$ of the **strip boundary rays** from $\{S_{ii'}^{l}\},\ l \in 0, 1$.
2. Add layer 2: find set $\{r_{ij}^{l2}, r_{ij}^{2l}\}$ for $l \in 0, 1$
3. Discard any crossing point $\{r_{ij}^{lm}\}$ which is not in a strip of any other layer $n \notin \{l, m\}$.
4. `goto` 2 for layers 3, 4, 5, ...

- This is the **inherently combinatoric** process.
    - good thing ray grid optimization is so fast!

Next, I walk through an example applying each layer to the toy simulation....

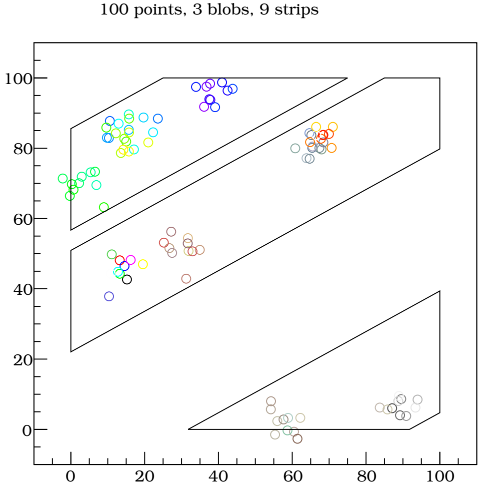
Layers 0, 1

- L0+L1 is trivial.
- A single blob results which spans the overlap of the single horizontal and vertical strips and thus exactly spans the active area of the anode.
- The example detector is 100×100 distance units in size.



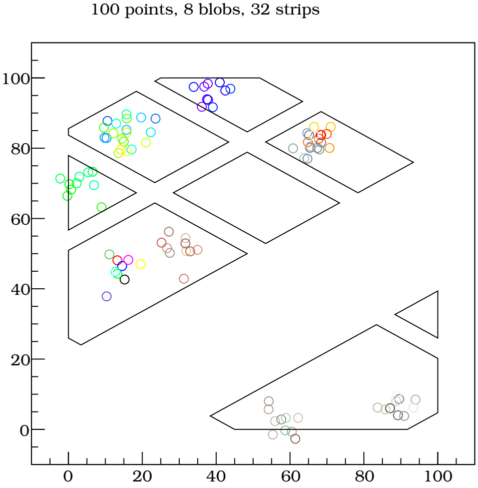
Layers 0, 1, 2

- L0+L1+L2 adds first actual wire plane
- The previous single blob is broken into three.
- Each blob is defined by a pair of boundary rays from each layer.



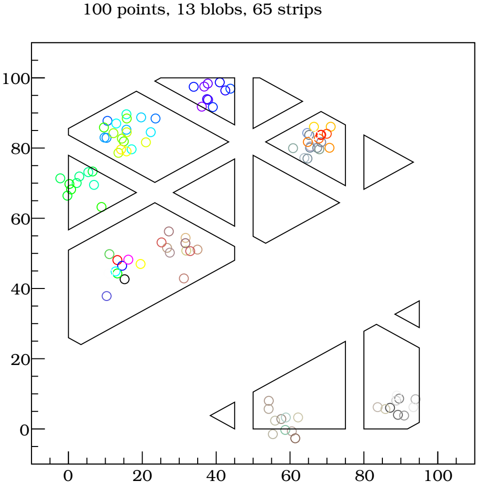
Layers 0, 1, 2, 3

- L0+L1+L2+L3 adds second wire plane
- Three blobs become eight.
- Knowing the “true electrons” we start to see ghosts.

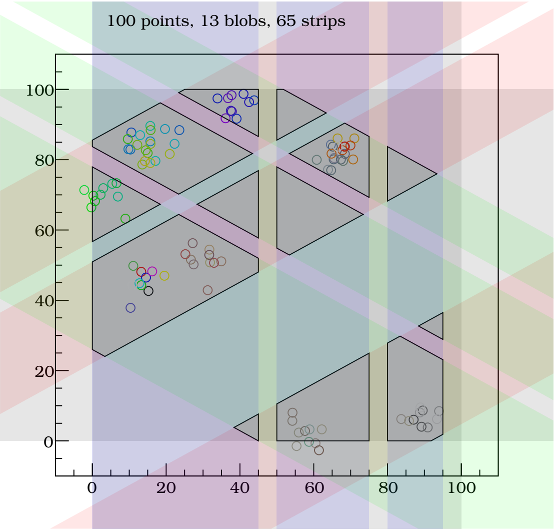


Layers 0, 1, 2, 3, 4

- L0+L1+L2+L3+L4 adds final wire plane
- Eight blobs become 13.
- Some ghosts reduced in size and increased in number.
- Some ghosts may be removed, but not in this example.



Final result + original strips



Clustering

Clustering (here) is the ***association*** *of blobs between neighboring time slices based on their relative positions.*

- Essentially an extension of tiling except compare blob corners (still ray grid crossing points) in one slice against strip bounds of blobs in another slice.
- Can be used for applying **connectivity conditions** to remove some ghosts, in “solving” or to build extended 3D clusters.
- Currently, the definition of “overlap” of blobs is hard-wired. This can be improved to allow some parameterized “overlap distance” to be used.

Clustering visitor

Clustering is a generic function which takes two lists of blobs and a **functor**:

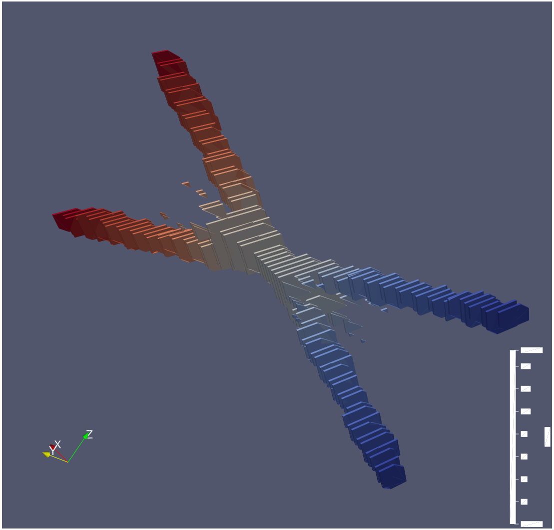
```
auto assoc = some_function_object;
RayGrid::associate(blobs_1, blobs_2, assoc);
```

The `associate()` function will call the `assoc` functor for every pair of blobs which are associated.

It's then up to the user to do something fun in their functor.

Visualization

- Support writing VTK files for ParaView and MayaVi.
- A little "hacky" but good for debugging.
- Shows blobs resolved from two line sources crossing at a point.
- Each "plate" is one blob.
- Color represents slice number.



Solving

Use measured charge information and wire geometry to invert $M = G \cdot S$.

S is charge in each blob, G is blob-channel connection, M is measured charge in channels.

- This work is just starting.
- It brings together blobs built from both APA faces because **now** wrapping adds complication.
- Maybe use connected subgraph trick (see "striping") to partition G into smaller problems?
- Maybe look for an alternative to `ress`? Or, is it already the best available?

Yet More To Do

- Complete the solving stage.
- Configure full chain test starting with trivial line sources.
- One more pass through my thinking to see how/where to handle dead channels.
- Test on real sim and real data.
- Help others to use, improve and validate.

A lot of more work, but progress is getting made!