Wire-Cell Toolkit Imaging

Brett Viren

April 19, 2019

1 Introduction and Scope

This (very informal and sloppy) note documents the Wire-Cell Toolkit (WCT) ionization electron distribution reconstruction package `wire-cell-img` aka *3D imaging*. It describes the algorithms and their implementation. First it describes basic building blocks of *rays* and how they are used to define coordinate systems called *ray grids* which allow fast queries on the wire geometry. These ray grids are then used to implement the core algorithm, from which first Wire-Cell derived its name, termed *tiling*. This restricts regions of a 2D plane in each readout time slice across channels which may contain ionization electrons. Then a channel **grouping** is performed to reduce the size of the problem of the final step called *solving*.

2 Rays Basics

In WCT, in the `WireCell` namespace, a `Ray` is simply a `std::pair<Point,Point>` defined in `wire-cell-util` in the `Point.h` header. Conceptually it represents an relative vector pointing from `Ray::first` and to `Ray::second`. Various vector operations are defined on `Point` and `Ray`. Note, one may see `Vector` used. It is a synonym for `Point` and sometimes used to semantically distinguish between a relative displacement and an absolute point.

Rays may be used in several contexts and one critical one is in describing geometry related to wires (as always, this means wire segments). The rest of this section describes how rays are used to implement the core Wire-Cell imaging algorithms. It covers a progression of functionality:

grid construction of a set of grids defined in terms of rays which allow fast *addressing* of special points in a 2D space

tiling partitioning the 2D space, based on the grid, to constrain location of activity

clustering combining elements of the 2D tiling along a third dimension

grouping combining overllaping "sources" ("blobs") and "measurements" ("merged channels")

solving determine the amount of ionization activity likely in a regions localized by tiling.

This functionality is based on bounding regions of the 2D plane with pairs of parallel rays. A region is considered to have infinite extent along the mutual direction of its bounding rays and to have finite extent along the perpendicular direction. This finite extent is represented by a **pitch** vector which has its end points on both rays. The ray on the tail of the pitch vector is considered inside the region while the other ray is considered outside. That is, the region is half-inclusive.

3 Ray Grid

The wire segments in a pair wire planes provide a basis for a non-orthogonal grid. Under the assumption that the wires share a common pitch and angle the grid is uniform. This assumption allows for optimization in calculating crossing ray crossing points in terms of indexed grid points. It further allows optimization in calculating the location of a crossing point in terms of the pitch location of a third plane.

A generalized *ray grid* as consisting of a number of coplanar *layers*. Each layer is composed of a infinite number of parallel rays of uniform pitch which may be generated by specifying an ordered pair of two neighboring rays. The *pitch vector* of this layer is generated as running perpendicular to both rays and of magnitude that of their separation. A three-layer ray grid is illustrated in figure 1.

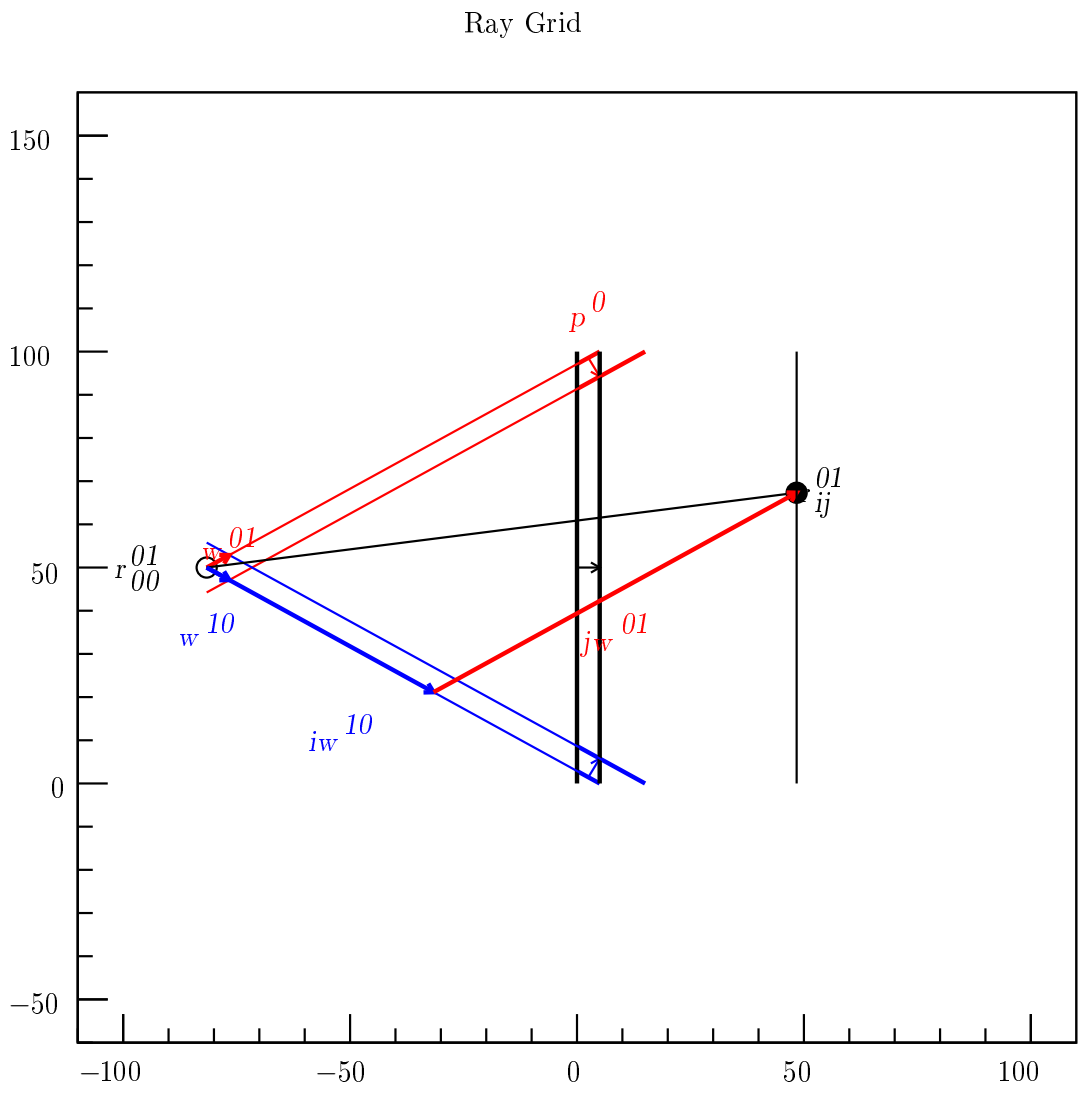


Figure 1: The points and vectors involved in constructing and using ray grids.

A number of vectors are defined. Their superscript indices label a layer and their subscripts label grid indices.

p^l the pitch vector for layer l

c^n a special center point on *ray 0* of layer n

r_{ij}^{lm} the crossing point of ray i from layer l and ray j from layer m

r_{00}^{lm} the crossing point the zero rays of layers l and m.

w^{lm} a vector giving the displacement along the direction of a ray in layer l between the crossing points of that ray and two neighboring rays from layer m

This last two tensors, r_{00}^{lm} and w^{lm} can be calculated with simple vector arithmetic (see `ray_pitch()` from `Point.h`), pair-wise among the N_l layers in the ray grid. The former is symmetric and both have undefined diagonals and so require $\mathcal{O}(N_l^2)$ operations where N_l is the number of layers (typically $N_l = 5$ as described below). Given these two tensors, arbitrary crossing points of rays from different layers can be written as

$$r_{ij}^{lm} = r_{00}^{lm} + jw^{lm} + iw^{ml} \tag{1}$$

which provides a faster calculation than the vector arithmetic required to find the crossing points of arbitrary rays.

Locating crossing points of rays from two layers in a third layer is another fundamental operation. Given that these crossing points are on the ray grid a tensor equation can be written:

$$p_{ij}^{lmn} = (r_{ij}^{lm} - c^n) \cdot \hat{p}^n \tag{2}$$

Expanding r_{ij}^{lm},

$$P_{ij}^{lmn} = r_{00}^{lm} \cdot \hat{p}^n + jw^{lm} \cdot \hat{p}^n + iw^{ml} \cdot \hat{p}^n - c^n \cdot \hat{p}^n \tag{3}$$

Given the tensor symmetries, one can express more simply this as: $P_{ij}^{lmn} = ja^{lmn} + ia^{mln} + b^{lmn}$. 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. That is, l, m, n must be mutually unique. Finally, it is common to look up some quantity that is indexed by the ray index. This can be found simply a $I_{ij}^{lmn} = floor(P_{ij}^{lmn}/p^n)$.

4 Ray Tiling

Central to Wire-Cell 3D imaging is the concept of "tiling". This is a partitioning of the 2D plane that is transverse to the electron drift direction by considering the measured signal samples from all channels summed over some slice of time (typically a few ms). The partitioning is performed along lines which are parallel to the wires of the planes on one face of an anode.

The goal of the tiling is to identify regions where ionized electrons likely existed in the time slice. The procedure involves selecting all wires attached to all channels which collected signal (after response deconvolution) above some threshold. These are called *active wires*. The ray parallel to an active wire and residing half way along the negative pitch direction to the active wires nearest neighbor is called an *active ray*. That is, an active ray provides the lower bounds of an *active strip* which is one pitch wide and centered along the active wire.

This is illustrated in figure 2. It represents a simple simulation of a time slice. Clumps of points are generated and for each point the nearest wire from each layer is located and its activity count is incremented.

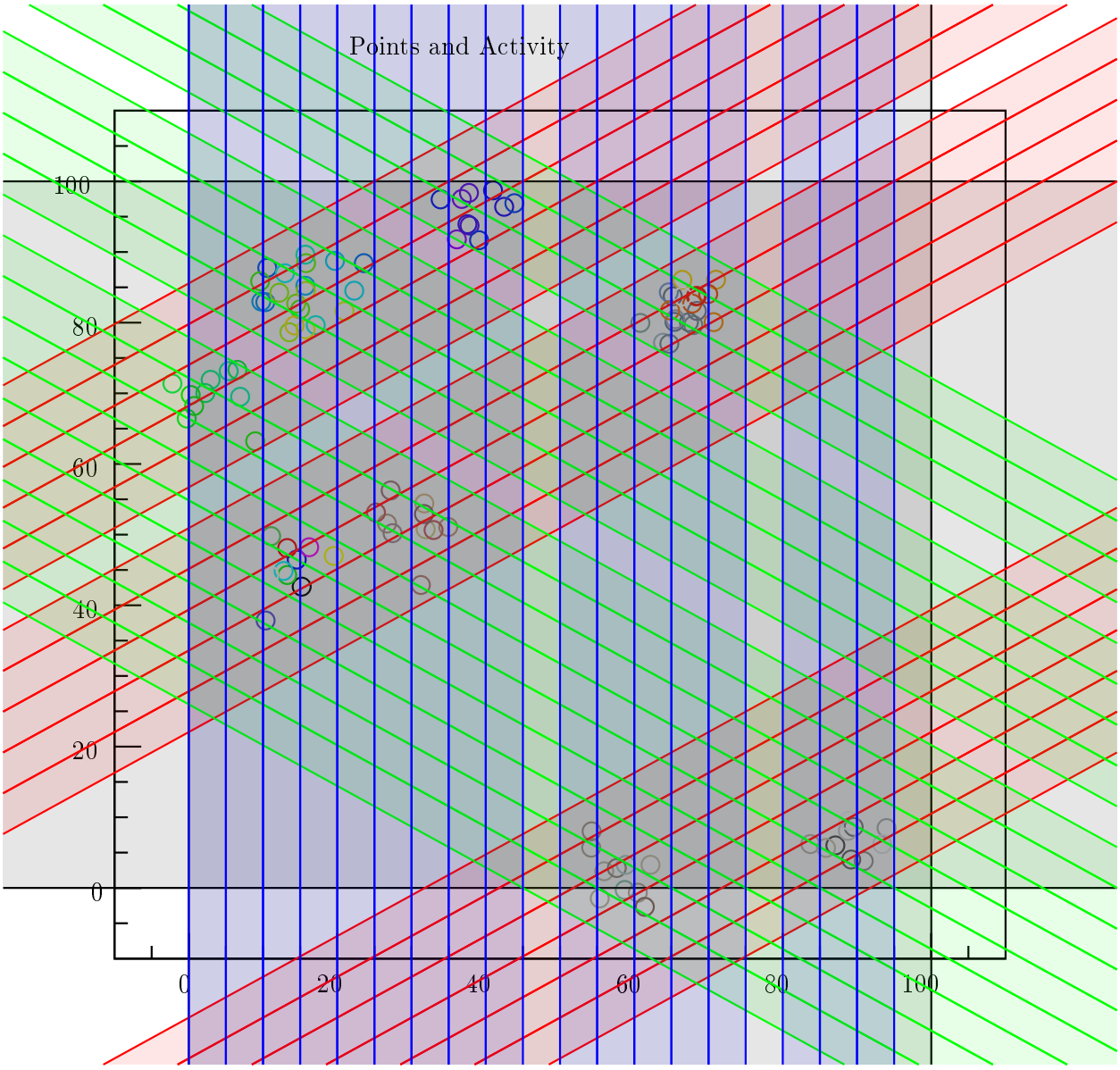


Figure 2: Depiction of the location of generated "electrons" in a toy simulation and the resulting activity arrays which colored based on the ray grid layer the span.

The figure then shows a the active wire regions for five layers. The first two layers are specially defined to provide the horizontal and vertical bounds of the detector. These layers have a single gray colored strip with a pitch that spans their respective active area (100 units in the example).

There are then three more layers of active strips colored red, green and blue and each which represents the active rays from one wire plane given the generated points represented as circles. Wires which have no activity above threshold are seen as gaps between colored strips of any given layer.

The goal of tiling is thus to identify intersecting regions of all active strips through the layers. A contiguous region of intersection of strips from all layers is termed a *blob*. Defining such blobs in terms of mutual intersecting of strips from many layers is inherently a combinatoric problem. Given an initial pair of strips one has an outline of an initial blob. A strip from a third layer is added and each corner point from the two-strip blob must be tested to determine if it is inside the new strip. Likewise, all crossing points of the rays in the new strip and the rays in a particular strip in the prior blob must be identified and tested to determine if they are inside the other strip of the blob. This must process must be iterated for every strip in every layer and extended for subsequent layers.

Exploiting the symmetry of uniform pitch and angle for each layer allows the optimized formula of the *ray grid* to greatly reduce the cost of the computational primitives in this problem. Calculating the pitch index I_{ij}^{lmn} of layer n for rays i of layer l and j of m allows immediate testing of points against an *activity vector* which carries nonzero values for the rays associated with any active wires[1]. The steps of tiling are illustrated in detail in the following subsection.

4.1 Illustration of tiling

Tiling begins with information shown in figure 2. While the generated points are drawn to help guide the eye, it's of course important to know that they are not consulted in the tiling. They only provide the initial *strip activity* arrays. Contiguous regions of the activity array of each layer are found and define initial and compound strips. All such initial strips are illustrated in figure 3.

The first layers is immediately applied, forming one blob for every strip. These 1-layer blobs have no corner points as they are bound only in a single pitch direction and extend infinitely in its perpendicular. After the second layer is applied are the resulting 2-layer blobs have finite boundary. The order of layer application is not mandatory but some optimization is gained by applying the special horizontal and vertical detector boundary layers first. The result is illustrated in the figure 4.

Here, the vertical and horizontal boundary of a the active region of this (fictional) anode plane face is shown as a single square blob. It can be seen that some point which were generated outside the detector active region will not be surrounded by any blob. Next, the third layer is applied and it is the first actual wire plane as shown in figure 5. It breaks up the single 2-layer blob into three.

The fourth layer which is the second wire plane is applied to the 3-layer blob. It is shown in figure 6.

Given knowledge of the generated points, it is seen that some 4-layer blobs are now formed which do not surround any. These are termed *ghost blobs*. Processing techniques to remove these are described later. For now they must be accepted. However, adding the final layer shows that some portion of their area is excluded, while portions of the 4-layer blobs are split off and become ghosts. The final layer for the toy three-plane detector simulation is shown in figure 7.

To help guide the eye, this final result overlayed with the original composite strips in figure 8.

5 Blob Clustering

Blob cluster entails associating blobs found by tiling one time slice with those found by tiling its neighboring time slice. Association is defined in terms of overlap. Two blobs considered "overlapping neighbors" if any corners of one blob are inside any strips of the other when both blobs are projected

[1] This latter optimization idea came from a Wire-Cell student at BNL, Orgho Anoronyo Neogi.

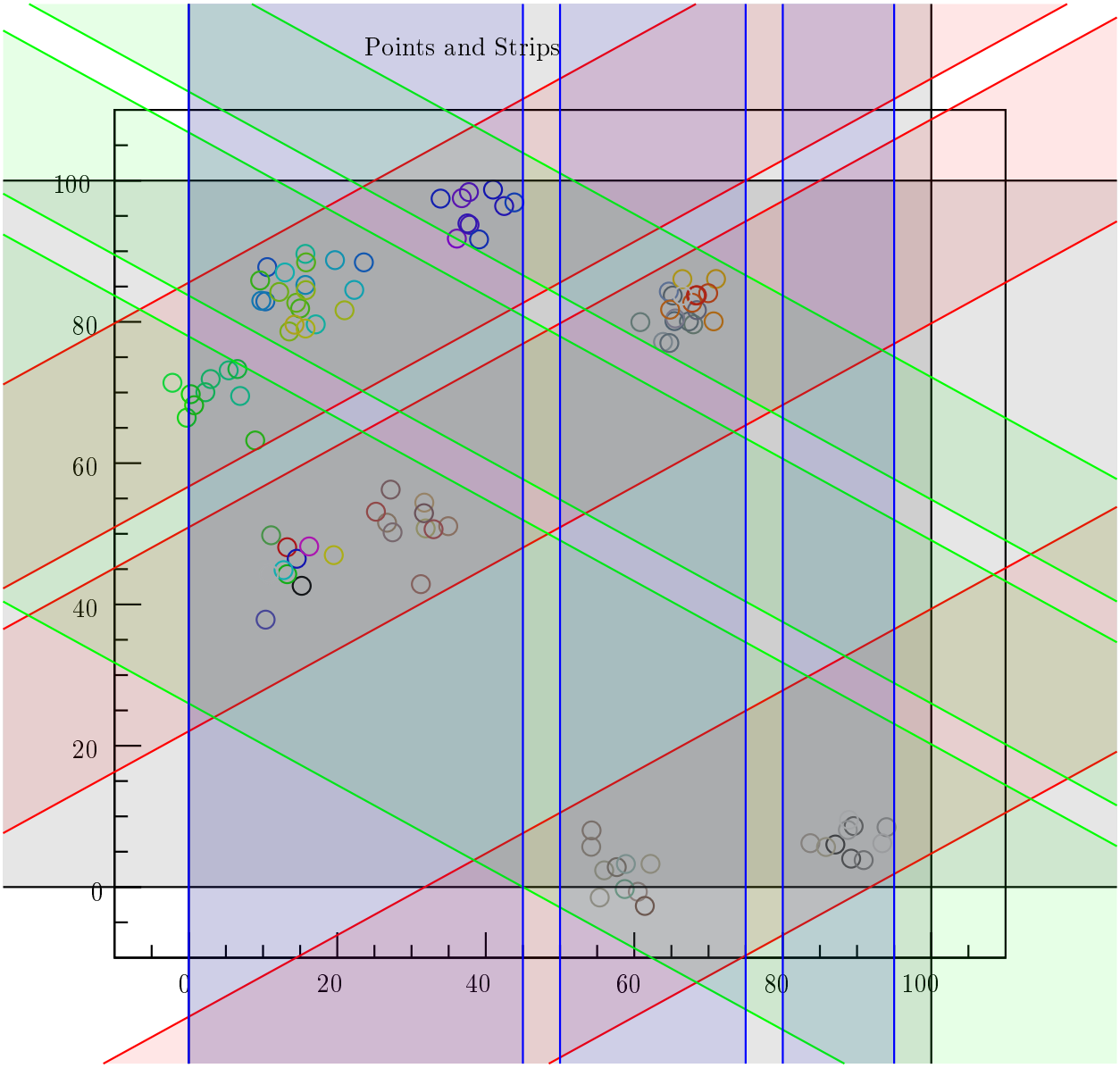


Figure 3: The contigous *strips* of activity for the toy simulation.

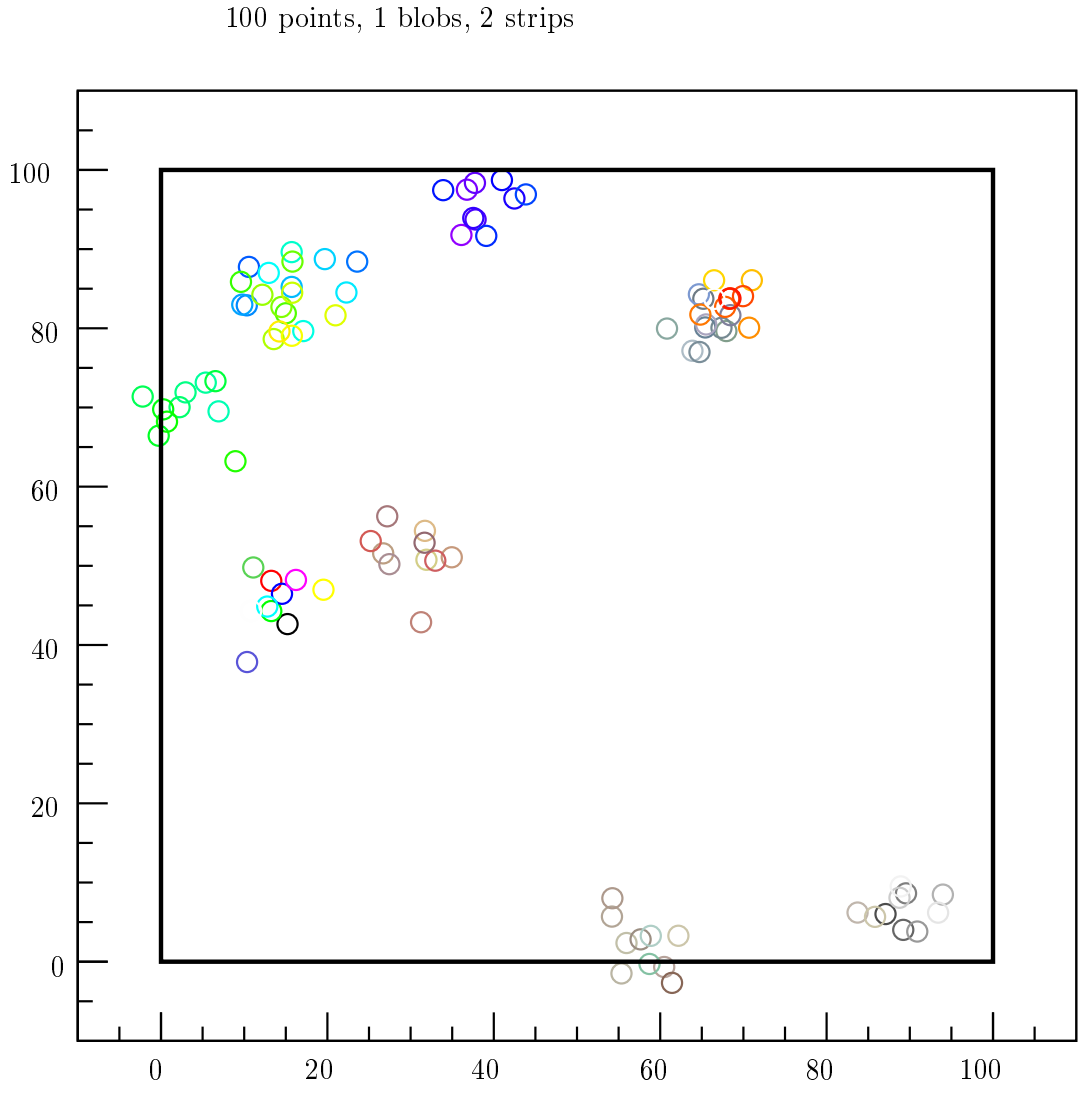


Figure 4: Trivial blob formed from the first two ray grid layers providing detector boundary.

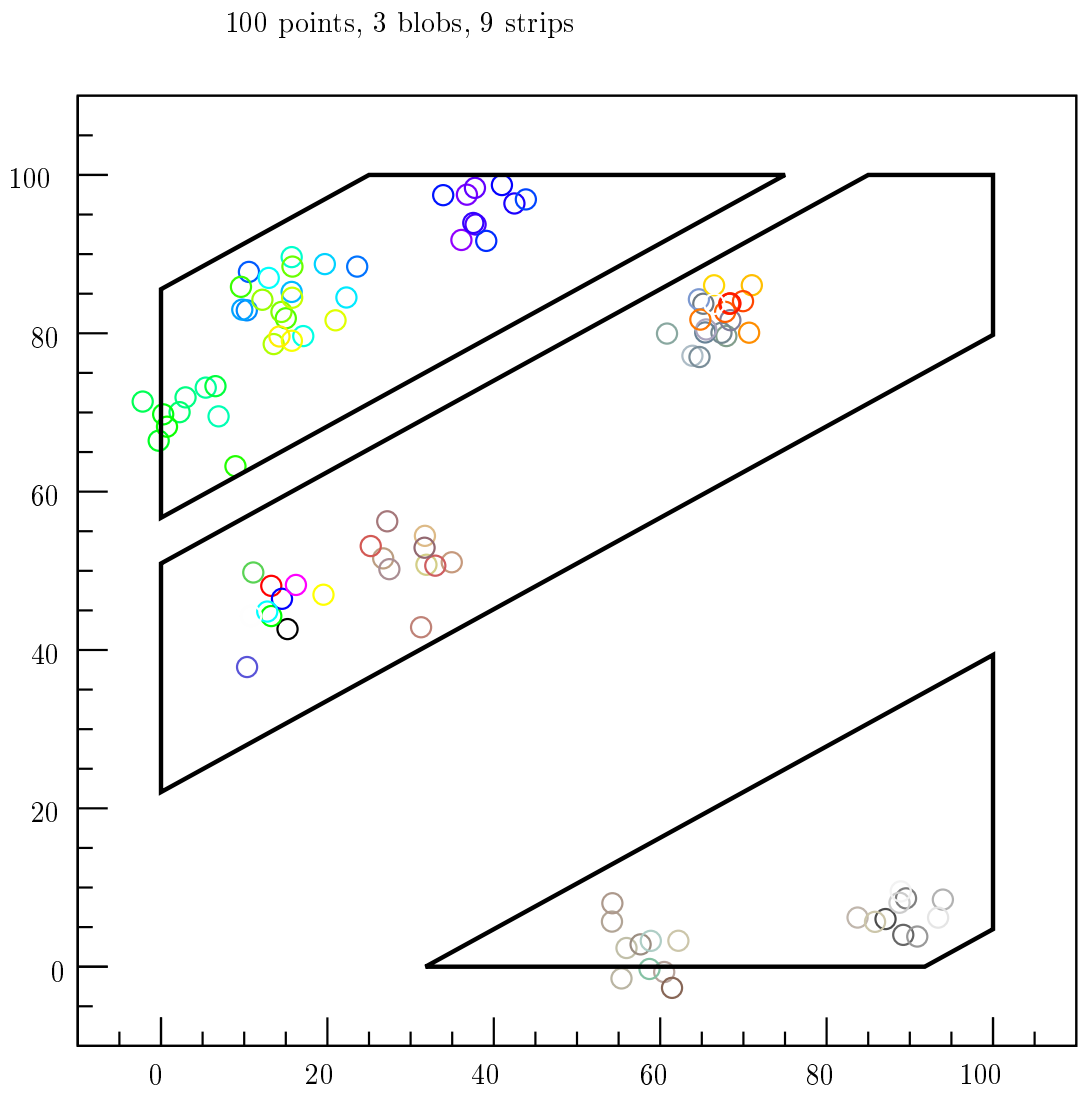


Figure 5: Addition of the third layer in the tiling.

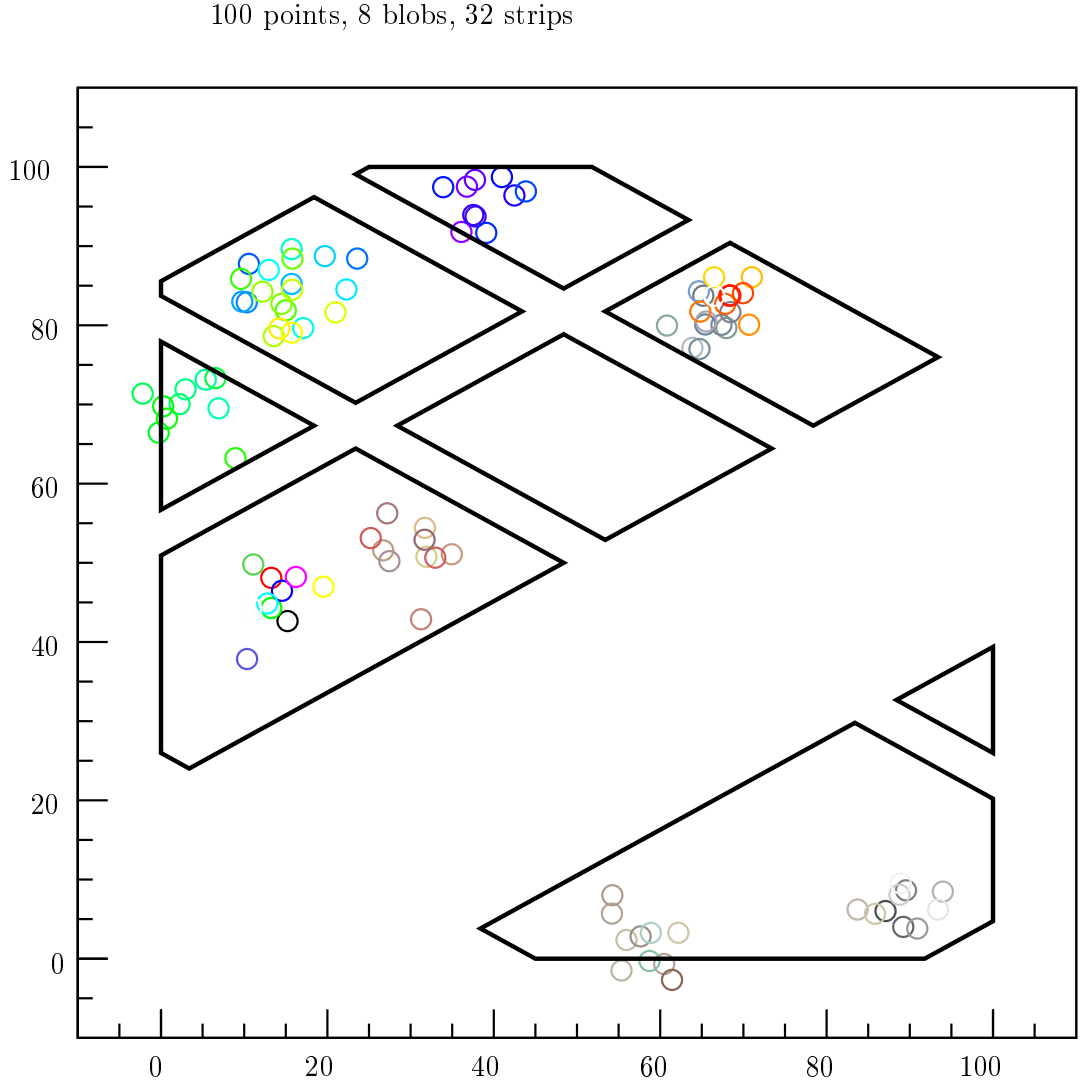


Figure 6: Addition of the fourth layer in the tiling.

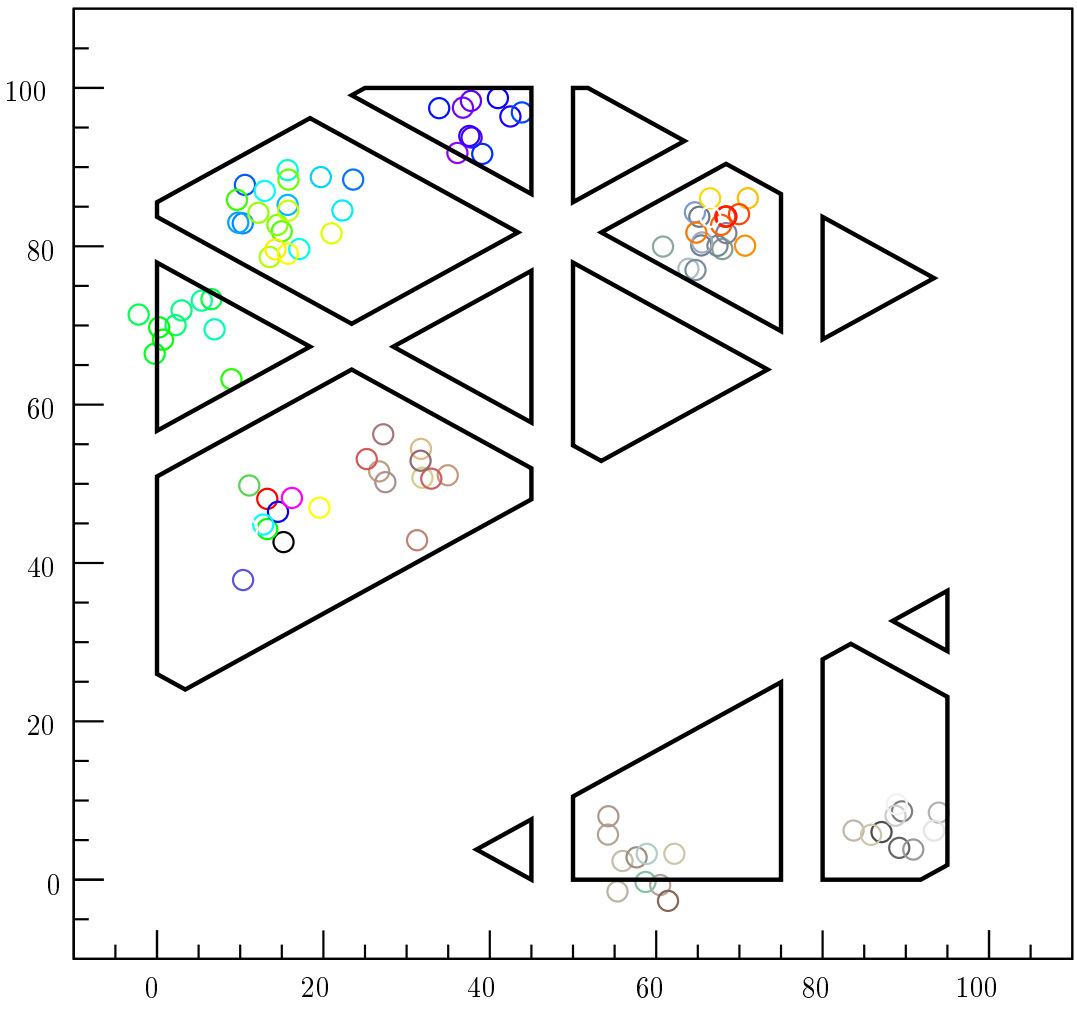


Figure 7: Addition of the fifth and final layer in the tiling.

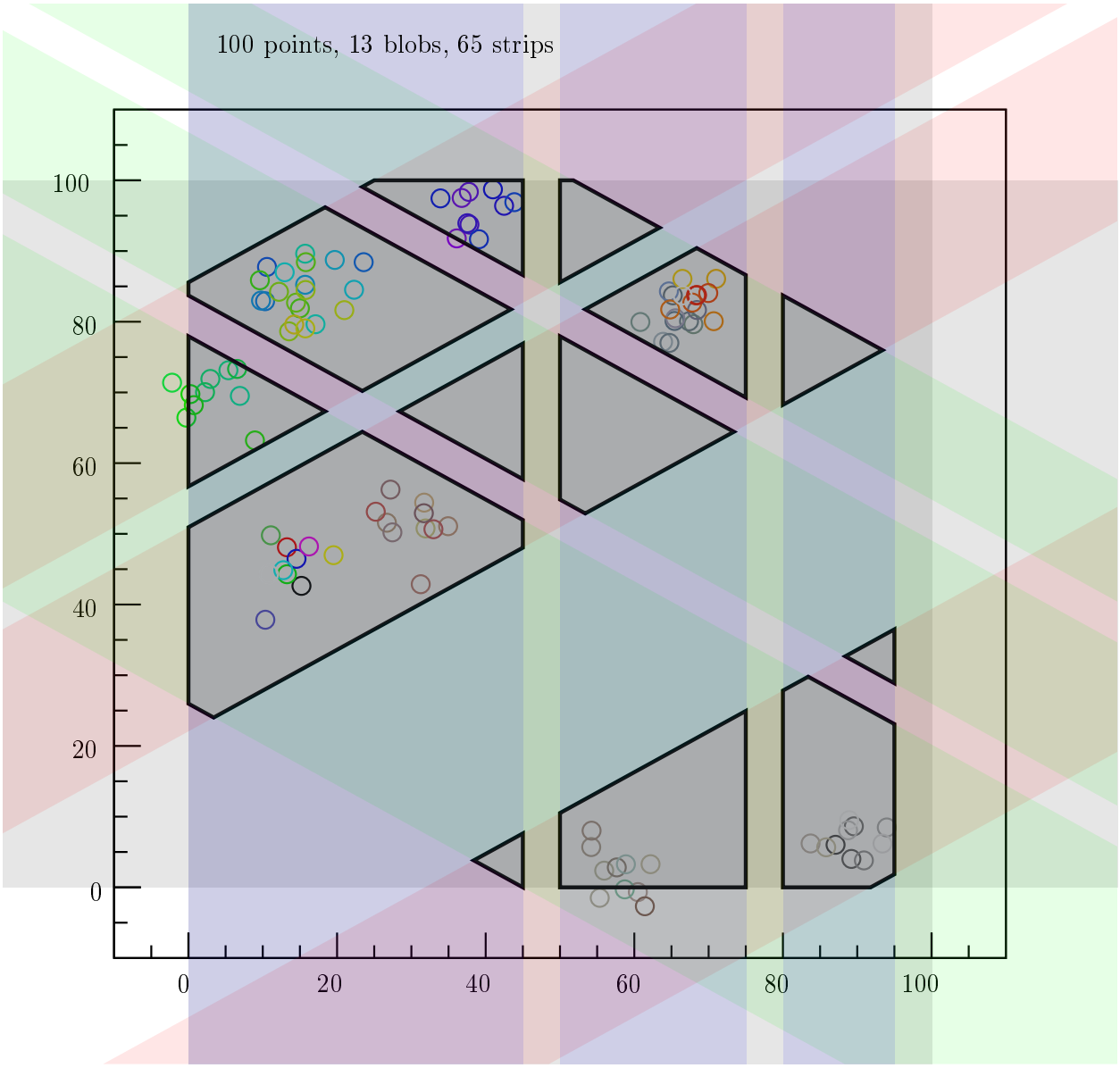


Figure 8: Final tiling results with activity strips overlayed.

to a common 2D plane. The association is considered to be symmetric. In particular if one blob fully surrounds another they are both considered overlapping each other.

The results of clustering are represented by an undirected graph called a *cluster graph*. This graph structure will be extended in the *grouping* and *solving* steps described below. Here, we describe the graph structure in generality covering these extensions. Each graph vertex has an associated `struct` of type `cluster_node_t`. This struct holds a WCT data interface as a shared pointer inside a `std::variant`. The type of pointer held can be determined with usual `std::variant` methods as well as the provided `code()` method which returns a `char` which is one of `"cwbsm"`.

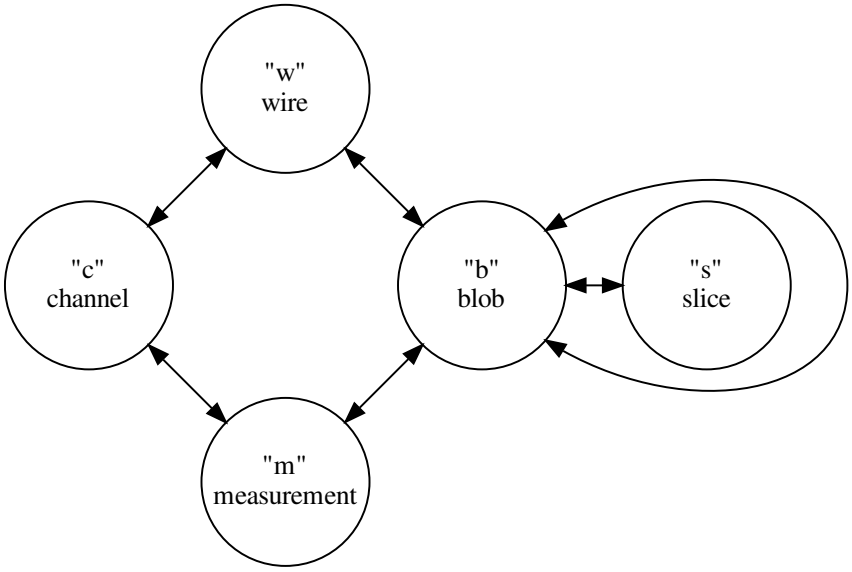


Figure 9: Illustration of the *types* of vertices and their allowed edges in a *cluster graph*.

Figure 9 shows a *type graph* describing what each vertex code stands for and what edges may be found between the different types of vertices. Any *cluster graph* is an instance of this *type graph* and some examples will be given later. It is important to note that not all *cluster graph* instances may contain all possible vertex types but when a type is present then only edges shown in the figure are allowed. For example, the initial clustering described above results in a graph lacking an `m`-node. This is added in *grouping* as described next. Another example and just as in physical existence no *cluster graph* shall have a direct edge between any blob and any channel. Their association is only meaningful with an intervening wire. Also take care to understand that a given `w`-node or `c`-node in a graph may be associated with many `s`-nodes (through `b`-nodes). That is, the same channel or wire is "in" many slices.

In WCT memory, the actual graph is represented as Boost Graph Library (BGL) graph object and is transferred between WCT components via an `ICluster` data interface shared pointer. The `ICluster` class also defines a `cluster_indexed_graph_t` which can wrap the BGL graph to provide shared pointer based node access which is more convenient to work with than the integer number based graph vertices of the basic Boost Graph object. This wrapper basically lets the user avoid maintaining their own `std::map` lookups.

6 Blob Solving

In tiling, the measurements are only considered in relation to some threshold to determine if a ray is "active". This section describes how the value of measurements are used to estimate the number of electrons within a blob. Potentially a threshold can be placed on this estimated "blob charge" in order to eliminate potential "ghost blobs".

6.1 Formalism

Solving (here) means, conceptually, to invert the equation $\vec{m} = R \cdot \vec{s}$ in order to determine $\vec{s}$, and where:

$\vec{m}$ is a vector of measured values (each from a set of channels)

$\vec{s}$ is a vector of sources of values (ionization activity in the region of a "blob")

R is a "response" or "geometry" matrix which associates sources to measures

However, in actual fact, there is measurement uncertainty that must be accounted for. A covariance error matrix C is introduced and inverting the matrix equation is converted into the minimization of a of a χ^2 function:

$$\chi^2 = (\vec{m} - R \cdot \vec{s})^\intercal \cdot C^{-1} \cdot (\vec{m} - R \cdot \vec{s}) \tag{4}$$

As an optimization, an additional penalty term which is linear in the solution vector is added.

$$\chi^2 = (\vec{m} - R \cdot \vec{s})^\intercal \cdot C^{-1} \cdot (\vec{m} - R \cdot \vec{s}) + \vec{\lambda} \cdot \vec{s} \tag{5}$$

The $\vec{\lambda}$ vector may be chosen freely to influence the relative weight of each source. Some level of bias is introduced by any particular choice. A heuristic is initially chosen that biases the solution against any blobs lacking overlapping neighbors (as described in Section 5).

Finding a solution that minimizes this function is done using the RESS package. It requires a function in a form that does not have a covariance matrix of measurement uncertainties. In the case of the problem here, measurements are on channels and when there is no cross talk the covariance matrix is diagonal and given simply in terms of an uncertainty σ_i on each measurement m_i. This allows a substitution, written in index notation, of:

$$m_i \to m'_i = \frac{m_i}{\sigma_i},\ R_{ij} \to R'_{ij} = \frac{R_{ij}}{\sigma_i}. \tag{6}$$

The input to RESS is then $\vec{m}'$, R' and $\vec{\lambda}$. Optionally, an initial starting solution $\vec{s}_o$ may be provided.

6.2 Problem granularity

Given knowledge of the blobs found by tiling and the full association of measurements in channels to the physical location of wires (and their associated rays) there is some freedom to choose $\vec{m}$ along with a corresponding implication for the definition of R. There are three obvious choices which are called here, fine, medium and coarse grained.

At its most fine, each element of $\vec{m}$ can be identified with a single channel. This such a vector would be constructed by enumerating the activities depicted in figure 2. After identifying blobs through tiling it becomes obvious that this channel level information is degenerate in terms of how

it connects to blobs. That is, certain subsets of channels all go to the same subsets of blobs and thus considering them individually can not provide additional information compared to considering them in aggregate.

Thus, a "medium grained" choice is to combine measurements into sets of maximum possible entries such that all the measurements in a set are connected to the same blobs. For example, blue strip on the left hand side of figure 8 may be split into as many of six sub-strips, each covering a unique set of blobs. Such a definition also splits blobs across multiple substrips.

The final natural choice is to retain the strips as shown in 8. That is, to group measurements in each layer such that they do not split any blob but which do not include wholly separate blobs.

From fine to medium scale, no information is lost and the size of the problem is reduced. From medium to coarse some information is discarded, that which is related to some of the blob boundaries, however the size of the problem is reduced even further. The current implementation elects the coarse choice. In the future, the medium choice should be investigated.

6.3 Grouping

One must first define the matrix equation to solve. To do this, there is the need to *group* the measurements following the coarse strategy as just described. This ends up being a conceptually trivial matter once it is couched in graph-theory terms. The procedure is described in this section.

First, a temporary *layer graph* M_l is created for each layer l within each slice. A *layer graph* is similar to a *cluster graph* but allows only `c`-nodes and `b`-nodes. Then, the graph-theoretical operation of identifying *connected component subgraphs* is applied to an M_l graph. Each of the resulting subgraphs is itself connected but is disconnected from the other subgraphs. Then, for each subgraph, the values of all `c` nodes are collected and represent a coarse grained *measurement* (Section 6.2). The measurement is added to the graph as an `m`-node with edges connecting to one `b`-node and a set of `c`-nodes. This procedure repeated for each layer in the slice resulting in one `(b-m)` edge for each view (wire plane) provided by the blob. Again, this multiplicity is governed by the choice of *coarse grained* grouping strategy. The *medium grained* strategy would produce a higher multiplicity of `(b-m)` edges. The final resulting *cluster graph* may be termed a *grouped graph* as it adds the `m`-nodes that group channels.

Figure 10 shows an example *grouped graph* produced from a test pattern formed by the ionization activity of two ideal, minimum ionizing tracks which cross at their midpoints. Figure 11 shows the blobs found through tiling each slice of this test pattern. Looking carefully at the graph structure one may see regions of simplicity at either end. These correspond to the upper and lower legs of the "X" formed by the two tracks. In these regions there is no ambiguity produced and thus the legs are resolved in simple tiling. In the center of the "X" where the two tracks cross, tiling is unable to resolve two tracks at all and thus a single column of blobs are found. This corresponds to the middle part of the graph that lacks complexity. Finally, in the transition between no ambiguity to full ambiguity there are many blobs found in each slice, most of them ghosts. This complexity can be seen in the graph structure as highly connected regions.

6.4 Solving

At this point, a solution may be attempted for $\vec{s}$, as defined in Section 6.1. The input to this step may be any graph of the form described in Section 6.3. The `b`-nodes of such a graph begin with zero associated "value". Solving is done by transforming a *grouped graph* into the matrix equation 5. The $\vec{m}$ vector is simply an ordered list of the values of the `m`-nodes. Their value is calculated by

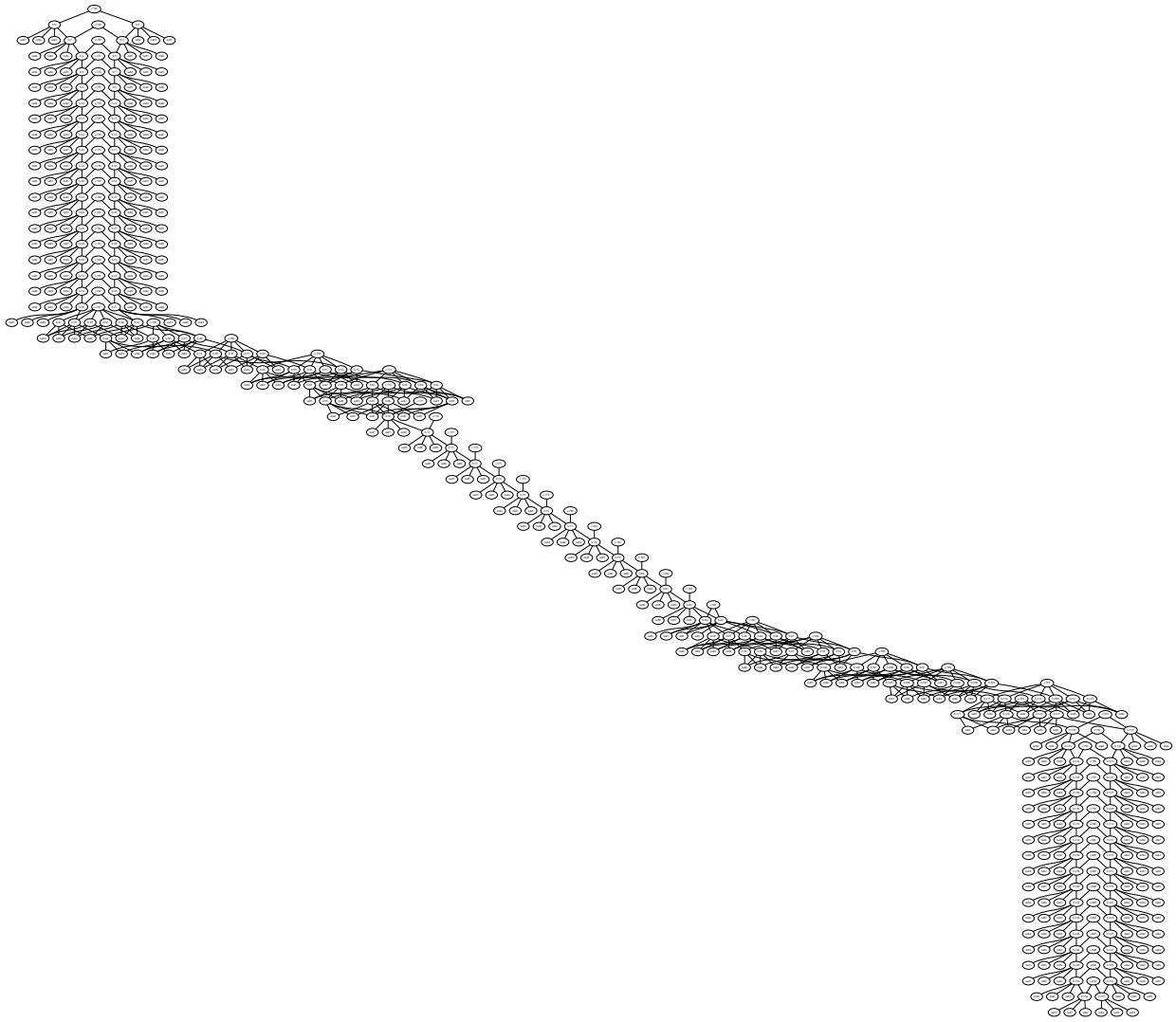


Figure 10: Example of graph produced by grouping given activity from a test pattern of two ideal tracks which cross at their midpoints forming an "X" in space. See figure 11 for a spatial view. See text for description.



Figure 11: The result of tiling a test pattern of two crossing ideal tracks. Coloring simply indicates the slice number in which a blob resides.

using their set of channels to look up the samples recorded in the associated `s`-node's slice. The initial $\vec{s_0}$ may be zero.

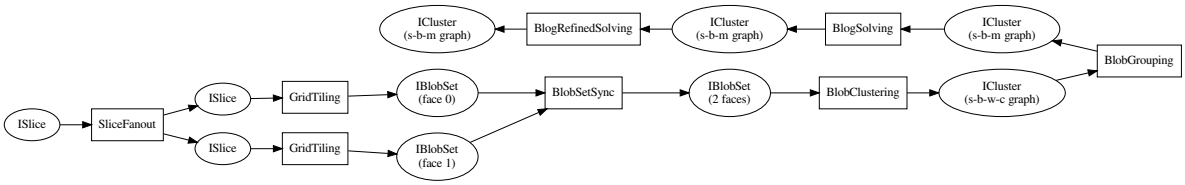


Figure 12: An illustrative example chain for solving. Upstream a frame is narrowed to one APA and then sliced. Each slice is sent to a per-face tiling which produce sets of blobs that are joined back into a per-slice set. These blobs are clustered, grouped and finally "solved". Subsequent solvers may be applied. The results may finally be dispatched (not shown) as 3D space/charge points or projected back into 2D views depending on the needs of the application.

However as the output of solving also produces the same type of graph as its input a chain of solving steps may be constructed. In such a chain the $\vec{s}$ output of one solution may be transferred to an input starting $\vec{s_o}$ of the next as illustrated in figure 12. As hinted in this illustration, some parallel sub-pipelines may be defined. The tiling is a per-face process and the entire chain shown is a per-APA process. Downstream the cluster may be combined if the application is to study intra-APA activity. A full WCT execution graph is shown in figure 13. This is one example. Other configurations may slice the graph after the digitizer and replace the downstream subgraph with one that provides raw waveform data. Or, the origin node which provides the ideal track energy depositions (`TrackDepos`) may be replaced with a richer source of depositions produced by Geant4 (eg, via LArSoft's `LArG4` module).

Continuing on with the solving step, The matrix R is simply the connection matrix of an induced subgraph formed by selecting only `(m-b)` edges. The weight vector $\vec{\lambda}$ may is formed in an ad-hoc manner. An element of the vector starts with the magic number `9`. If it corresponding blob has an overlapping neighbor from a prior slice it is reduced by a magic factor of `3`. Likewise, if the subsequent slice at least one overlapping blob, another reduction of `3` is taken. Other weighting schemes may be investigated in the future.

The solving itself is performed by the RESS package and the elements of the resulting $\vec{s}$ solution vector are transferred back to a copy of the input graph, blob by blob. The solution is graphically displayed as colors in figure 14.

Thus, solving assigns a blob a (potentially) non-zero value which represents an estimation of the amount of ionization electrons that may be residing inside the blobs spatial extent. When successful, blobs with near zero value may be eliminated as being likely "ghosts". However, as shown in figure 14 some as "ghost" blobs are retained, their charge is lower than MIP level but nonzero and so thresholds must be applied carefully, if at all.

Figure 15 shows a zoom in of reconstructed activity as solved blobs. The job DFP graph is shown in figure 16. Note that the per-APA pipelines are constructed by "slicing" the DFP graph in figure 13 in front of the SP components and attaching upstream NF and channel selector components. The waveform input data is then taken from `art::Event`. After SP, 2-way fanouts are used. One port sends the per-APA frame into imaging and the other sends the same frame into a 6-way fanin so that the a full 6-APA frame can be merged and sent back to `art::Event` for saving as a vector of `recob:Wire` objects.

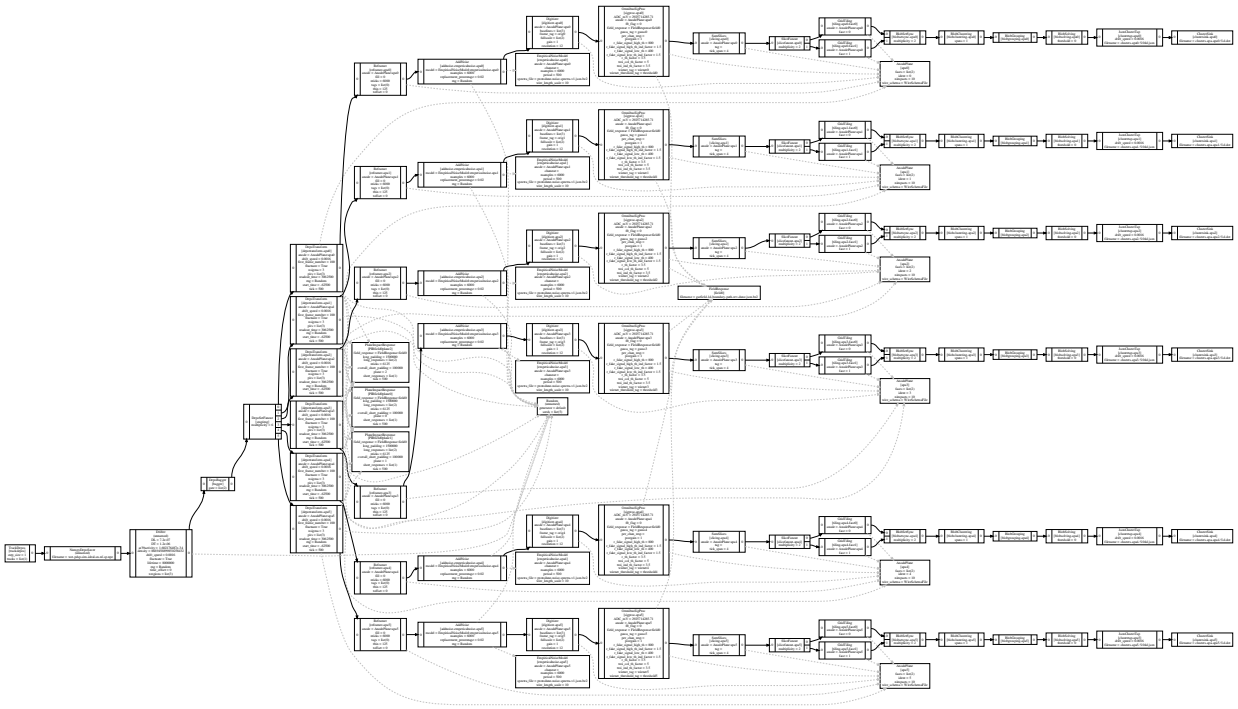


Figure 13: Graphical depiction of the Wire-Cell Toolkit simulation, signal processing, tiling and solving job for the six APA protoDUNE-SP. Energy deposits enter on the left, fan-out to a per-APA pipeline. Each pipeline applies field and electronics response to produce induced signals. Noise model is then applied and their sum is digitized. The signal processing deconvolves the field and electronics response and the resulting reconstructed signals are sliced, tiled, clustered, grouped, solved and the final resulting clusters are dumped for plotting and evaluation.

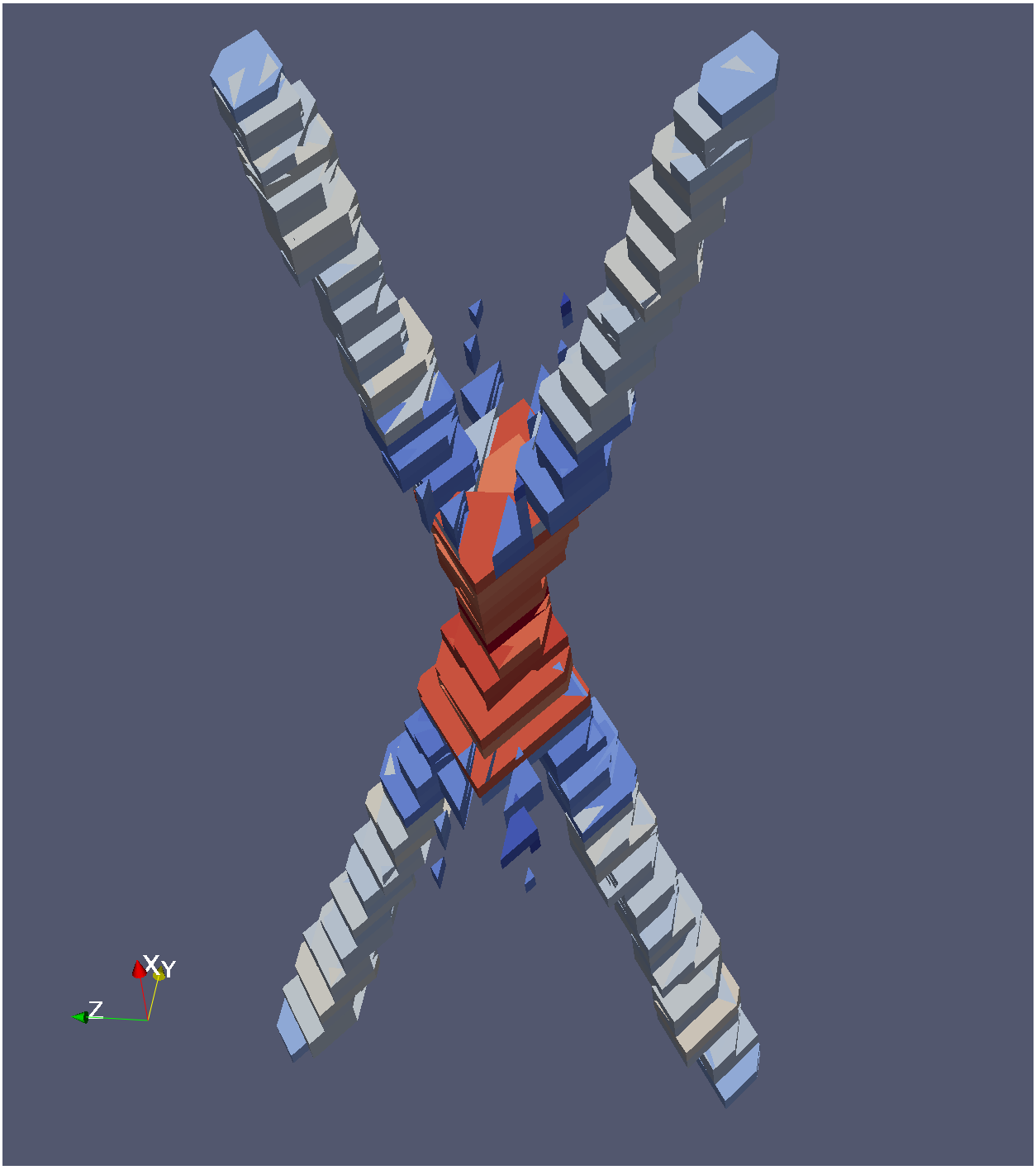


Figure 14: Solved blobs. Colors indicate the amount of ionization electrons likely to be contained in any given blob region. The pale gray indicates approximately one MIP, the rust/red color approximately two MIPs. Most of the blue is above 10k electrons.

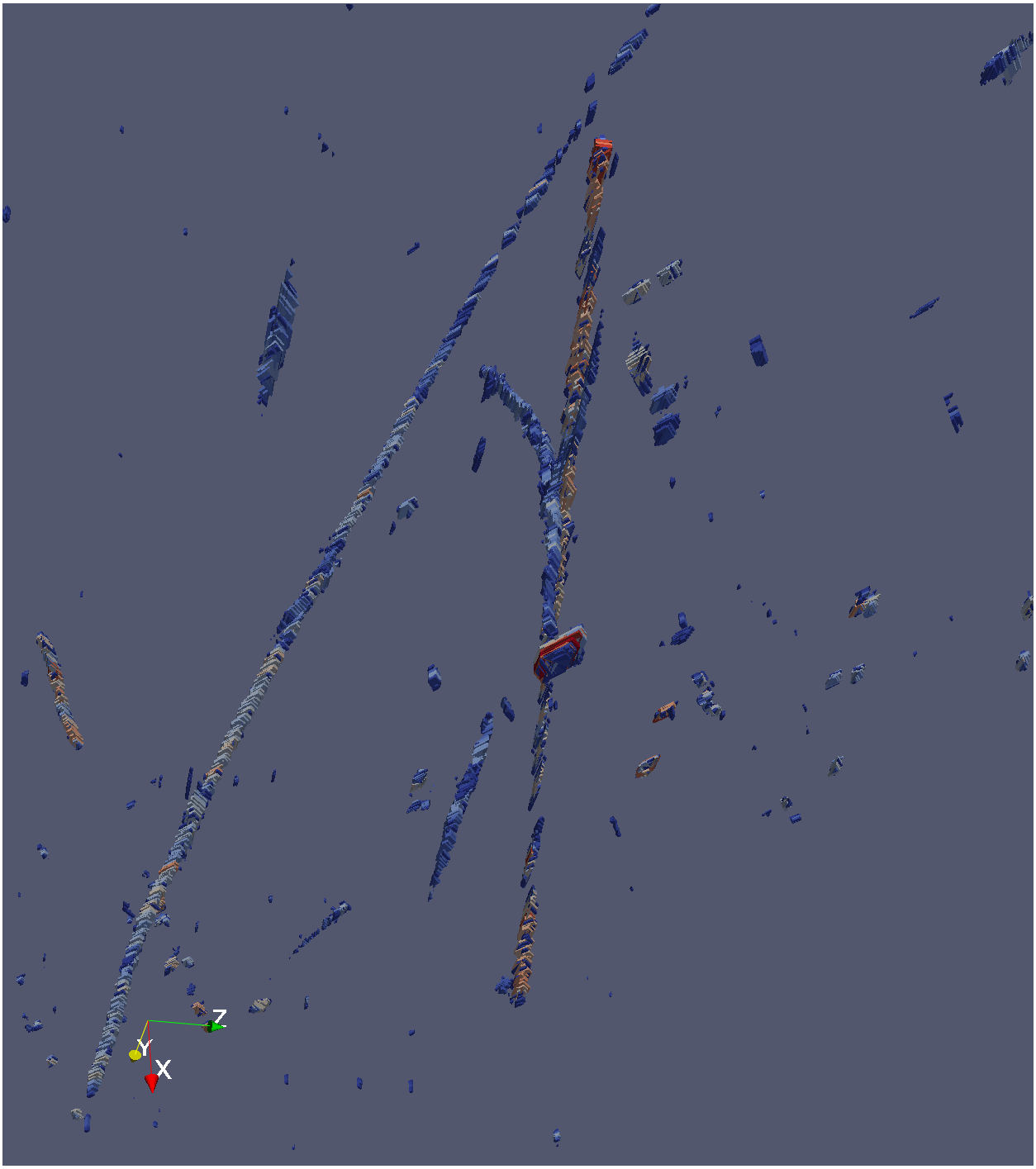


Figure 15: A zoom in on a region of solved blobs from ProtoDUNE-SP data.

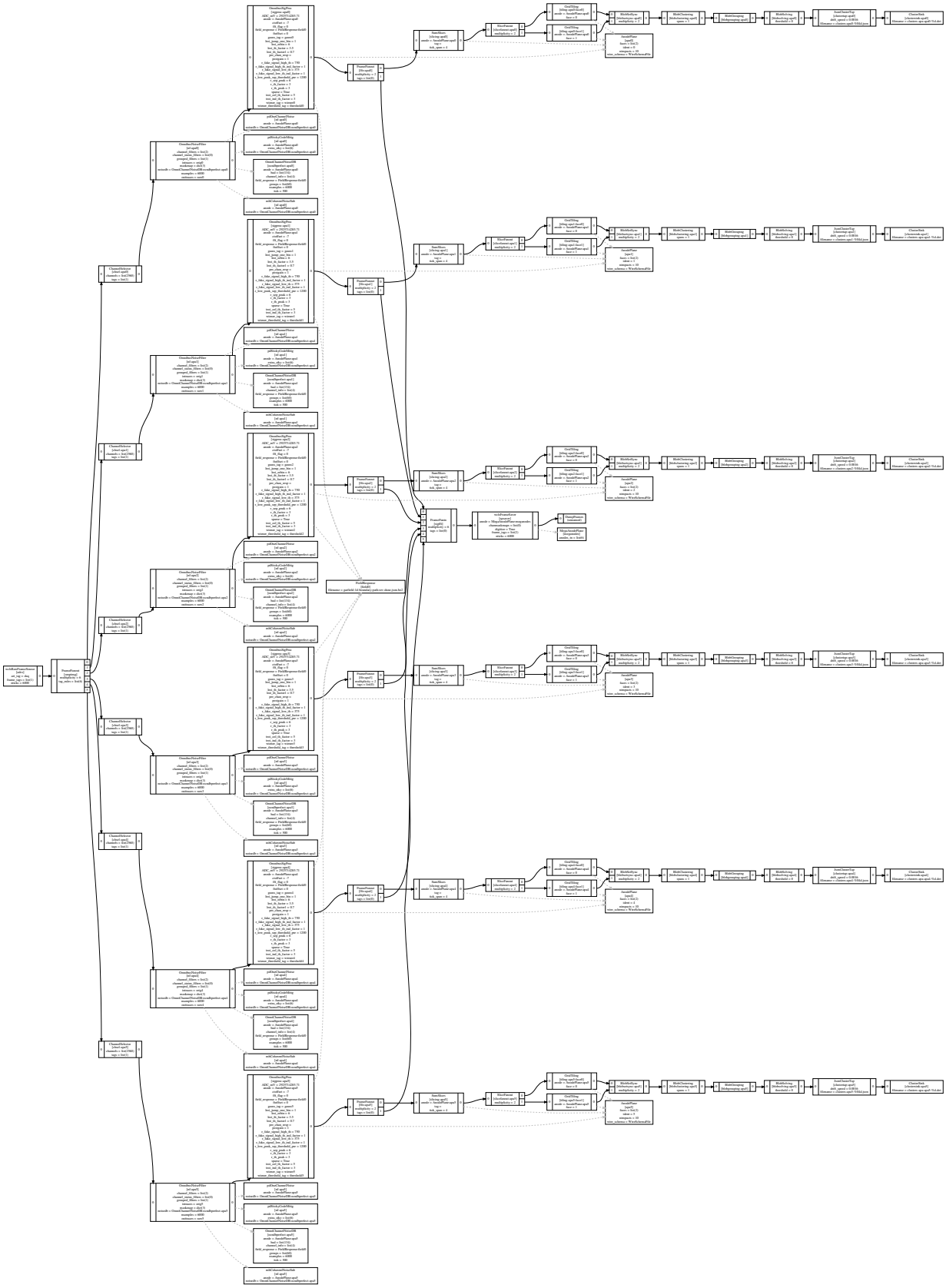


Figure 16: The WC/LS DFP graph for running NF+SP+IMG from inside art.