mu3eTrirec and mu3eEfficiency

Charlie Kinsman

February 1, 2023

1 Overview

The purpose of mu3eTrirec is to reconstruct the tracks of the particles using the output file from mu3eSort. There are output trees, segs and frames. The segs tree contains information on individual segments or tracks, so one entry corresponds to one segment. The frames tree contains vectors of segments and effectively categorises these segements. It should be remembered that when in cartesian, the z direction is centered around the stopping target where the positive direction is parallel to the beam direction. The positive y axis is pointing upwards, in reference to the detector. The x axis is right handed. Some of the products listed in the root files are carried through from mu3eSim and mu3eSort. Mu3eEfficiency is part of the dev branch rather than the main branch. It calculates efficiencies of track reconstruction when certain factors are changed, such as the hit efficiency of a layer being reduced or a hit in a track being removed entirely. The dev branch as well as having it's own executable with it's own outputs, also has additions to the Trirec branch to allow for some changes to be made before efficiencies are calculated.

2 mu3eTrirec - Main Branch

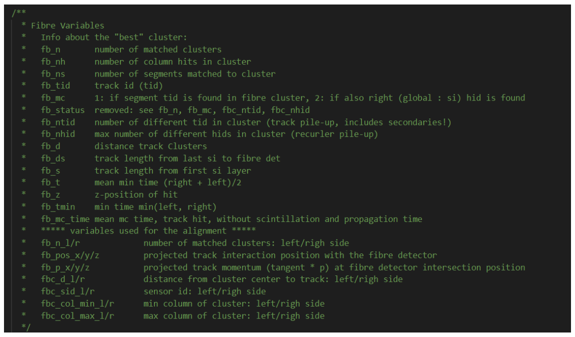
2.1 Overview

The main branch of the simulation does not contain efficiency calculations, it does however contain truth information and has the groundwork for Trirec that the dev branch is built on.

2.2 To Add

hid: Hit ID. sid: silicon hit id. x0, y0, z0...:. n_shared_hits and n_shared_segs:. tan01, lam01:.nsg, nhg, ns:. s3n, s6n_0:.mc_vpca:. prop_z:.

Util:units.hpp



```
/**
  * Fibre Variables
  *   Info about the "best" cluster:
  *   fb_n      number of matched clusters
  *   fb_nh     number of column hits in cluster
  *   fb_ns     number of segments matched to cluster
  *   fb_tid    track id (tid)
  *   fb_mc     1: if segment tid is found in fibre cluster, 2: if also right (global : si) hid is found
  *   fb_status removed: see fb_n, fb_mc, fbc_ntid, fbc_nhid
  *   fb_ntid   number of different tid in cluster (track pile-up, includes secondaries!)
  *   fb_nhid   max number of different hids in cluster (recurler pile-up)
  *   fb_d      distance track Clusters
  *   fb_ds     track length from last si to fibre det
  *   fb_s      track length from first si layer
  *   fb_t      mean min time (right + left)/2
  *   fb_z      z-position of hit
  *   fb_tmin   min time min(left, right)
  *   fb_mc_time mean mc time, track hit, without scintillation and propagation time
  *   ***** variables used for the alignment *****
  *   fb_n_l/r            number of matched clusters: left/righ side
  *   fb_pos_x/y/z        projected track interaction position with the fibre detector
  *   fb_p_x/y/z          projected track momentum (tangent * p) at fibre detector intersection position
  *   fbc_d_l/r           distance from cluster center to track: left/righ side
  *   fbc_sid_l/r         sensor id: left/righ side
  *   fbc_col_min_l/r     min column of cluster: left/righ side
  *   fbc_col_max_l/r     max column of cluster: left/righ side
  */
```

Figure 1: Table of fb values to add into the document.

2.3 Segs Tree

2.3.1 runId

The run ID refers to the seed used in the simulation, and therefore gives a repeatability aspect. As this is all in the same branch, technically it refers to the run of the simulation that this particular track/entry belongs to.

2.3.2 eventId

The event ID refers to the frame from which the track was reconstructed. Like the run ID, it is labeled by each event, so each event will have it's own particular ID depending on the frame from which it was reconstructed.

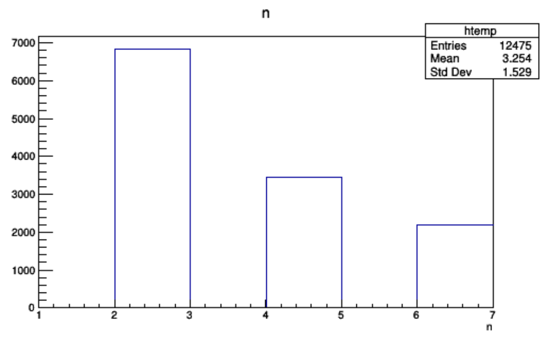


Figure 2: Histogram of n. In this instance n refers to the number of triplets used to reconstruct a track. This is different to the n used in the frames tree

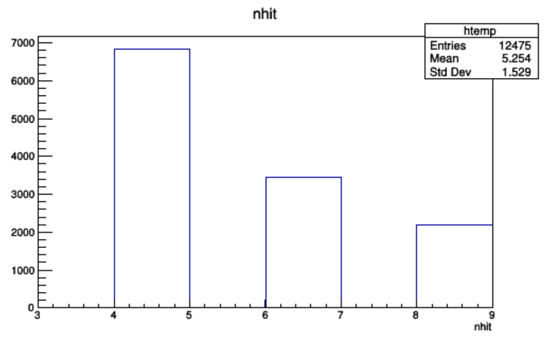


Figure 3: Histogram of nhit. nhit refers to the number of hits used to reconstruct a track. This differs from n because n refers to the number of triplets used to reconstruct a track

2.3.3 n

n refers to the number of triplets found and used to reconstruct each track. An example histogram of this distribution is shown by figure 2.

2.3.4 ndf

ndf refers to the number of degrees of freedom that are associated with each track. This is proportional to the number of triplets ($ndf = 2 \cdot (n - 1)$).

2.3.5 nhit

nhit refers to the number of hits used to reconstruct a track. An example histogram of nhit is given by figure 3.

2.3.6 r

r is the three dimensional radius or curvature of the helix of the fitted track/segment. An example histogram of r is given by figure 4.

2.3.7 rerr2

rerr2 refers to the square of the error of the radius of the helix for the given track. An example histogram of rerr2 is given by figure 5.

2.3.8 p

p refers to the momentum of the fitted track.

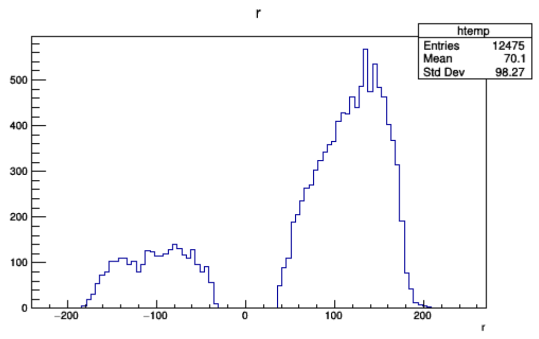


Figure 4: Histogram of r. In this instance r refers to the radius of the fitted/reconstructed tracks. The main distinct feature of this bimodal distribution is the second distribution being much more populated then the first. This is down to the charge of the particles, there is twice as many positive particles than negative particles.

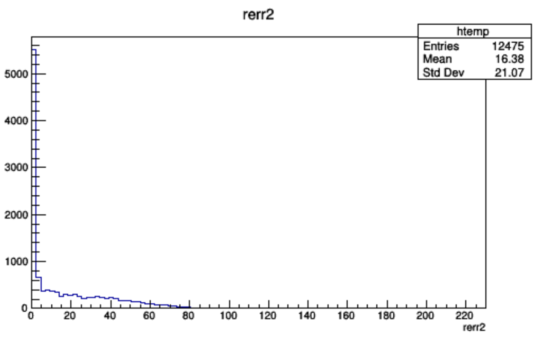


Figure 5: Histogram of the squared error of the radius of the fitted tracks. These can be related to the r histogram when computing efficiencies and resolutions etc.

2.3.9 perr2

Like rerr2, this is the square of the error of the momentum.

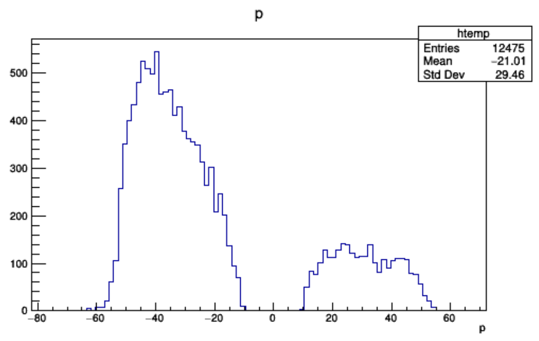


Figure 6: Histogram of momentum of the fitted tracks.

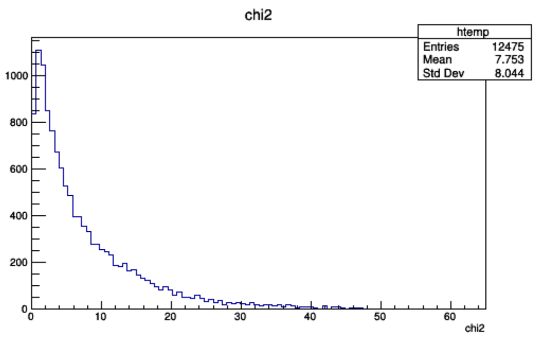


Figure 7: Histogram of the chi squared distribution of the fits. The significance of the distribution of fits can be shown in table 7

2.3.10 chi2

This is the χ^2 values of the fits. These are individual χ^2 values that have been placed in the histogram shown in figure 7.

2.4 Segs MC Tree

2.4.1 Overview

The MC tree refers to the monte carlo aspect of the simulation. This effectively gives us the truth information of the tracks.

2.4.2 mc

mc refers to if the hits on the in a given track are from the same track or different and if they are in the correct sequence. This value is 1 for a given track if this condition is true. This therefore somewhat implies if the track is true or not.

2.4.3 mc_prime

mc_prime is 1 if mc is 1 and the first segment of the track is correct for a given particle. As the first segment is the most important, if this is not correct for a particle then the rest of the track is not going to be correct either.

2.4.4 mc_type

mc_type refers to the origin of the particle in question. As this is all truth information (mc), the correct particle and from which decay it originated can be identified. 5.1 shows the table of digits that this refers to.

2.4.5 mc_pid

mc_pid refers to the particle ID. Again using the same numbering system as before. The output of this is different, such that the entries which are not from the relevant decays are removed and placed in the negative section. The relevant decays and particles are therefore in the positive section.

2.4.6 mc_tid

mc_tid is the track ID of the particle. This is the truth information, therefore this can relate certain tracks to other tracks or to the mother particle.

2.4.7 mc_mid

mc_mid is the same as mc_tid but is the ID of the mother particle. This can therefore be used by the vertex analysis to produce the truth information of the decay vertices.

2.4.8 mc_p

mc_p refers to the momentum of the particle after decay. This momentum combined with the transverse momentum can be used to calculate the longitudinal momentum. An example of distribution can be seen in figure 8.

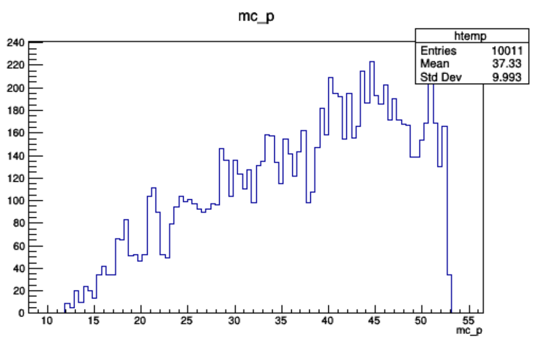


Figure 8: Histogram of the particle momentum. This can be used to calculate the helix parameters and the decay products. This distribution falls to 0 at $53MeV$ because that is the muon mass and the muons have been stopped.

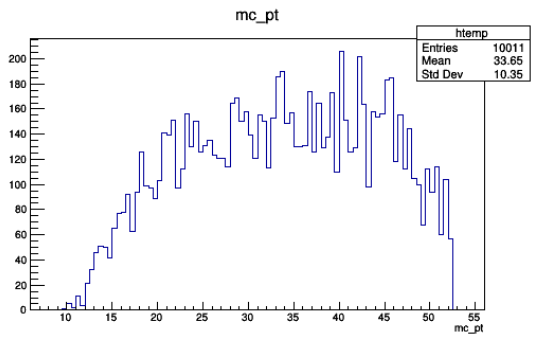


Figure 9: Histogram of the transverse momentum. This can be used to calculate the helix parameters and also the decay products. This distribution also falls to 0 at $53MeV$ because that is the muon mass.

2.4.9 mc_pt

mc_pt refers to the transerse momentum of the particle. This can be used to calculate the radius of the helix of the particles. An example of distribution can be seen in figure 9.

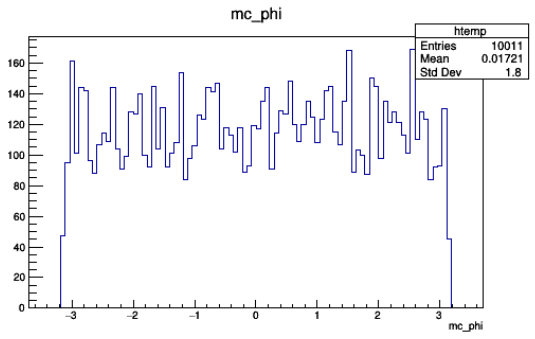


Figure 10: This shows the truth information for the phi positions of the vertices.

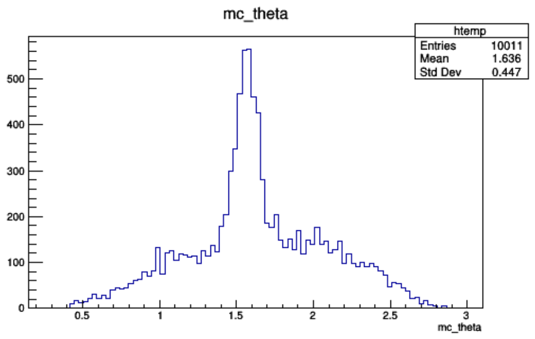


Figure 11: This is the truth information for the theta positions of the vertices.

2.4.10 mc_phi

mc_phi is one of the parameters of the helix of the particles. This can be used with the other parameters to reconstruct the tracks of the particles. This particular parameter is a transverse parameter as it refers to the angle completely in the $x-y$ plane. It's zero point shouldn't matter, if studying the total distribution, as any distribution that is produced should be uniform. This is the phi distribution of the vertices. An example of distribution can be seen in figure 10.

2.4.11 mc_lam

Like mc_phi, mc_lam is another of the parameters of the helix tracks of the particles. Unlike mc_phi however, mc_lam is a longitudinal parameter. This is the angle between the z axis and the $x-y$ plane. Where $\lambda = 0$ is pointing in the positive z direction. Also $\lambda = \frac{\pi}{2} - \theta$. This is the lambda distribution of the vertices.

2.4.12 mc_theta

Like mc_lam, mc_theta is another longitudinal parameter of the helix track of the particle. This parameter is the opposite of mc_lam in the sense that $\theta = 0$ is pointing in the negative z axis. Again $\theta = \frac{\pi}{2} - \lambda$. This is the theta distribution of the vertices. An example of distribution can be seen in figure 11.

2.4.13 mc_vx

mc_vx refers to the origin vertex position in the x direction. This is not the x component of the velocity. An example of the distribution is shown in figure 12. As seen, the distribution peaks around $0mm$ and quickly drops off. As can be seen the distribution is approximately even which is to be expected as the stopping target is itself symmetrical around the 0 position.

2.4.14 mc_vy

mc_vy refers to the origin vertex position in the y direction.

2.4.15 mc_vz

mc_vz refers to the origin vertex position in the z direction. An example of a z distribution is given by figure 13. As can be seen, if a cut of the histogram is taken between $-100mm$ and $100mm$, the distribution is fairly even as is the stopping target. It also starts to pick up at $-50mm$ and $50mm$ which is the dimensions of the stopping target in the z direction. It also slightly picks up on the downstream side of the stopping target which makes sense as it becomes increasingly unlikely that a muon is stopped on the tip of the upstream side of the target. There is also a small peak before the stopping target around $-150mm$, this represents muons that either decay before hitting the stopping target or decay vertices that originate from particles that have previously decayed on the stopping

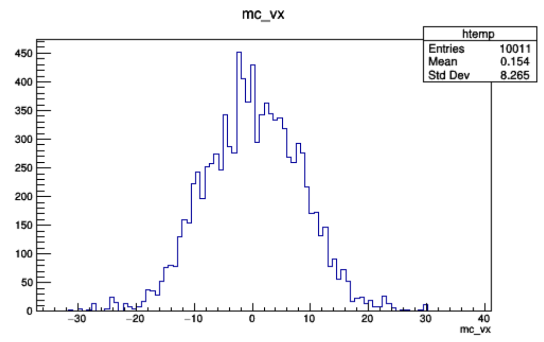


Figure 12: This shows the truth information of where the vertices of the decays are. In this instance, this is the x position of the vertex.

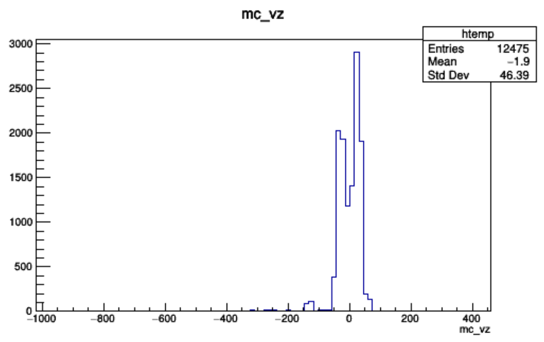


Figure 13: Again this is the truth information of the vertices of the decay. This histogram is the z position.

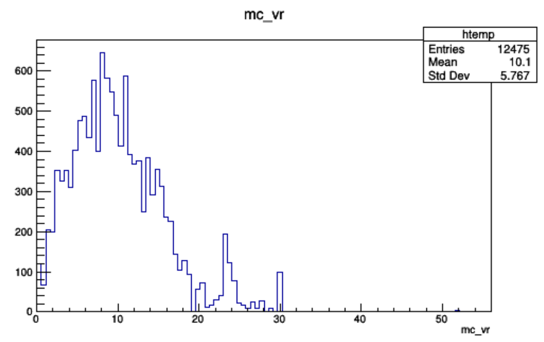


Figure 14: This is the truth information of the radial component.

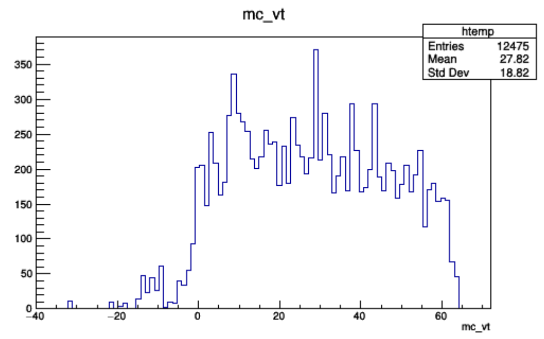


Figure 15: This is the truth information of the time of the decay at the vertex.

target. Another distribution of muons stopped in the z direction is given in the technical design of the stopping target, which shows the bimodal distribution of muons stopped in the z direction.

2.4.16 mc_vr

mc_vr refers to the origin vertex position in the radial axis. An example of this distribution is shown in figure 14. This does not peak around the $0mm$ mark which given the 0 mark refers to either the two tips of the stopping target and therefore refers to less surface area in comparison to $10mm$ for instance, this result makes sense. The first distribution drops off at approximately $20mm$ which corresponds to the radius of the stopping target, so again, this result makes sense as the vertices in this section of the distribution generally refer to the decay products of surface muons. The second peak refers to vertices of the decay products of the surface muons. It should also peak approximately where it is because in this simulation, the width and length of the beam are $7.8mm$ and $9.1mm$ respectively, therefore there is a lower chance of muons being stopped outside of this range.

2.4.17 mc_vt

mc_vt refers to the time of the origin vertex.

2.5 Frames Tree

2.5.1 Overview

Each entry contains vectors of segments/tracks with a given event ID. This effectively categorises the segments rather than describes physical qualities as happened in the segs tree. More simply, this will collect the segments into frames as opposed to just listing all the events.

2.5.2 runId

This again refers to the random seed used for the run of the simulation.

2.5.3 eventId

This again refers to the frame of the simulation, not the track specifically.

2.5.4 weight

This is the weight associated with the event, generally this is preset by the generator. This is probably adapted by mu3eSim as opposed to mu3eTrirec, but that is worth checking if it becomes relevant.

2.5.5 n

As opposed to the segs tree wherein n referred to the number of triplets used to reconstruct the track, this time n refers to the number of segments found in each event/frame. This is then further broken down into the number of 3, 4, 6 and 8 hit segments found in a frame. In the n histogram (shown by figure 16), this details the number of segments found in each frame and groups the frames where the same number of segments found together. So the number of frames where 2 segments are found are listed and so on for instance.

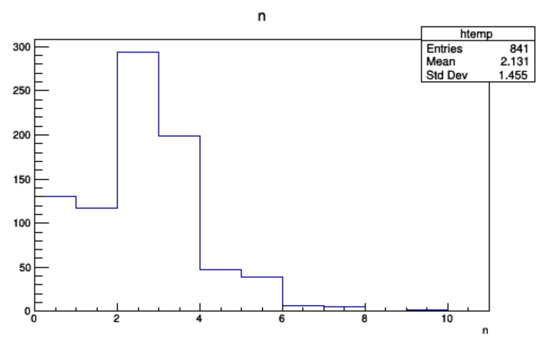


Figure 16: Histogram of n. This shows the number of frames with n referring to the number of segments found in them, irrespective of length.

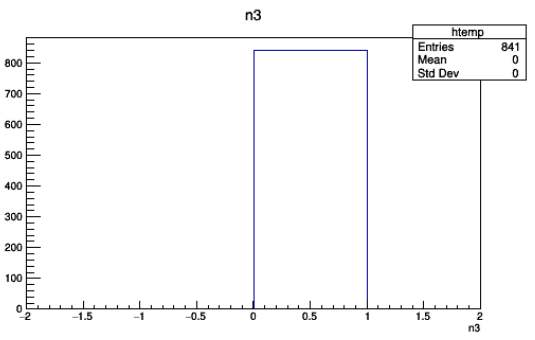


Figure 17: Histogram of n3. This shows the number of frames where a 3 hit segment was found. Excluding empty frames, this can only be a single entry into the histogram because all segments are made up of triplets and therefore every frame is going to contain a 3 hit segment.

2.5.6 n3

n3 refers to the number of segments that were reconstructed with three hits or rather one triplet. Physically, this will most likely be a particle exiting the central station. An example of this is shown in figure 17.

2.5.7 n4

n4 again refers to the number of segments that were reconstructed with four hits or rather two triplets. Again this represents just a particle exiting the central station. As described in the example figures, the numbered histograms show the number of frames where a number of 4 hit segments were found.

2.5.8 n6

The number of hits is now self-explanatory and the number of triplets is three. This refers to a particle exiting the central station and entering one of the recurl stations.

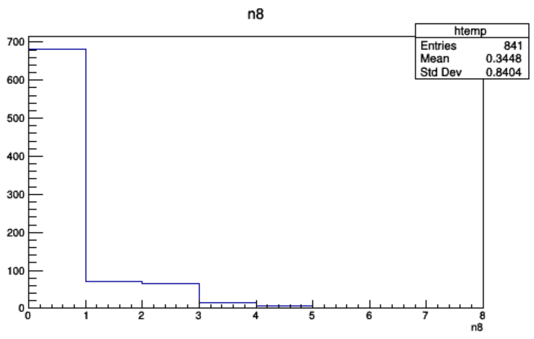


Figure 18: This shows the frames in which an 8 hit segment was found, where the number refers to the number of segments that were found in a frame. So the histogram entry corresponding to 3 refers to the number of frames where 3, 8 hit segments were found.

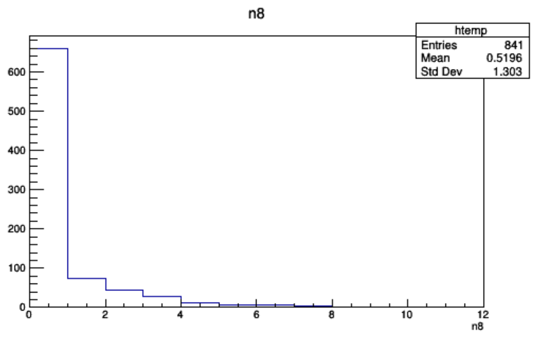


Figure 19: This is the truth information for the n8 histogram.

2.5.9 n8

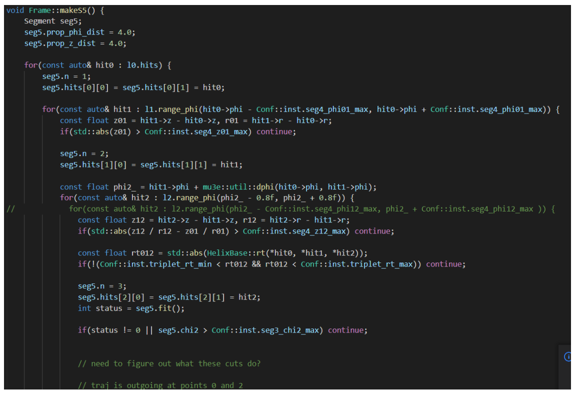
This represents a particle exiting and entering the central station.

3 mu3eTrirec - Efficiency Branch

3.1 Overview

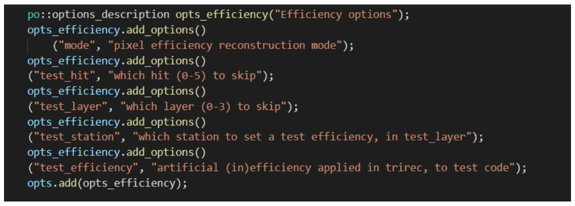
3.2 To Add

ein eout etc



```
void Frame::makeS5() {
    Segment seg5;
    seg5.prop_phi_dist = 4.0;
    seg5.prop_z_dist = 4.0;

    for(const auto& hit0 : l0.hits) {
        seg5.n = 1;
        seg5.hits[0][0] = seg5.hits[0][1] = hit0;

        for(const auto& hit1 : l1.range_phi(hit0->phi - Conf::inst.seg4_phi01_max, hit0->phi + Conf::inst.seg4_phi01_max)) {
            const float z01 = hit1->z - hit0->z, r01 = hit1->r - hit0->r;
            if(std::abs(z01) > Conf::inst.seg4_z01_max) continue;

            seg5.n = 2;
            seg5.hits[1][0] = seg5.hits[1][1] = hit1;

            const float phi2_ = hit1->phi + mu3e::util::dphi(hit0->phi, hit1->phi);
            for(const auto& hit2 : l2.range_phi(phi2_ - 0.8f, phi2_ + 0.8f)) {
//            for(const auto& hit2 : l2.range_phi(phi2_ - Conf::inst.seg4_phi12_max, phi2_ + Conf::inst.seg4_phi12_max )) {
                const float z12 = hit2->z - hit1->z, r12 = hit2->r - hit1->r;
                if(std::abs(z12 / r12 - z01 / r01) > Conf::inst.seg4_z12_max) continue;

                const float rt012 = std::abs(HelixBase::rt(*hit0, *hit1, *hit2));
                if(!(Conf::inst.triplet_rt_min < rt012 && rt012 < Conf::inst.triplet_rt_max)) continue;

                seg5.n = 3;
                seg5.hits[2][0] = seg5.hits[2][1] = hit2;
                int status = seg5.fit();

                if(status != 0 || seg5.chi2 > Conf::inst.seg3_chi2_max) continue;


                // need to figure out what these cuts do?

                // traj is outgoing at points 0 and 2
```

Figure 20: S5 code to go add into the document.



```
    po::options_description opts_efficiency("Efficiency options");
    opts_efficiency.add_options()
        ("mode", "pixel efficiency reconstruction mode");
    opts_efficiency.add_options()
    ("test_hit", "which hit (0-5) to skip");
    opts_efficiency.add_options()
    ("test_layer", "which layer (0-3) to skip");
    opts_efficiency.add_options()
    ("test_station", "which station to set a test efficiency, in test_layer");
    opts_efficiency.add_options()
    ("test_efficiency", "artificial (in)efficiency applied in trirec, to test code");
    opts.add(opts_efficiency);
```

Figure 21: Efficiency code to go add into the document.

3.3 Segs Tree

3.3.1

4 mu3eEfficiency

4.1 Overview

5 Tables

5.1 Numbering

Table 5.1 shows the digits used to reference particles and their origins in the software.

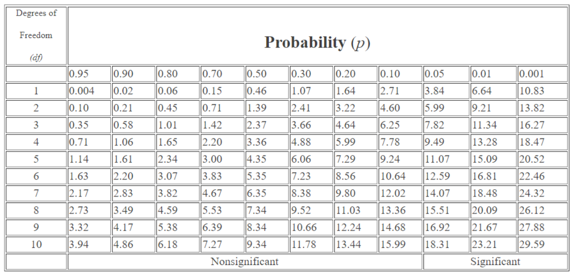
First Digit	Particle Origin	Second Digit	Particle
0	Primary Particle (From particle gun/generator)	0	Photon
1	Michel Decay	1	Positron
2	Radiative Muon Decay	2	Electron
3	Internal Conversion Decay	3	Mu-
4	Photon Conversion	4	Mu+
5	Bhabha Scattering	5	Pi+
6	Bremsstrahlung	6	Pi-
7	Positron Annihilation	7	Neutrino
8	Compton Scattering/Photoelectric Effect	8	Geantino
9	Mu→3e	9	Reserved
10	Mu→eX		Note: Negative sign refers to unidentified/invalid/combinatorial etc

5.2 Helix Parameters

Table 5.2 shows the parameters that are used when describing the helix.

Name	Parameter	Function
kappa	κ	Describes the curvature of the helix.
tan	$\tan(\lambda)$	Describes the distance between turns of the helix.
phi	ϕ	Transverse angle of the helix, in the $x - y$ plane.
dca	dca	Offset of the centre of the helix from the z axis.
z0	$z0$	Origin of the helix in the z axis.

5.3 Chi Squared versus p-value



Degrees of Freedom *(df)*	Probability (p)										
	0.95	0.90	0.80	0.70	0.50	0.30	0.20	0.10	0.05	0.01	0.001
1	0.004	0.02	0.06	0.15	0.46	1.07	1.64	2.71	3.84	6.64	10.83
2	0.10	0.21	0.45	0.71	1.39	2.41	3.22	4.60	5.99	9.21	13.82
3	0.35	0.58	1.01	1.42	2.37	3.66	4.64	6.25	7.82	11.34	16.27
4	0.71	1.06	1.65	2.20	3.36	4.88	5.99	7.78	9.49	13.28	18.47
5	1.14	1.61	2.34	3.00	4.35	6.06	7.29	9.24	11.07	15.09	20.52
6	1.63	2.20	3.07	3.83	5.35	7.23	8.56	10.64	12.59	16.81	22.46
7	2.17	2.83	3.82	4.67	6.35	8.38	9.80	12.02	14.07	18.48	24.32
8	2.73	3.49	4.59	5.53	7.34	9.52	11.03	13.36	15.51	20.09	26.12
9	3.32	4.17	5.38	6.39	8.34	10.66	12.24	14.68	16.92	21.67	27.88
10	3.94	4.86	6.18	7.27	9.34	11.78	13.44	15.99	18.31	23.21	29.59
	Nonsignificant								Significant		

Figure 22: Table of χ^2 values as a function of degrees of freedom and p-value or rather significance.

5.4 Conversion Equations

Table 5.4 shows the conversion between Cartesian coordinates and the coordinate system that best helps describe a helix.

Parameter	Conversion	Comments
rt	$rt = \sqrt{x^2 + y^2}$	The distance to the point from the z-axis or rather in the $x - y$ plane.
r	$r = \sqrt{x^2 + y^2 + z^2}$	The 3D radius, that being the distance to a point from the origin.
phi	$\phi = \arctan(\frac{y}{x})$	Angle between the vector in the $x - y$ plane.
lam	$\lambda = \arctan(\frac{z}{rt})$	Offset of the centre of the helix from the z axis.

6 Code

6.1 Write about

Segment.h Segment.cpp efficiency.cpp Frame.cpp (Both versions) Frame.h (Both versions) Root.cpp trirec.cpp (Not too important for now) HelixBase.h TripletBase.h

6.2 float.hpp/double.hpp

6.2.1 Overview

These pieces of code focus on the building blocks of the vectors involved. They initially define the vectors that are to be used, either defining a 2 or 3 dimensional vector. Both are used to define what to do with a vector if it is in a double or float data type. The float.hpp file has the most information about what to do. There are initial definitions of what the vectors are, i.e. a function to collect the three coordinates, these are completed in the make_float functions. There are then functions to convert the coordinates into a more useful coordinate system for the helix. These are completed in the float2 or float3 functions (there are also 4-vectors). Finally operators are defined, such that when two vectors are added or subtracted etc, this can be done.

6.2.2 Conversions

The initial Cartesian inputs are taken with the make_float function. Then using float2 or float3 (depending if it is a 2D or 3D vector), a series of conversions is completed in order to put them into a form more useful for the geometry of the helix trajectories of the particles. The equations that relate these conversions can be found in table 5.4.

6.2.3 Vector Operations

The code also defines operators that can be used on these defined vectors. These operators are written the same as could be used on regular variables, but are redefined for the sake of vector operations. For instance + and += have the same function, but are used for vector operations. Multiplication and division are slightly different, they multiply a vector with a scalar, not another vector. Finally, the dot and cross products of two vectors are defined.

6.3 HelixBase.h

6.3.1 Overview

The purpose of this code is to generate helices from the hits that are detected. This piece of code provides the mathematical foundation for the helices in the form of functions. Initially the code generates helices from two connected hits in two separate layers. It does this by selecting the two hits, then using known parameters, generates a helix that would incorporate these two hits. A description of these processes is provided in detail later. The next thing it does is start to relate two helices together to generate a triplet. It should be noted that this code is not used for the selection of related hits, but simply accepts two hits that the selection process has related and generates helices and then triplets.

6.3.2 Helix Generation

The generation of the helices is done by taking two consecutive hits in the layers and uses known parameters to gauge an estimation of the helix that could be generated by a particle that caused these hits. Initially the code takes in two float3 variables (as found in the float.hpp code 6.2.1). These float3 variables correspond to 2 hits that are found in successive layers.

Distance and angle between hits - This part of the code takes the two hits, specifically their x and y components and takes the difference and the angle between the two. The difference is done with the operators that are mentioned in the float.hpp section 6.2.1. The angle that is taken is the ϕ angle between the hits, which is the angle in the $x-y$ plane. It is taken with the condition that the distance found is greater than the value FLT_EPSILON. If the value is less than this value, then the angle is found to be 0. Anything else is found using the function std::copysign. This takes the magnitude of the first value and applies the sign of the second value. In this instance the first value is the angle and the second value is the y-component, y01.

- **x01** - Difference between x-coordinate in hit 0 and 1 $\rightarrow x01 = h1.x - h0.x$
- **y01** - Difference between y-coordinate in hit 0 and 1 $\rightarrow y01 = h1.y - h0.y$
- **d01** - Distance between hits 0 and 1 in $x-y$ plane, effectively the transverse radius $\rightarrow d01 = \sqrt{x01^2 + y01^2}$
- **d01_2** - Half distance between hits 0 and 1 in $x-y$ plane $\rightarrow d01\_2 = \frac{d01}{2}$
- **FLT_EPSILON** - Not a piece of code that was written for this purpose but is worth mentioning. This is the difference between 1 and the smallest floating point number of type float that is greater than 1. Effectively, a very small number.
- **phi01x** - The angle between the x01 direction and the d01 direction. The sign of this angle is positive if the angle is in the positive y-direction and negative in the opposite case $\rightarrow phi01x = \arccos\frac{x01}{d01}$

Fitting triplets - Just listing various angle definitions to clarify knowledge. These definitions are focused on relating the two hit pairs.

- **x02** - The x-component of the difference between the the first and third hit of the triplet $\rightarrow x02 = h2.x - h0.x$
- **y02** - The y-component of the difference between the the first and third hit of the triplet $\rightarrow y02 = h2.y - h0.y$
- **d02** - The distance between the first and third hit of the triplet in the $x-y$ plane $\rightarrow d02 = \sqrt{x02^2 + y02^2}$
- **rt** - This is the centre of the circle that is in the $x-y$ plane, or rather the radius of the tangential component $\rightarrow rt = \frac{d01 \cdot d12 \cdot d02}{0.5((x01 \cdot y12) - (x12 \cdot y01))}$
- **dphi** - This is the difference in the angle ϕ between the two helices (h01 and h12) given by the pair of hits, i.e. the h01.phi01x value. This uses a function that is defined in the utilities/math.hpp file, but this function is there to generate an error message if an angle greater than π is calculated, other than that the equation is exactly what would be expected $\rightarrow dphi = h01.phi01x - h12.phi01x$
- **kt** - This is the curvature of the triplet in the $x-y$ plane, formed by the two helices (h01 and h12). Whereby $kt = \frac{1}{rt}$. This is calculated using the cross product function written in the float.hpp piece (6.2.1). The equation used is as follows $\rightarrow kt = \frac{((h01.h1 - h01.h0) \times (h12.h1 - h12.h0)).z}{4 * h01.d01_2 * h12.d01_2 * d02_2}$. This equation takes the z component of the cross product of the vectors that define the pairs of hits in each helix. It then divides by the magnitude of these vectors in order to just get the curvature. The alternative version of writing this equation shows an equation more closely related to the general formula for the cross product of two vectors $\rightarrow kt = \frac{\sin dphi}{d02_2}$. As seen here, this is much closer to the standard equation for the cross product of the vectors

- **phi01c_2** - This represents the angle ϕ at the centre in between the two hits in the helix. This is the local angle ϕ though, not the angle relative to the origin of the coordinate system (the stopping target), but the angle relative to the two hits in the pair. It should be noted that these values can be referred to the value dphi $\rightarrow dphi = phi01c\_2 + phi12c\_2 = \frac{phi01c + phi12c}{2}$. The angle is calculated with the following equation $\rightarrow h01.phi01c\_2 = \arcsin(h01.d01\_2 \cdot kt)$. The other angle can be calculated with the equation $\rightarrow h12.phi01c\_2 = dphi - h01.phi01c\_2$

- **z01_2** - The halfway point between the two z coordinates of the hits in a pair. The equation is given by $\rightarrow z01\_2 = \frac{h1.z - h0.z}{2}$

- **tanlam01c** - This represents the angle pointing in the direction of the z-axis from the $x-y$ plane. It should be remembered that $kt = \frac{1}{rt}$.The equation is given by $\rightarrow \tan(lam01c) = \frac{z01_2 \cdot kt01}{phi01c_2}$

- **cos2lam01c** - Cos2 is $\cos^2$. It should be repeated that these angles are referring to a pair of hits (not a triplet), hence the '01' suffix. It also the angle that is found centred between the hits, hence the suffix 'c'. The derivation for this equation is found with simple trig identities of $\tan x = \frac{\sin x}{\cos x}$. The equation used in the code is given by $\rightarrow cos2lam01c = \frac{1}{1 + tanlam01c^2}$

- **delta01** - This is ultimately going to be used as part of the correction factor that turns the local angles of the helix into global angles (as in relative to the origin, not the helix). This particular calculation uses the xtanx function defined in the math.hpp file if the value of phi01c_2 is less than 1. The equation for delta01 is given by $\rightarrow delta01 = \frac{1 - phi01c\_2}{\tan(phi01c\_2)}$. This is then multiplied with the cos2lam01c value.

- **k01c** - This is the 3D curvature, not just the transverse curvature. This is centered in between the two hits. Whilst the transverse curvature is constant, the k01c value is dependant on the angle from which it is measured. The equation is given by $\rightarrow k01c = \cos(lam01c) \cdot kt01$

- **alpha01/alpha01_2** - This is the correction factor that is going to convert local ϕ values to global ones. The equation for both alpha01 and alpha01_2 is given by the rate of change of ϕ with respect to k (the general curvature of the helix, not just the transverse curvature). The equations are given by $\rightarrow alpha01 = \frac{1}{1 - delta01}$ and $alpha01\_2 = alpha01 \cdot \frac{phi01c\_2}{k01c}$

- **beta01** - Like alpha01, this is a correction factor that converts from local angles to global ones. In this instance, β changes the λ angle. This is found with the rate of change of λ with respect to the radius of the helix. The equation is given by $\rightarrow beta01 = \frac{delta01}{1 - delta01} \cdot \frac{tanlam01c}{-k01c}$

- **phi01_2** - FINALLY, we can find the global angles relevant to the helix. This means we can now describe the helix relative to the origin of the coordinate system. For ϕ, we find that the conversion is given by $\rightarrow phi01\_2 = phi01c\_2 + ((k - k01c) \cdot alpha01\_2)$

- **lam01** - We can do the same for the λ angle with the following equation $\rightarrow lam01 = lam01c + ((k - k01c) \cdot beta01)$

- **tan0** - The tangent to the circle in the $x - y$ plane at hit 0. This is given by $\rightarrow lam0 = phi01x - phi01\_2$

- **tan1** - The tangent to the circle in the $x - y$ plane at hit 1. This is given by $\rightarrow lam1 = phi01x + phi01\_2$

Everything else in the code that isn't listed, is relatively self-explanatory and can be easily derived from the items described above. The only other function that should be mentioned is pca. This refers to the point of closest approach, and puts the helix parameters relative to a point v. This will therefore be used to assess helices that originate at a decay vertex not centered at the origin of the coordinates.

6.4 TripletBase.h

6.4.1 Overview

The aim is to build up a triplet based on the hit pairs. Part of this fitting process is done in the HelixBase.h file. In that file, code is written such that two hit pairs, with associated helix parameters are collected constructed into a triplet and an initial fit is checked (Whereby an error is thrown if d01 or d12 or d02 is less than 1mm or dphi is less than 1mrad). It also builds a 3D curvature which minimises the χ^2 of the triplet. The result of this fit is such that a triplet is generated and has a series of parameters associated with it like a curvature and error. It also includes all the parameters that were contained in the hit pairs.

6.5 SegmentBase.h

6.5.1 Overview

This code does an initial fit on a segment. It constructs segments and at each step uses the fit_triplet from the HelixBase.h file in order to check that it falls within the requirements of a triplet and assigns new parameters as a segments such as weights and a χ^2 value. Importantly, this doesn't take into account any physical variables such as energy loss, pixel positions or any material knowledge.

6.6 segment.h/.cpp

6.6.1 Overview

This is a more in depth look at the segments, the header file just calls the functions but the .cpp file actually writes the functions. There is not much to write about this as this is a well commented piece of code. If there is more time, I will add to this document a run through of the code, for now I'll just add the relevant pieces for me. The main focus of this code is to add in various pieces to the segment that are not covered by the simple initial base. These include physical corrections, such as materials and scattering effects.

The code initially uses helix base to build up segments. In conjunction with this, it introduces variables for monitoring the propagation and where that is taking the track. These are going to be important when looking at ϕ and z windows. For the side of the efficiency, variables such as hit_found and hits_in_layer are introduced. These are used in order to give diagnostics on the segment. The final point of interest is the ns, nh variables. ns represents the number of shared segments or rather the number of segments that intersect this segment. nh represents the number of shared hits, the number of hits in a segment that are shared by other segments. nsg is the same as ns but in a more general sense, it is the number of segments in a cluster of segments, that this segment is connected to. And nhg is the number of hits in the group.

7 Install and Compilation

7.1 Things to remember

- source /cvmfs/sft.cern.ch/lcg/views/LCG_102/x86_64-centos7-gcc11-opt/setup.sh
- cmake
- make
- make clean
- make install

7.2 Event display