a Tool based Reconstruction Algorithm for Characterising Showers (TRACS)

Dom Barker, Ed Tyley, Dom Brailsford
on Behalf of the SBN Shower Reconstruction Group

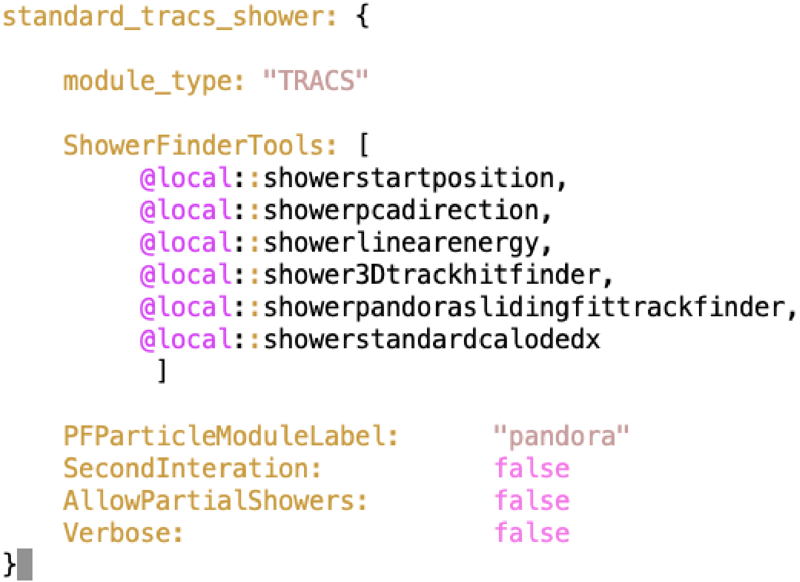
University of Sheffield

August 13, 2019

Introduction

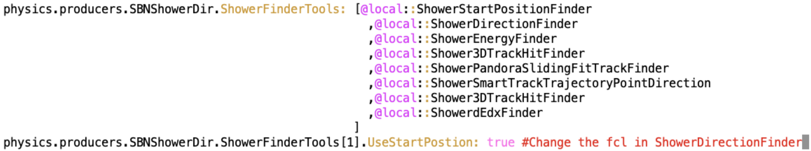
- The SBN team has created a new shower reconstruction module.
- First I'll go through the framework then I'll go over the physics.
- The module characterises showers-like pfparticles that come from the Pandora reconstruction
- The idea behind the module is to split the reconstruction into art-tools with each calculated one (or more) of the recob:showers characteristics (start position, direction, Energy, dEdx).
- The user is able to hot-swap tools and add new tools in an easy art-fcl way.
- We have created some base tools (some of which took inspiration from existing modules. Thanks) in order for users to run out the box and are currently developing many more. These are written in a detector agnostic way.
- The framework has been tested in SBND and ICARUS successfully and we have interest from analyzers in the DUNE far detector.
- We are hoping to get the framework and some tools into larreco.
- See all the work on DomBarker/Shower_Branch

standard fcl Parameter Set



```
standard_tracs_shower: {

    module_type: "TRACS"

    ShowerFinderTools: [
         @local::showerstartposition,
         @local::showerpcadirection,
         @local::showerlinearenergy,
         @local::shower3Dtrackhitfinder,
         @local::showerpandoraslidingfittrackfinder,
         @local::showerstandardcalodedx
          ]

    PFParticleModuleLabel:      "pandora"
    SecondInteration:           false
    AllowPartialShowers:        false
    Verbose:                    false
}
```

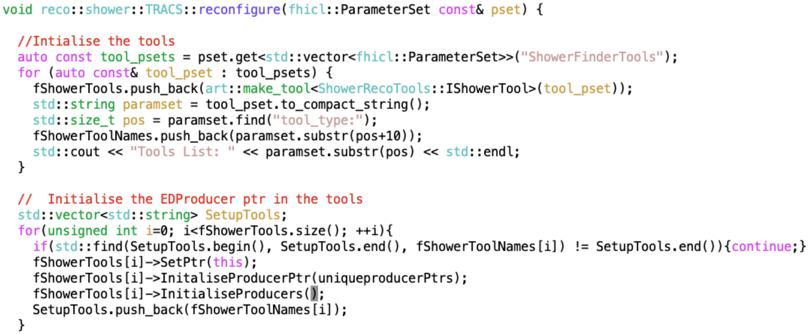
Advanced fcl Parameter Set



```
physics.producers.SBNShowerDir.ShowerFinderTools: [@local::ShowerStartPositionFinder
                                                    ,@local::ShowerDirectionFinder
                                                    ,@local::ShowerEnergyFinder
                                                    ,@local::Shower3DTrackHitFinder
                                                    ,@local::ShowerPandoraSlidingFitTrackFinder
                                                    ,@local::ShowerSmartTrackTrajectoryPointDirection
                                                    ,@local::Shower3DTrackHitFinder
                                                    ,@local::ShowerdEdxFinder
                                                  ]
physics.producers.SBNShowerDir.ShowerFinderTools[1].UseStartPostion: true #Change the fcl in ShowerDirectionFinder
```

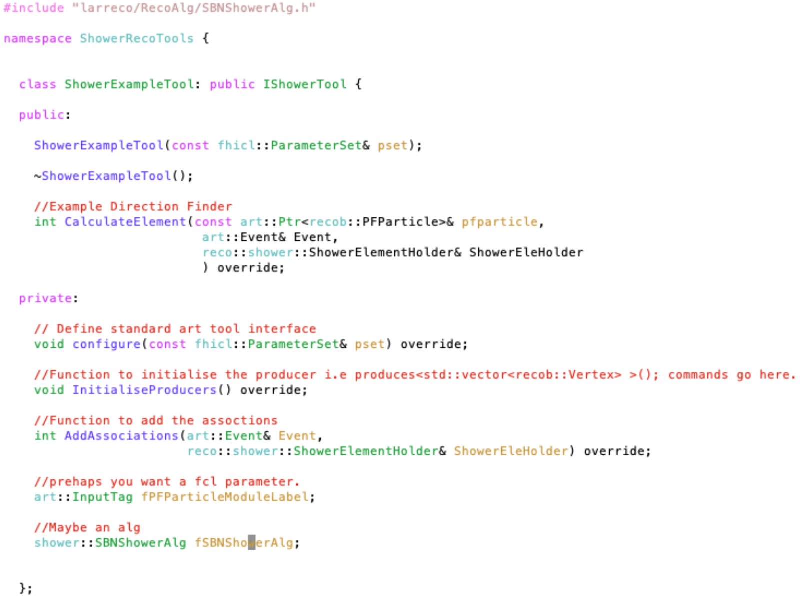
TRACS Module - Initialise the Tools

- The module initialises a set of tools best on a parameter list given.
- Each too inherits from a IShowerTool.h base class. The user can override three of the base class functions: Configure, CalculateElement and AddAssociations



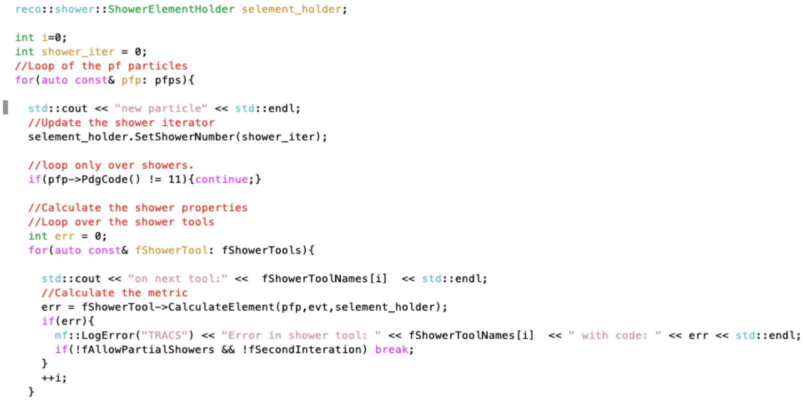
```
void reco::shower::TRACS::reconfigure(fhicl::ParameterSet const& pset) {

  //Intialise the tools
  auto const tool_psets = pset.get<std::vector<fhicl::ParameterSet>>("ShowerFinderTools");
  for (auto const& tool_pset : tool_psets) {
    fShowerTools.push_back(art::make_tool<ShowerRecoTools::IShowerTool>(tool_pset));
    std::string paramset = tool_pset.to_compact_string();
    std::size_t pos = paramset.find("tool_type:");
    fShowerToolNames.push_back(paramset.substr(pos+10));
    std::cout << "Tools List: " << paramset.substr(pos) << std::endl;
  }

  //  Initialise the EDProducer ptr in the tools
  std::vector<std::string> SetupTools;
  for(unsigned int i=0; i<fShowerTools.size(); ++i){
    if(std::find(SetupTools.begin(), SetupTools.end(), fShowerToolNames[i]) != SetupTools.end()){continue;}
    fShowerTools[i]->SetPtr(this);
    fShowerTools[i]->InitaliseProducerPtr(uniqueproducerPtrs);
    fShowerTools[i]->InitialiseProducers();
    SetupTools.push_back(fShowerToolNames[i]);
  }
```

Example Tool: Derived tool class of IShowerTool



```
#include "larreco/RecoAlg/SBNShowerAlg.h"

namespace ShowerRecoTools {


  class ShowerExampleTool: public IShowerTool {

  public:

    ShowerExampleTool(const fhicl::ParameterSet& pset);

    ~ShowerExampleTool();

    //Example Direction Finder
    int CalculateElement(const art::Ptr<recob::PFParticle>& pfparticle,
                          art::Event& Event,
                          reco::shower::ShowerElementHolder& ShowerEleHolder
                          ) override;

  private:

    // Define standard art tool interface
    void configure(const fhicl::ParameterSet& pset) override;

    //Function to initialise the producer i.e produces<std::vector<recob::Vertex> >(); commands go here.
    void InitialiseProducers() override;

    //Function to add the assoctions
    int AddAssociations(art::Event& Event,
                        reco::shower::ShowerElementHolder& ShowerEleHolder) override;

    //prehaps you want a fcl parameter.
    art::InputTag fPFParticleModuleLabel;

    //Maybe an alg
    shower::SBNShowerAlg fSBNShowerAlg;


  };
```

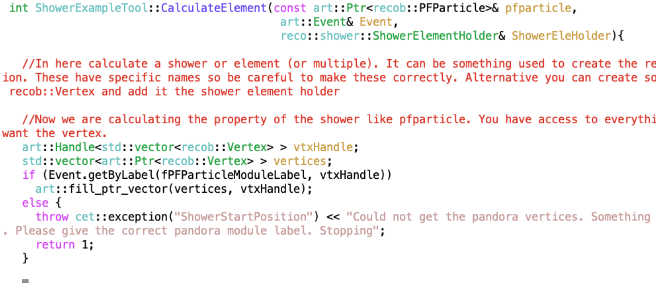
TRACS Module - loop over the pfpparticles



```
reco::shower::ShowerElementHolder selement_holder;

int i=0;
int shower_iter = 0;
//Loop of the pf particles
for(auto const& pfp: pfps){

  std::cout << "new particle" << std::endl;
  //Update the shower iterator
  selement_holder.SetShowerNumber(shower_iter);

  //loop only over showers.
  if(pfp->PdgCode() != 11){continue;}

  //Calculate the shower properties
  //Loop over the shower tools
  int err = 0;
  for(auto const& fShowerTool: fShowerTools){

    std::cout << "on next tool:" <<  fShowerToolNames[i]  << std::endl;
    //Calculate the metric
    err = fShowerTool->CalculateElement(pfp,evt,selement_holder);
    if(err){
      mf::LogError("TRACS") << "Error in shower tool: " << fShowerToolNames[i]  << " with code: " << err << std::endl;
      if(!fAllowPartialShowers && !fSecondInteration) break;
    }
    ++i;
  }
```

ExampleTool - CalculateElement

- CalculateElement() is where the user does the physics. It is the equivalent of "produce" for each shower. You do work only on the one shower in question.
- CalculateElement() is given the art::Event and a object called ShowerElementHolder so that the users can do access previously calculated data products and properties of the shower.
- The CalculateElement() function returns an int to stop the chain continuing

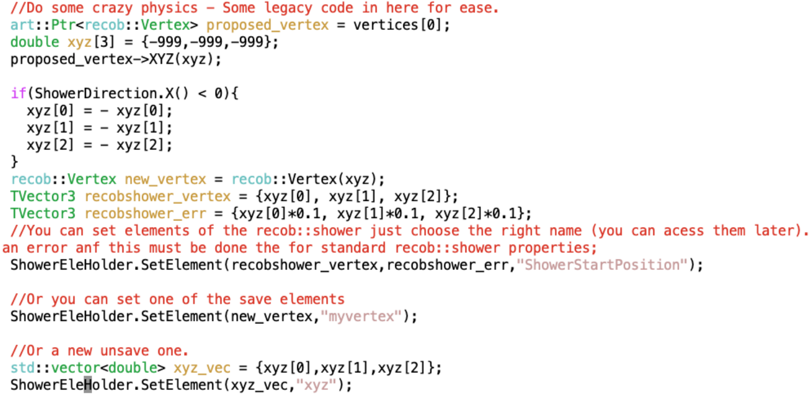


```
int ShowerExampleTool::CalculateElement(const art::Ptr<recob::PFParticle>& pfparticle,
                                        art::Event& Event,
                                        reco::shower::ShowerElementHolder& ShowerEleHolder){

  //In here calculate a shower or element (or multiple). It can be something used to create the re
ion. These have specific names so be careful to make these correctly. Alternative you can create so
 recob::Vertex and add it the shower element holder

  //Now we are calculating the property of the shower like pfparticle. You have access to everythi
want the vertex.
  art::Handle<std::vector<recob::Vertex> > vtxHandle;
  std::vector<art::Ptr<recob::Vertex> > vertices;
  if (Event.getByLabel(fPFParticleModuleLabel, vtxHandle))
    art::fill_ptr_vector(vertices, vtxHandle);
  else {
    throw cet::exception("ShowerStartPosition") << "Could not get the pandora vertices. Something
'. Please give the correct pandora module label. Stopping";
    return 1;
  }
```

Using the Shower Element Holder

- The ShowerElementHolder holds data products and shower properties created in the tools.
- The class structure allows for any object to be held (except ptrs and arrays (the object have to exist passed the CalculateElement stage)) (Template Inheritance).
- The holder has Get/Set/Check functions for the user to access previously calculated properties and data products. Elements are accessed via a unique std::string name.
- The shower properties that go into making a recob::shower have specific names and must be filled by the tools for a recob::shower to be made.
- There is a distinction to what we call a properties and a data-product. Properties have to be set with error.
- Elements that are stored in the holder can also be placed into the event. Only data-products can be created by the producer modules.
- Checks are performed on the shower properties which make the recob::shower and the data products that are to be stored in the event. It is up to the user to initialise the their own data products.

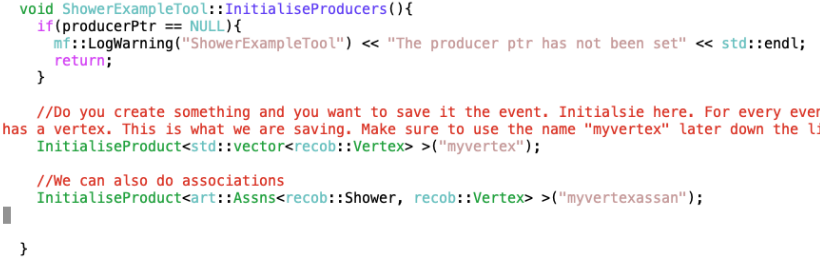
Example Tool - Using the Shower Element Holder



```
 //Do some crazy physics - Some legacy code in here for ease.
 art::Ptr<recob::Vertex> proposed_vertex = vertices[0];
 double xyz[3] = {-999,-999,-999};
 proposed_vertex->XYZ(xyz);

 if(ShowerDirection.X() < 0){
   xyz[0] = - xyz[0];
   xyz[1] = - xyz[1];
   xyz[2] = - xyz[2];
 }
 recob::Vertex new_vertex = recob::Vertex(xyz);
 TVector3 recobshower_vertex = {xyz[0], xyz[1], xyz[2]};
 TVector3 recobshower_err = {xyz[0]*0.1, xyz[1]*0.1, xyz[2]*0.1};
 //You can set elements of the recob::shower just choose the right name (you can acess them later).
an error anf this must be done the for standard recob::shower properties;
 ShowerEleHolder.SetElement(recobshower_vertex,recobshower_err,"ShowerStartPosition");

 //Or you can set one of the save elements
 ShowerEleHolder.SetElement(new_vertex,"myvertex");

 //Or a new unsave one.
 std::vector<double> xyz_vec = {xyz[0],xyz[1],xyz[2]};
 ShowerEleHolder.SetElement(xyz_vec,"xyz");
```

Adding Data Products and Associations to the Event

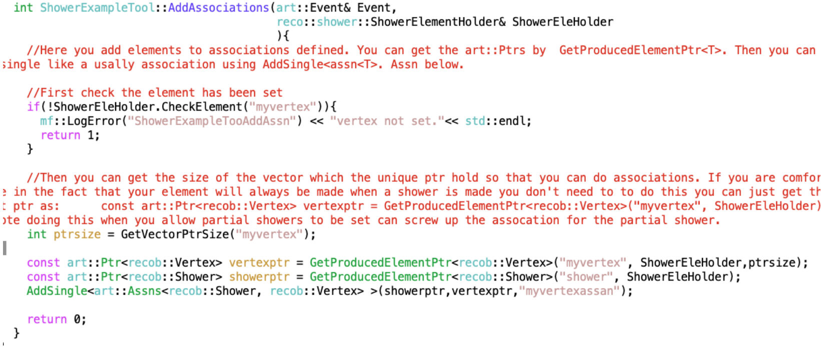
- The user can also create data products and associations in the tools a la a producer module.
- A holder maintains the unique ptrs and art PtrMakers that will be added to event.
- All the work is done "behind the scenes" leaving the user with a few functions with that are analogous to the corresponding art functions.



```
  void ShowerExampleTool::InitialiseProducers(){
    if(producerPtr == NULL){
      mf::LogWarning("ShowerExampleTool") << "The producer ptr has not been set" << std::endl;
      return;
    }

    //Do you create something and you want to save it the event. Initialsie here. For every ever
has a vertex. This is what we are saving. Make sure to use the name "myvertex" later down the li
    InitialiseProduct<std::vector<recob::Vertex> >("myvertex");

    //We can also do associations
    InitialiseProduct<art::Assns<recob::Shower, recob::Vertex> >("myvertexassan");


  }
```

Making Associations

- Once an association has been created the user can add to in AddAssociations function.
- This is ran at the end of the pfparticle loop.
- The user can access the art::Ptrs via the GetProducedElementPtr function.
- Users can use the addSingle(T A, T2 B) function add the association



```
  int ShowerExampleTool::AddAssociations(art::Event& Event,
                                          reco::shower::ShowerElementHolder& ShowerEleHolder
                                          ){
    //Here you add elements to associations defined. You can get the art::Ptrs by  GetProducedElementPtr<T>. Then you can
single like a usally association using AddSingle<assn<T>. Assn below.

    //First check the element has been set
    if(!ShowerEleHolder.CheckElement("myvertex")){
      mf::LogError("ShowerExampleTooAddAssn") << "vertex not set."<< std::endl;
      return 1;
    }

    //Then you can get the size of the vector which the unique ptr hold so that you can do associations. If you are comfor
e in the fact that your element will always be made when a shower is made you don't need to to do this you can just get th
t ptr as:      const art::Ptr<recob::Vertex> vertexptr = GetProducedElementPtr<recob::Vertex>("myvertex", ShowerEleHolder)
ote doing this when you allow partial showers to be set can screw up the assocation for the partial shower.
    int ptrsize = GetVectorPtrSize("myvertex");

    const art::Ptr<recob::Vertex> vertexptr = GetProducedElementPtr<recob::Vertex>("myvertex", ShowerEleHolder,ptrsize);
    const art::Ptr<recob::Shower> showerptr = GetProducedElementPtr<recob::Shower>("shower", ShowerEleHolder);
    AddSingle<art::Assns<recob::Shower, recob::Vertex> >(showerptr,vertexptr,"myvertexassan");

    return 0;
  }
```

Base Tools

- We suggest that along with the framework a set of base tools are also added to larreco.
- Below we describe the tools suggested for base tool list, but we also want to add more.
- Caveat: The PandoraSlidingFitTrackFinder requires larreco to have a larpandora depenacy.
- The calcuations in the tools have been heavily based on existing shower reconstruction module PandoraShowerCreation and EMShower. Both of which are compared to TRACS.
- The comparison is made with a analyses module called ShowerValidation which attempts to match truth based showers to recob::Showers and reconstructed properties to truth.
- The Validation module heavily relies on the backtracker in the form of functions from Utilitiy packages (https://indico.fnal.gov/event/19364/)
- Both SBND and ICARUS are using the validation module and if wanted this can come as part of the package.
- Modules compared using a particle gun isotropic sample of single electron and charged pion both created at a vertex.

Current Tools List

Red is new tools that we suggest should go in. Orange is tools we would want in soon. Blue is the tools we want as the based running tools and will be discussed in this section.

ShowerStartPosition_tool.cc

ShowerPCADirection_tool.cc
ShowerTrackTrajectoryPointDirection_tool.cc
ShowerSmartTrackTrajectoryPointDirection_tool.cc
ShowerTrackSpacePointDirection_tool.cc
ShowerTrackHitDirection_tool.cc
ShowerTrackDirection_tool.cc
ShowerTrackPCADirection_tool.cc

Shower2DLinearRegressionTrackHitFinder_tool.cc
Shower3DTrackHitFinder_tool.cc

ShowerPMATrackFinder_tool.cc
ShowerPandoraSlidingFitTrackFinder_tool.cc

ShowerLinearEnergy_tool.cc

ShowerStandardCalodEdx_tool.cc
ShowerSlidingStandardCalodEdx_tool.cc

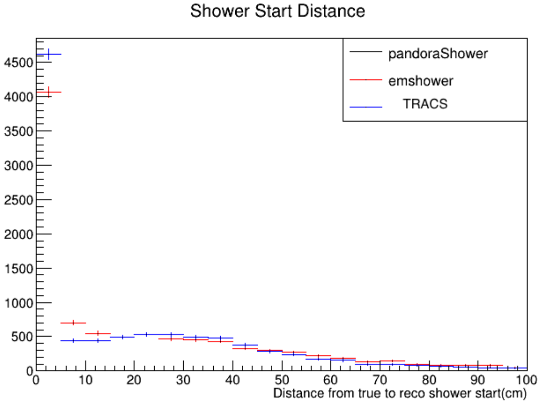
ShowerTrackTrajToSpacepoint_tool.cc

ShowerExampleTool_tool.cc
ShowerGenericTool_tool.cc

ShowerDirectionCheater_tool.cc
ShowerStartPositionCheater_tool.cc
ShowerTrackFinderCheater_tool.cc

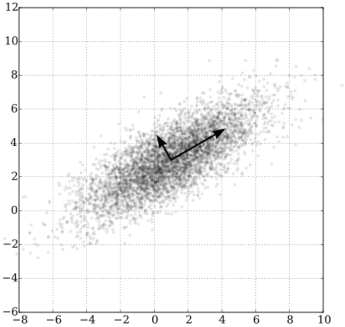
Shower Start Position

- Get the vertex associated with the PFParticle
- Exactly the same as Pandora Shower

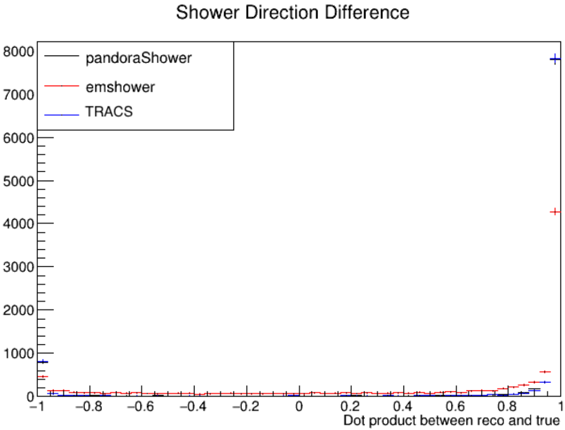


Shower Direction

- Performs a principal component analysis to find the principle eigenvector
- Always take the direction from the start position to the centre of the shower
- This is the same method as used in Pandora Shower but with charge weighting (option set in fcl)



Shower Direction



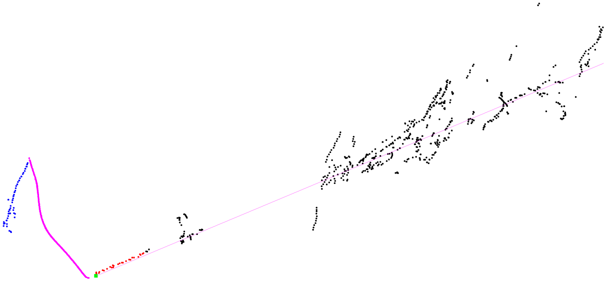
Initial Track Hit Finders

- 2D Method from EMShower has been ported:
  - Project 3D direction into wire-tick space
  - Order hits based on projection along direction
  - Recursively perform linear fits in wire-tick space
  - Select hits within some tolerance of this fit
- Written new 3D shower tool:
  - Order space points based on projection along direction in 3D
  - Create a cylinder from the vertex along the direction of the shower
  - FCL parameters define the dimensions of the cylinder
  - New tool that still needs tuning
- These hits are passed to the dEdx module

Tracking

- Can either use PMA which takes the track hits and draws a straight line from them using the two most populated planes.
- Or the Pandora sliding fit track finder which is a la PandoraTrackCreation.
- In the base chain we don not use the the recob::track at all. But new tools which calculate the direction from the track have shown to have good improvements in the direction and dEdx if the start position is correctly found

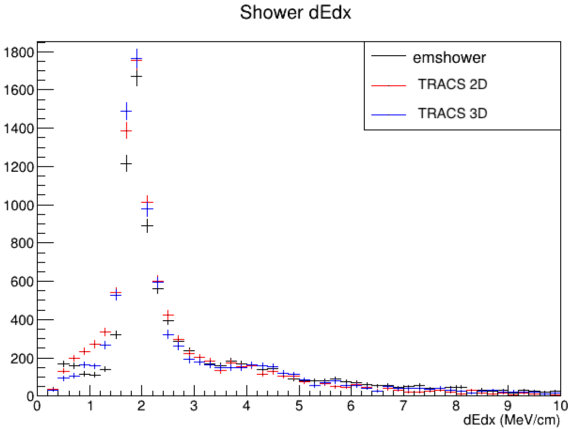
Debug Event Display



Shower dEdx

- Loops over the wires with hits in the initial track:
  - Discards the first hit
  - Uses hits within some FCL parameter distance of the vertex (2.4cm default)
- Finds the median hit integral (dQ) from these hits.
- Converts this dQ to dE using CalorimetryAlg
- Calculates an effective pitch for each plane based on shower direction
- Returns a dE/dx value for each plane
- Defines the best plane as plane with the most hits within the distance

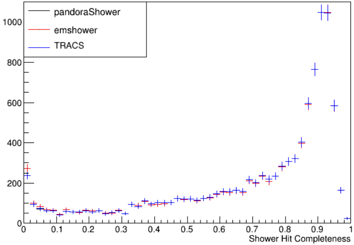
Shower dEdx


Shower dEdx
emshower
TRACS 2D
TRACS 3D
0
200
400
600
800
1000
1200
1400
1600
1800
0
1
2
3
4
5
6
7
8
9
10
dEdx (MeV/cm)

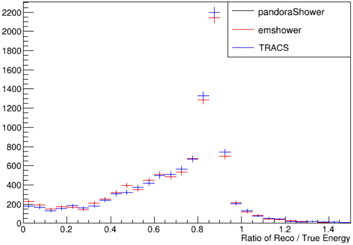
Shower Energy

- Method heavily based on EMShower method
- Collects all of the hits in each plane
- Correct for electron lifetime
- Sum all of the integrals in each plane to calculate total charge deposited
- Convert this charge to energy using a linear fit
  - These fit parameters are found by analyzing muon tracks (Currently uses truth information)
- Energy reconstruction is heavily dependant on the clustering and shower segmentation currently hurts this module
- Caveat that this is on an idealistic MC sample: No space charge, DIC etc. (Eventually Calibration group will provide more compete mappings)

Shower Energy



Shower Hits Completeness



Shower Energy Ratio

Conclusion

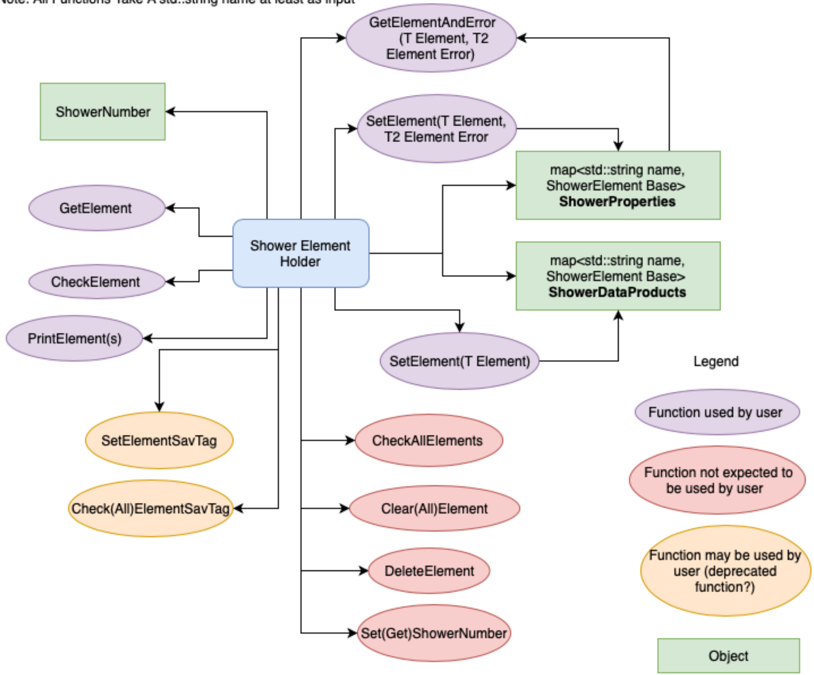
- We have created a new shower reconstruction module that uses art-tools to segment the calculation
- A user can easily plug and play new tools into the chain to improve their reconstruction.
- The framework is very malleable and the user can save whatever they want so it can be used in later tools and in saved in the art event.
- There exist a good base tool list for users to run out of the box which gives a similar performance to existing reconstruction modules.
- There exists a cheating tools which help in evaluating the performances of downstream tools.
- Further tools are all ready under development with promising results.
- Anyone who is interested in development feel free to contact us. We would love some user feedback and more tools.

Backup (passed here SBNShower = TRACS)

Shower Element Holder

- The Shower Element holder has an inheritance structure with the derived classes being templates. Hence the user can put whatever they want in the holder.
- Elements are set with std::string name so they can be accessed.
- The properties that go into the shower have specific names: "ShowerStartPosition", "ShowerDirection","ShowerEnergy","ShowerdEdx","ShowerBestPlane". More can easily follow.
- The holder now holds objects called data products and objects called properties. Both derive from the same base class and so are treated similar for the user.
- The only difference between the two is that a property is given an error when it is set whilst a data product has the potential of being saved in the Event.
- The end user does not directly interact with the elements.

Note: All Functions Take A std::string name at least as input



GetElementAndError (T Element, T2 Element Error)

ShowerNumber

SetElement(T Element, T2 Element Error

map<std::string name, ShowerElement Base> **ShowerProperties**

GetElement

Shower Element Holder

CheckElement

map<std::string name, ShowerElement Base> **ShowerDataProducts**

PrintElement(s)

SetElement(T Element)

Legend

SetElementSavTag

Function used by user

CheckAllElements

Function not expected to be used by user

Check(All)ElementSavTag

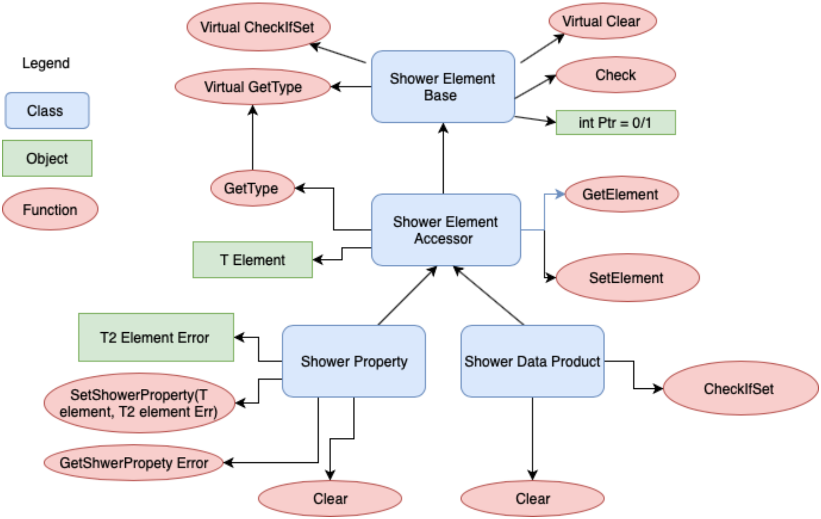
Clear(All)Element

DeleteElement

Function may be used by user (deprecated function?)

Set(Get)ShowerNumber

Object

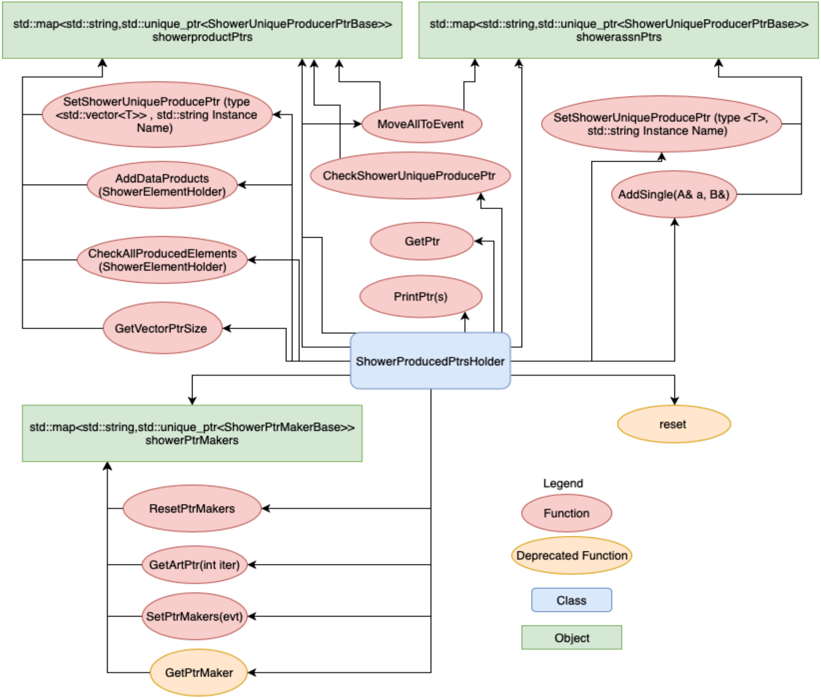

Legend
Class
Object
Function
Virtual CheckIfSet
Virtual GetType
Shower Element Base
Virtual Clear
Check
int Ptr = 0/1
GetType
Shower Element Accessor
GetElement
T Element
SetElement
T2 Element Error
SetShowerProperty(T element, T2 element Err)
GetShwerPropety Error
Shower Property
Shower Data Product
CheckIfSet
Clear
Clear

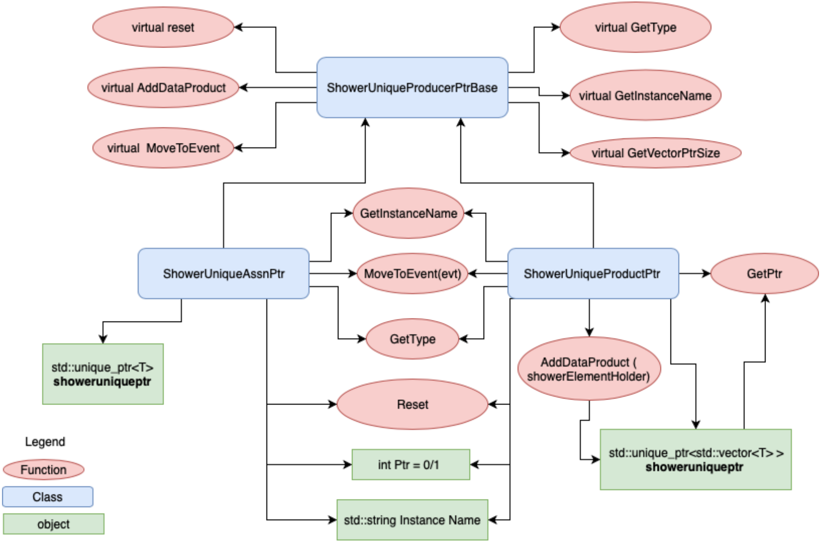
ShowerProducedPtrsHolder

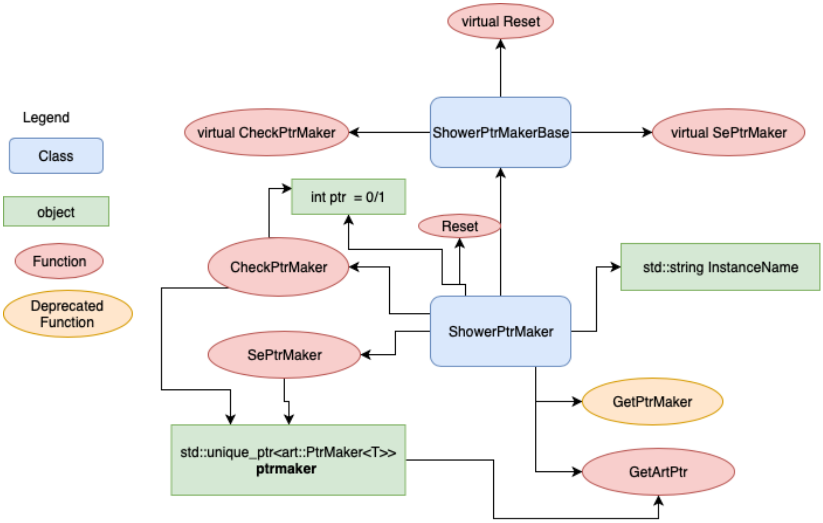
- This object holds the unique ptrs that are going to be saved in the event.
- As this is a event object it is cleared at the start of the event. But the object is initialised in the beginJob() function.
- The object also holds the PtrMakers for all the data objects being stored so that a user can make ArtPtrs to make associations
- Even the assocations and recob::shower are stored in here.
- IMPORTANT: The user never deals with the holder directly. The showertools now have functions to get the elements.
- The holder holds associations and vector<T> unique ptrs (you have one ptr for each event but you have multiple showers per event e.g. std::vector<recob::shower>)
- The data-product stored in the ShowerElementHolder are then added to the unique ptr.

ShowerProducedPtrsHolder Cont

- IMPORTANT: Elements stored in here must have the same name as in the shower element holder. Data products that exist without a corresponding ptr exist but are not stored.
- IMPORTANT: The user does not add the ptrs directly in the event.
- This has the similar structure to the shower element holder when the unique ptrs are stored in a base class ShowerUniquePtrBase but the underlying object is a derived template class, ShowerUniqueAssnPtr, ShowerUniqueProductPtr

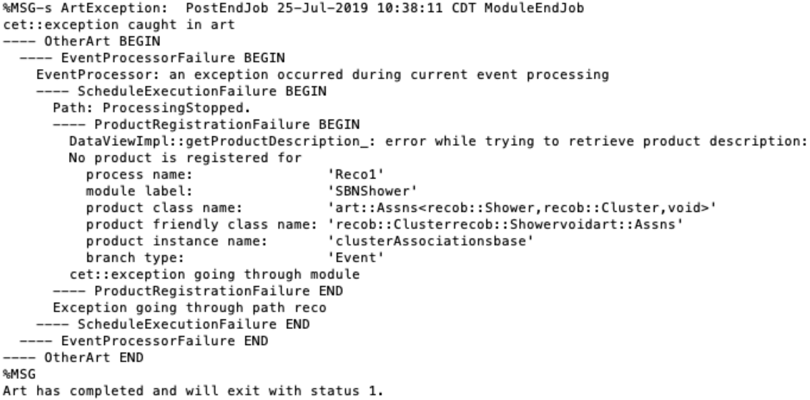

std::map<std::string,std::unique_ptr<ShowerUniqueProducerPtrBase>> showerproductPtrs
std::map<std::string,std::unique_ptr<ShowerUniqueProducerPtrBase>> showerassnPtrs
SetShowerUniqueProducePtr (type <std::vector<T>> , std::string Instance Name)
MoveAllToEvent
SetShowerUniqueProducePtr (type <T>, std::string Instance Name)
CheckShowerUniqueProducePtr
AddDataProducts (ShowerElementHolder)
AddSingle(A& a, B&)
GetPtr
CheckAllProducedElements (ShowerElementHolder)
PrintPtr(s)
GetVectorPtrSize
ShowerProducedPtrsHolder
std::map<std::string,std::unique_ptr<ShowerPtrMakerBase>> showerPtrMakers
reset
ResetPtrMakers
GetArtPtr(int iter)
SetPtrMakers(evt)
GetPtrMaker
Legend
Function
Deprecated Function
Class
Object


virtual reset
virtual AddDataProduct
virtual MoveToEvent
ShowerUniqueProducerPtrBase
virtual GetType
virtual GetInstanceName
virtual GetVectorPtrSize
GetInstanceName
ShowerUniqueAssnPtr
MoveToEvent(evt)
ShowerUniqueProductPtr
GetPtr
GetType
std::unique_ptr<T>
showeruniqueptr
AddDataProduct (
showerElementHolder)
Reset
std::unique_ptr<std::vector<T> >
showeruniqueptr
int Ptr = 0/1
std::string Instance Name
Legend
Function
Class
object


virtual Reset
Legend
Class
object
Function
Deprecated Function
virtual CheckPtrMaker
ShowerPtrMakerBase
virtual SePtrMaker
int ptr = 0/1
Reset
CheckPtrMaker
std::string InstanceName
ShowerPtrMaker
SePtrMaker
GetPtrMaker
std::unique_ptr<art::PtrMaker<T>> **ptrmaker**
GetArtPtr

Catches

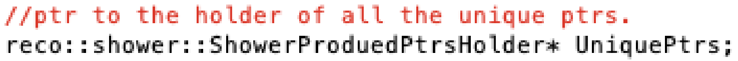
There is plenty of exception that to try prevent the user doing something silly. I need new people to try out the framework and see how they get on with the information proved



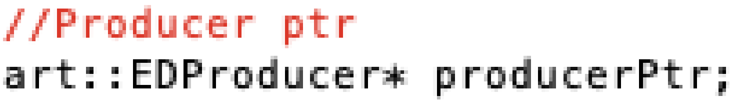
```
%MSG-s ArtException:  PostEndJob 25-Jul-2019 10:38:11 CDT ModuleEndJob
cet::exception caught in art
---- OtherArt BEGIN
  ---- EventProcessorFailure BEGIN
    EventProcessor: an exception occurred during current event processing
    ---- ScheduleExecutionFailure BEGIN
      Path: ProcessingStopped.
      ---- ProductRegistrationFailure BEGIN
        DataViewImpl::getProductDescription_: error while trying to retrieve product description:
        No product is registered for
          process name:                'Reco1'
          module label:                'SBNShower'
          product class name:          'art::Assns<recob::Shower,recob::Cluster,void>'
          product friendly class name: 'recob::Clusterrecob::Showervoidart::Assns'
          product instance name:       'clusterAssociationsbase'
          branch type:                 'Event'
        cet::exception going through module
      ---- ProductRegistrationFailure END
      Exception going through path reco
    ---- ScheduleExecutionFailure END
  ---- EventProcessorFailure END
---- OtherArt END
%MSG
Art has completed and will exit with status 1.
```

IShowerTool

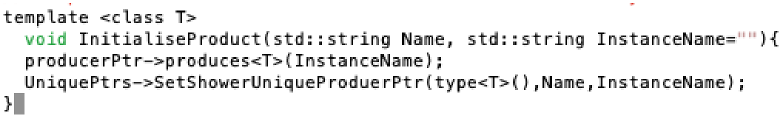
- The following functions are how the user communicate with the unique ptrs.
- How to use these function is explained in the example tool.



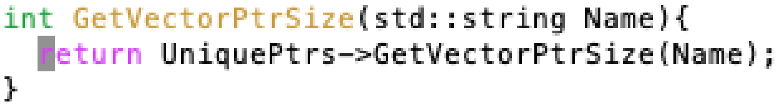
```
//ptr to the holder of all the unique ptrs.
reco::shower::ShowerProduedPtrsHolder* UniquePtrs;
```



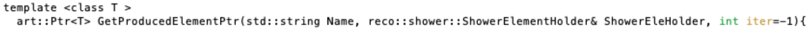
```
//Producer ptr
art::EDProducer* producerPtr;
```



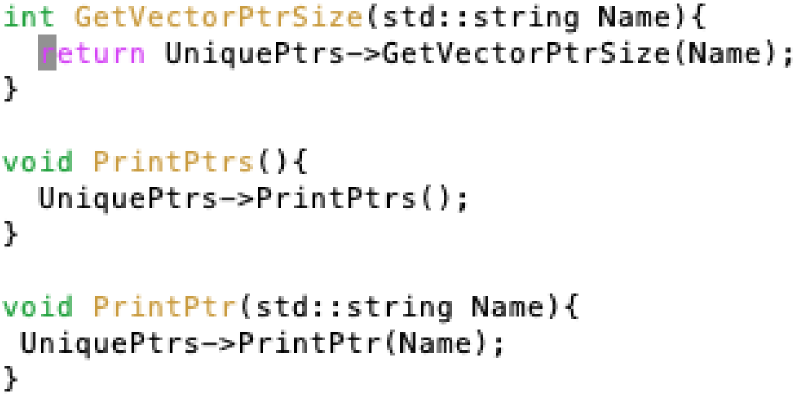
```
template <class T>
  void InitialiseProduct(std::string Name, std::string InstanceName=""){
  producerPtr->produces<T>(InstanceName);
  UniquePtrs->SetShowerUniqueProduerPtr(type<T>(),Name,InstanceName);
}
```



```
int GetVectorPtrSize(std::string Name){
  return UniquePtrs->GetVectorPtrSize(Name);
}
```



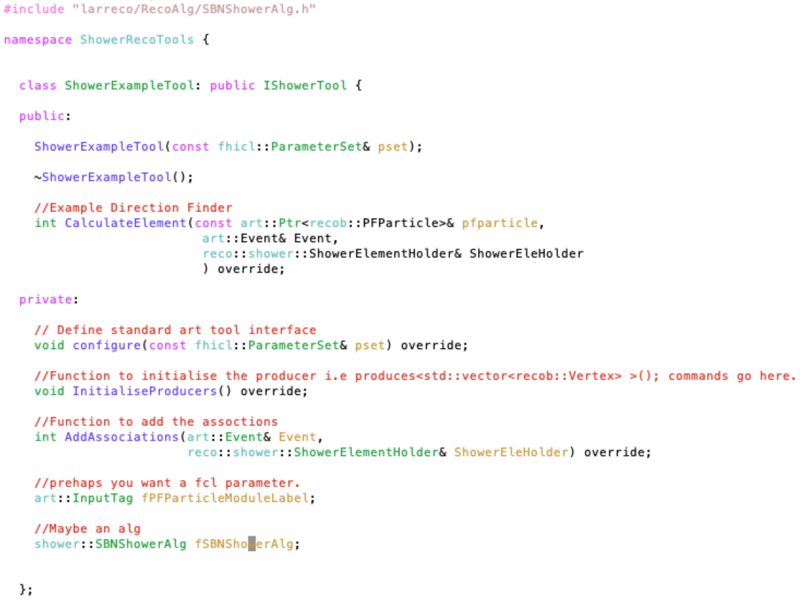
```
template <class T >
  art::Ptr<T> GetProducedElementPtr(std::string Name, reco::shower::ShowerElementHolder& ShowerEleHolder, int iter=-1){
```



```
int GetVectorPtrSize(std::string Name){
  return UniquePtrs->GetVectorPtrSize(Name);
}

void PrintPtrs(){
  UniquePtrs->PrintPtrs();
}

void PrintPtr(std::string Name){
 UniquePtrs->PrintPtr(Name);
}
```

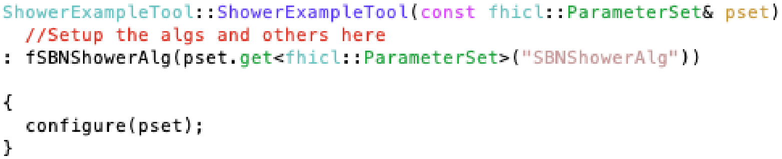
Running through the example tool.

The Class



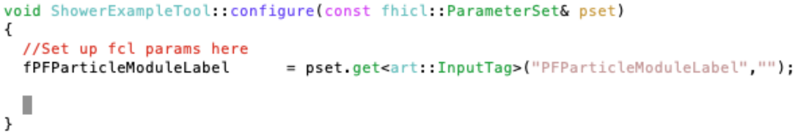
```
#include "larreco/RecoAlg/SBNShowerAlg.h"

namespace ShowerRecoTools {


  class ShowerExampleTool: public IShowerTool {

  public:

    ShowerExampleTool(const fhicl::ParameterSet& pset);

    ~ShowerExampleTool();

    //Example Direction Finder
    int CalculateElement(const art::Ptr<recob::PFParticle>& pfparticle,
                          art::Event& Event,
                          reco::shower::ShowerElementHolder& ShowerEleHolder
                          ) override;

  private:

    // Define standard art tool interface
    void configure(const fhicl::ParameterSet& pset) override;

    //Function to initialise the producer i.e produces<std::vector<recob::Vertex> >(); commands go here.
    void InitialiseProducers() override;

    //Function to add the assoctions
    int AddAssociations(art::Event& Event,
                        reco::shower::ShowerElementHolder& ShowerEleHolder) override;

    //prehaps you want a fcl parameter.
    art::InputTag fPFParticleModuleLabel;

    //Maybe an alg
    shower::SBNShowerAlg fSBNShowerAlg;


  };
```

How the class is Initialised



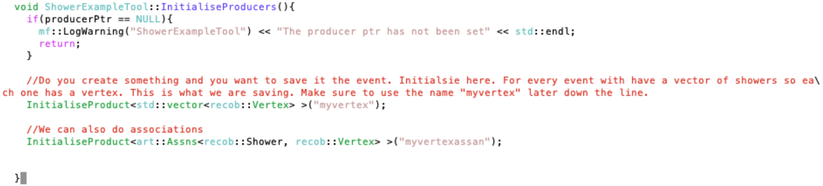
```
ShowerExampleTool::ShowerExampleTool(const fhicl::ParameterSet& pset)
  //Setup the algs and others here
: fSBNShowerAlg(pset.get<fhicl::ParameterSet>("SBNShowerAlg"))

{
  configure(pset);
}
```

How to configure fcl parameters



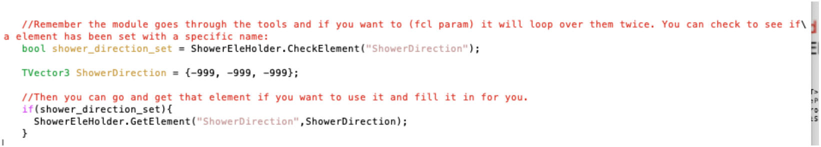
```
void ShowerExampleTool::configure(const fhicl::ParameterSet& pset)
{
  //Set up fcl params here
  fPFParticleModuleLabel      = pset.get<art::InputTag>("PFParticleModuleLabel","");

}
```

How to Initialise data products to be stored into the event



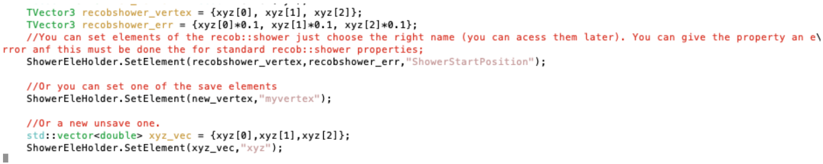
```
  void ShowerExampleTool::InitialiseProducers(){
    if(producerPtr == NULL){
      mf::LogWarning("ShowerExampleTool") << "The producer ptr has not been set" << std::endl;
      return;
    }

    //Do you create something and you want to save it the event. Initialsie here. For every event with have a vector of showers so ea\
ch one has a vertex. This is what we are saving. Make sure to use the name "myvertex" later down the line.
    InitialiseProduct<std::vector<recob::Vertex> >("myvertex");

    //We can also do associations
    InitialiseProduct<art::Assns<recob::Shower, recob::Vertex> >("myvertexassan");


  }
```

Check and Get Elements stored in the element holder



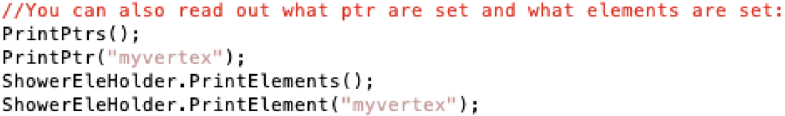
```
   //Remember the module goes through the tools and if you want to (fcl param) it will loop over them twice. You can check to see if\
a element has been set with a specific name:
   bool shower_direction_set = ShowerEleHolder.CheckElement("ShowerDirection");

   TVector3 ShowerDirection = {-999, -999, -999};

   //Then you can go and get that element if you want to use it and fill it in for you.
   if(shower_direction_set){
     ShowerEleHolder.GetElement("ShowerDirection",ShowerDirection);
   }
```

Set an Element stored in the element holder



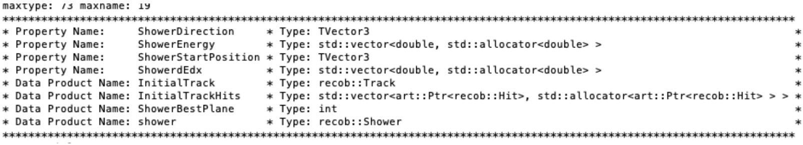
```
    TVector3 recobshower_vertex = {xyz[0], xyz[1], xyz[2]};
    TVector3 recobshower_err = {xyz[0]*0.1, xyz[1]*0.1, xyz[2]*0.1};
    //You can set elements of the recob::shower just choose the right name (you can acess them later). You can give the property an e\
rror anf this must be done the for standard recob::shower properties;
    ShowerEleHolder.SetElement(recobshower_vertex,recobshower_err,"ShowerStartPosition");

    //Or you can set one of the save elements
    ShowerEleHolder.SetElement(new_vertex,"myvertex");

    //Or a new unsave one.
    std::vector<double> xyz_vec = {xyz[0],xyz[1],xyz[2]};
    ShowerEleHolder.SetElement(xyz_vec,"xyz");
```

print what is stored



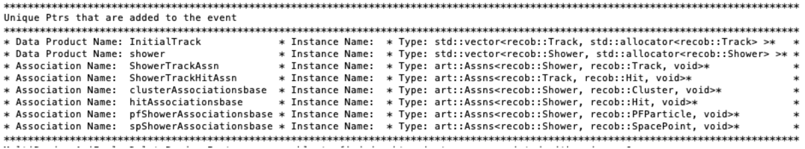
```
//You can also read out what ptr are set and what elements are set:
PrintPtrs();
PrintPtr("myvertex");
ShowerEleHolder.PrintElements();
ShowerEleHolder.PrintElement("myvertex");
```

Element Holder Print



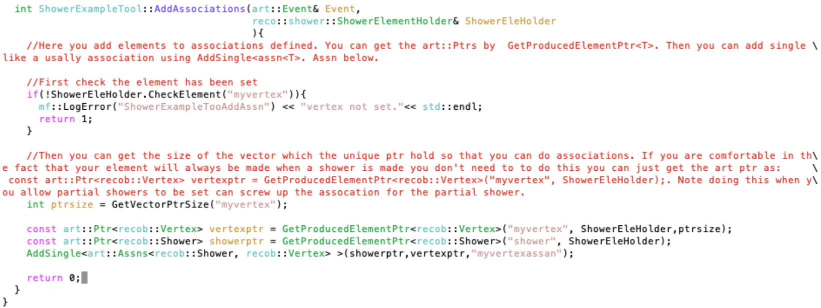
```
maxtype: 73 maxname: 19
***************************************************************************************************************************
* Property Name:     ShowerDirection     * Type: TVector3                                                                  *
* Property Name:     ShowerEnergy        * Type: std::vector<double, std::allocator<double> >                              *
* Property Name:     ShowerStartPosition * Type: TVector3                                                                  *
* Property Name:     ShowerdEdx          * Type: std::vector<double, std::allocator<double> >                              *
* Data Product Name: InitialTrack        * Type: recob::Track                                                              *
* Data Product Name: InitialTrackHits    * Type: std::vector<art::Ptr<recob::Hit>, std::allocator<art::Ptr<recob::Hit> > > *
* Data Product Name: ShowerBestPlane     * Type: int                                                                       *
* Data Product Name: shower              * Type: recob::Shower                                                             *
***************************************************************************************************************************
```

Unique Ptr Holder Print



```
*************************************************************************************************************************************
Unique Ptrs that are added to the event
*************************************************************************************************************************************
* Data Product Name: InitialTrack             * Instance Name:  * Type: std::vector<recob::Track, std::allocator<recob::Track> >*   *
* Data Product Name: shower                   * Instance Name:  * Type: std::vector<recob::Shower, std::allocator<recob::Shower> >* *
* Association Name:  ShowerTrackAssn          * Instance Name:  * Type: art::Assns<recob::Shower, recob::Track, void>*              *
* Association Name:  ShowerTrackHitAssn       * Instance Name:  * Type: art::Assns<recob::Track, recob::Hit, void>*                 *
* Association Name:  clusterAssociationsbase  * Instance Name:  * Type: art::Assns<recob::Shower, recob::Cluster, void>*            *
* Association Name:  hitAssociationsbase      * Instance Name:  * Type: art::Assns<recob::Shower, recob::Hit, void>*                *
* Association Name:  pfShowerAssociationsbase * Instance Name:  * Type: art::Assns<recob::Shower, recob::PFParticle, void>*         *
* Association Name:  spShowerAssociationsbase * Instance Name:  * Type: art::Assns<recob::Shower, recob::SpacePoint, void>*         *
*************************************************************************************************************************************
```

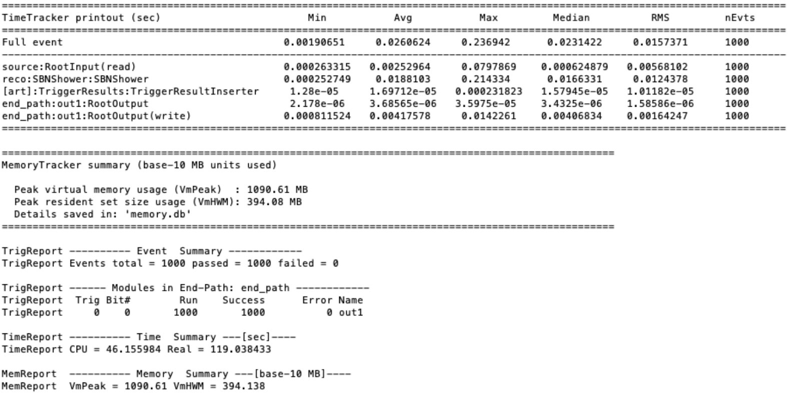
Make your associations



```
  int ShowerExampleTool::AddAssociations(art::Event& Event,
                                         reco::shower::ShowerElementHolder& ShowerEleHolder
                                         ){
    //Here you add elements to associations defined. You can get the art::Ptrs by  GetProducedElementPtr<T>. Then you can add single \
like a usally association using AddSingle<assn<T>. Assn below.

    //First check the element has been set
    if(!ShowerEleHolder.CheckElement("myvertex")){
      mf::LogError("ShowerExampleTooAddAssn") << "vertex not set."<< std::endl;
      return 1;
    }

    //Then you can get the size of the vector which the unique ptr hold so that you can do associations. If you are comfortable in th\
e fact that your element will always be made when a shower is made you don't need to to do this you can just get the art ptr as:     \
 const art::Ptr<recob::Vertex> vertexptr = GetProducedElementPtr<recob::Vertex>("myvertex", ShowerEleHolder);. Note doing this when y\
ou allow partial showers to be set can screw up the assocation for the partial shower.
    int ptrsize = GetVectorPtrSize("myvertex");

    const art::Ptr<recob::Vertex> vertexptr = GetProducedElementPtr<recob::Vertex>("myvertex", ShowerEleHolder,ptrsize);
    const art::Ptr<recob::Shower> showerptr = GetProducedElementPtr<recob::Shower>("shower", ShowerEleHolder);
    AddSingle<art::Assns<recob::Shower, recob::Vertex> >(showerptr,vertexptr,"myvertexassan");

    return 0;
  }
}
```

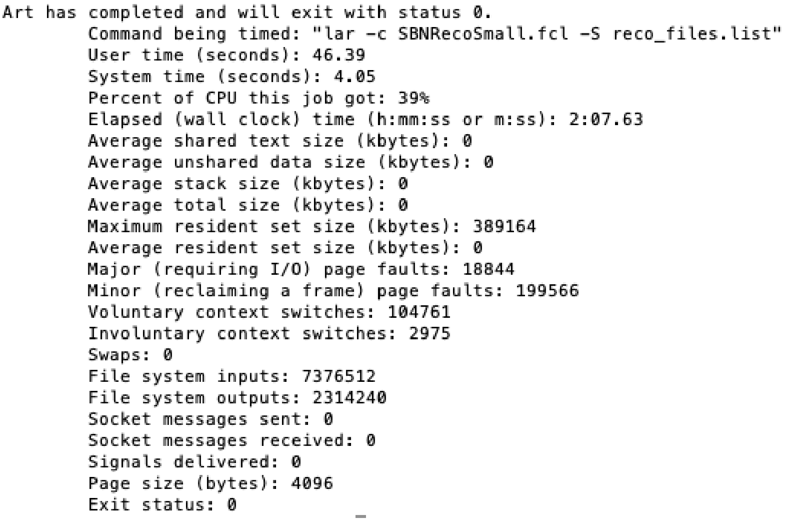
Note on timing and memory usage

This has not been properly analyses but the suggestions from the larsoft helper suggest that we still have a good speed and memory usage. Even with all the dynamic casting of the templates.



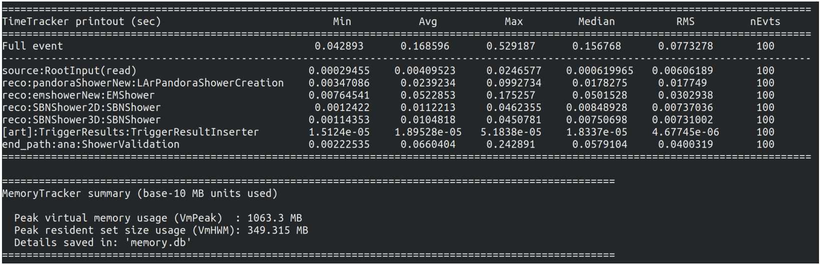
```
===============================================================================================================================
TimeTracker printout (sec)                        Min           Avg           Max         Median          RMS         nEvts
===============================================================================================================================
Full event                                    0.00190651     0.0260624     0.236942      0.0231422     0.0157371      1000
-------------------------------------------------------------------------------------------------------------------------------
source:RootInput(read)                        0.000263315   0.00252964     0.0797869    0.000624879   0.00568102      1000
reco:SBNShower:SBNShower                      0.000252749    0.0188103     0.214334      0.0166331     0.0124378      1000
[art]:TriggerResults:TriggerResultInserter     1.28e-05     1.69712e-05   0.000231823   1.57945e-05   1.01182e-05     1000
end_path:out1:RootOutput                       2.178e-06    3.68565e-06   3.5975e-05    3.4325e-06    1.58586e-06     1000
end_path:out1:RootOutput(write)               0.000811524   0.00417578     0.0142261    0.00406834    0.00164247      1000
===============================================================================================================================

====================================================================================================
MemoryTracker summary (base-10 MB units used)

  Peak virtual memory usage (VmPeak)  : 1090.61 MB
  Peak resident set size usage (VmHWM): 394.08 MB
  Details saved in: 'memory.db'
====================================================================================================

TrigReport ---------- Event  Summary ------------
TrigReport Events total = 1000 passed = 1000 failed = 0

TrigReport ------ Modules in End-Path: end_path ------------
TrigReport  Trig Bit#        Run    Success      Error Name
TrigReport     0    0       1000       1000          0 out1

TimeReport ---------- Time  Summary ---[sec]----
TimeReport CPU = 46.155984 Real = 119.038433

MemReport  ---------- Memory  Summary ---[base-10 MB]----
MemReport  VmPeak = 1090.61 VmHWM = 394.138
```

time results



```
Art has completed and will exit with status 0.
        Command being timed: "lar -c SBNRecoSmall.fcl -S reco_files.list"
        User time (seconds): 46.39
        System time (seconds): 4.05
        Percent of CPU this job got: 39%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 2:07.63
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 389164
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 18844
        Minor (reclaiming a frame) page faults: 199566
        Voluntary context switches: 104761
        Involuntary context switches: 2975
        Swaps: 0
        File system inputs: 7376512
        File system outputs: 2314240
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

Comparison in CPU Time against EMShower and Pandora Shower



```
====================================================================================================================================
TimeTracker printout (sec)                            Min           Avg           Max         Median          RMS         nEvts
====================================================================================================================================
Full event                                         0.042893      0.168596      0.529187      0.156768      0.0773278       100
------------------------------------------------------------------------------------------------------------------------------------
source:RootInput(read)                            0.00029455    0.00409523     0.0246577    0.000619965   0.00606189       100
reco:pandoraShowerNew:LArPandoraShowerCreation    0.00347086     0.0239234     0.0992734     0.0178275     0.017749        100
reco:emshowerNew:EMShower                         0.00764541     0.0522853     0.175257      0.0501528     0.0302938       100
reco:SBNShower2D:SBNShower                         0.0012422     0.0112213     0.0462355    0.00848928    0.00737036       100
reco:SBNShower3D:SBNShower                        0.00114353     0.0104818     0.0450781    0.00750698    0.00731002       100
[art]:TriggerResults:TriggerResultInserter        1.5124e-05    1.89528e-05   5.1838e-05    1.8337e-05    4.67745e-06      100
end_path:ana:ShowerValidation                     0.00222535     0.0660404     0.242891      0.0579104     0.0400319       100
====================================================================================================================================

====================================================================================================
MemoryTracker summary (base-10 MB units used)

  Peak virtual memory usage (VmPeak)  : 1063.3 MB
  Peak resident set size usage (VmHWM): 349.315 MB
  Details saved in: 'memory.db'
====================================================================================================
```

New Shower Tools

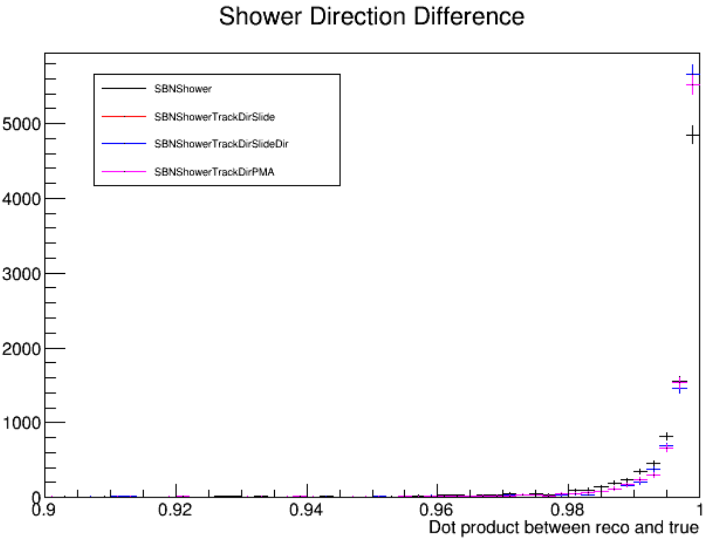
Tools for Calculating the Direction from the Unitial Track

- Several tools have been created to calculate the direction of the shower from the initial track. I'll go through each of them now. Some you might like.
- Many combinations have been tried to see which spits out the best direction.
- Even using the tool iteratively has been tested.
- PMA was only used on ShowerTrackDirection_tool.cc as its a straight line so the results would be the same for the rest of the tools.
- For comparison to SBN shower the direction was only compared to events where the vertex was correct to within 5 cm. This is an important note if you get the position wrong you get the track wrong and the direction wrong. The PCA seem to do a better job at getting the position roughly right most of the time so what is better? all(not all) events roughly right. Some events really right. Considering tools check the start position.

ShowerTrackDirection_tool.cc

- fcl parameters: fUsePandoraVertex,fUsePositionInfo
- From the trajectory points of the track calculate the direction from the pandoravertex or the first trajectory point in the track.
- direction is calculated either from the taking the direction from the (geo::Point_t) 3D coordinates themselves or using the trajectory points (geo::Vector_t) direction (DirectionAtPoint) (depends on fUsePositionInfo).
- A mean value of the direction is calculated along with the RMS in each direction.
- Any trajectory point directions that do not lie with 1 RMS of the mean are remove and the mean is calculated again.
- The mean is taken as the direction of the shower, whilst the rms is the error.

ShowerTrackDirection_tool.cc



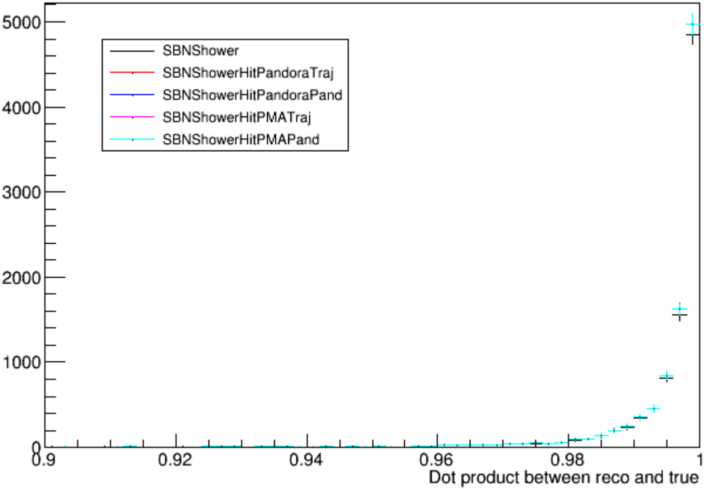
ShowerTrackSpacePointDirection_tool.cc

- fcl parameters: fUsePandoraVertex
- Exact same procedure as the TrackDirection tool. Difference is that the spacepoints from the track hit tool are used instead.

ShowerTrackHitDirection_tool.cc

- fcl parameters: fUsePandoraVertex
- Exact same procedure as the ShwerSpacePoint tool. Only difference is that the spacepoints associated to hit need to found first.

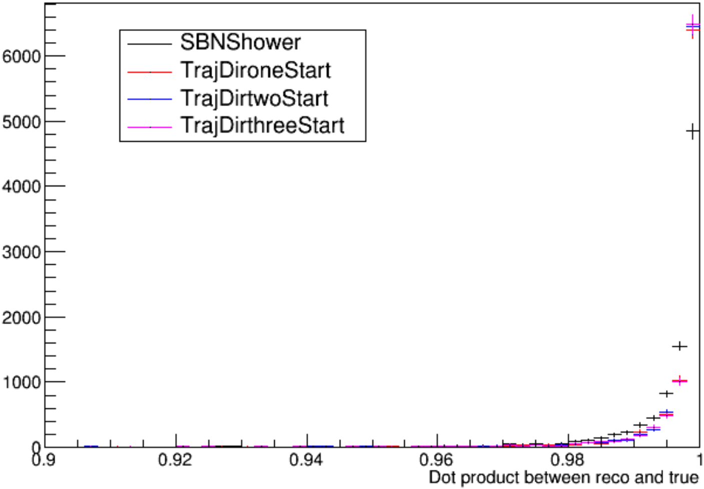
ShowerTrackHitDirection_tool.cc



ShowerTrackTrajectoryPointDirection_tool.cc

- fcl parameters: fUsePandoraVertex, fUsePositonInfo,fTrajPoint
- Rather than calculate the average value instead take the direction from the fTrajPoint as the direction.
- The direction is either defined as the direction from the point to start position or the DirectionAtPoint value of the track.

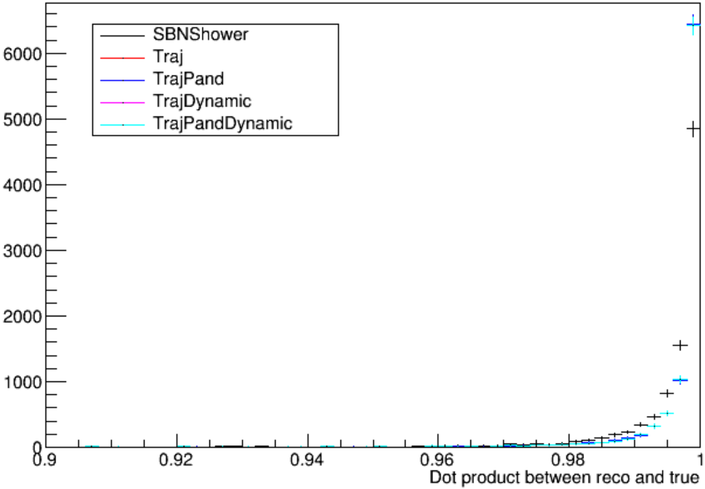
ShowerTrackTrajectoryPointDirection_tool.cc



ShowerSmartTrackTrajectoryPointDirection_tool.cc

- fcl parameters: fUsePandoraVertex, fAllowDynamicSliding, fUseStartPos, fUsePositionInfo, fAngleCut
- Finds the direction of the first two trajectory points. If angle between the two directions is less then fAngleCut it then moves on up one trajectory point and redoes the analysis.
- When the angle is less than fAngleCut the direction of the first trajectory point is taken as the direction of the shower.
- if fAllowDynamicSliding is turned on then rather than use the direction from the start position to the two trajectory point it looks at the direction from the previous trajectory point.
- fAllowDynamicSliding only works for when fUsePositionInfo is true.

ShowerSmartTrackTrajectoryPointDirection_tool.cc

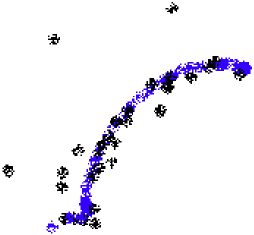


ShowerTrackPCADirection_tool.cc

- Perform the PCA analysis on the track hits only.
- Performs poorly and needs reconsidering.

Consider a banana shaped track coming from a messy vertex

- This is a case where most of the above tools will fail.
- ShowerTrackDirection_tool.cc the average track direction is not the start direction of the tool.
- ShowerTrackSpacePointDirection_tool. Too many random hits that won't be remove from the average skewing the direction. Also average is not the start.
- ShowerTrackTrajectoryPointDirection_ mess at the vertex. What point do you take to measure the direction?
- ShowerSmartTrackTrajectoryPointDirection_tool.cc: Has potential if tuned correctly.



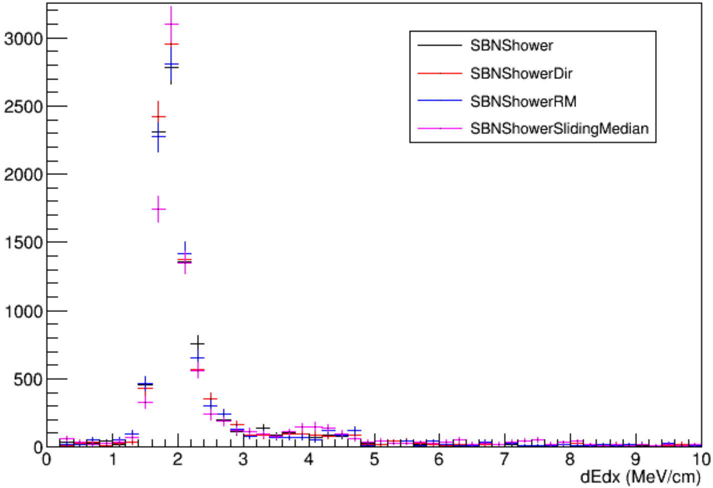
Track Hit removal: ShowerTrackTrajToSpacepoint_tool.cc

- Once you have your track hits and you fit a track we still keep all the hits even if some are not from the track.
- Looping over the trajectory points match the closest spacepoint within fMaxDist and add it to the new track list.
- Remove the added spacepoint from consideration in other trajectory points.
- Currently causing the metrics to be worse somehow but might be more usuful for neutrino events.

New Calorimetry Tool - ShowerSlidingStandardCalodEdx

- fcl parameters: fMinDistCutOff,fMaxDist,fMinAngleToWire, fShapingTime, fdEdxTrackLength, fUseMedian
- Still early stages.
- Loop over the spacepoints. If the spacepoint is greater than fMinDistCutOff wires away from the start position and less than fdEdxTrackLength then use it.
- Find the closest trajectory point within fMaxDist, that is not bogus, to the spacepoint in the track. and find the track direction at the spacepoint.
- If the angle between trajectory direction and the Increasing Wire Direction of the plane is less than fMinAngleToWire do not consider the spacepoint.
- Find the distance travelled in the drift direction within one wire pitch. If the time taken to travel that distance by the drifting electrons is greater than the shaping time remove the spacepoint.
- Calculate the track pitch as: (TrajDirection*(wirepitch/TrajDirection.Dot(PlaneWireDirection))).Mag().

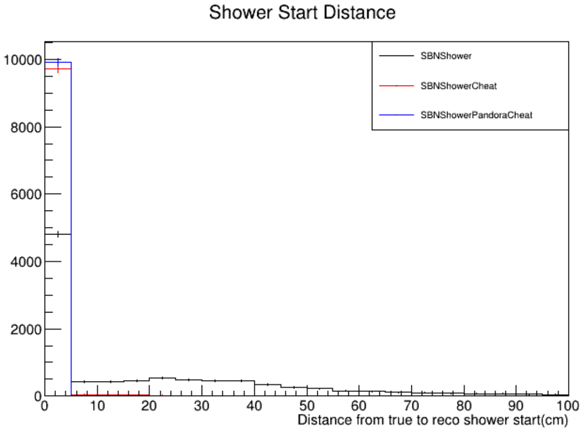
dEdx


SBNShower
SBNShowerDir
SBNShowerRM
SBNShowerSlidingMedian
0
500
1000
1500
2000
2500
3000
0
1
2
3
4
5
6
7
8
9
10
dEdx (MeV/cm)

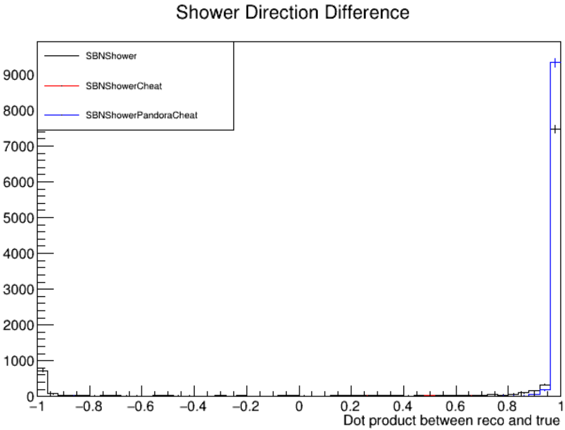
SBNShower Cheating

- Implemented cheating modules for SBNShower:
  - Cheating modules to get truth (Start position, direction and initial track hits)
  - Implemented Pandora cheating clustering (Not working fully as some assns are not being made?)
- Can be dropped in at any part in the chain (pandora vertex, true direction)
- These allow for validation of calculations (e.g. checking dEdx calculation works with perfect input)
- Allows for testing and tuning of methods in truth (e.g. optimise cylinder dimensions for initial track hit finding based on truth distributions of initial track hits)
- Caveat that this cheating still relies on hitfinding and Pandora SpacePoints

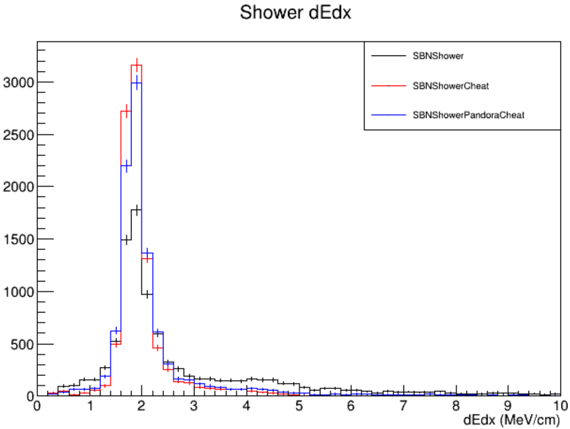
Shower Start Distance



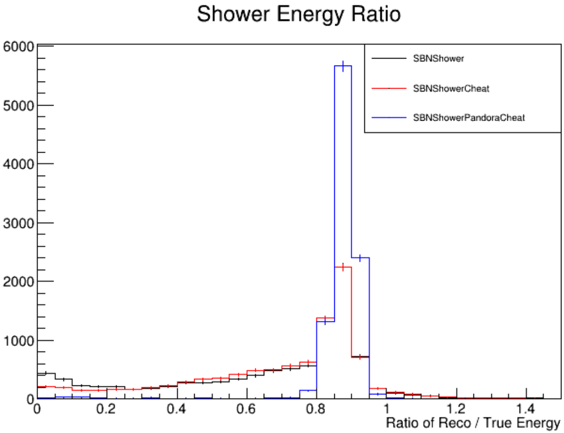
Shower Direction



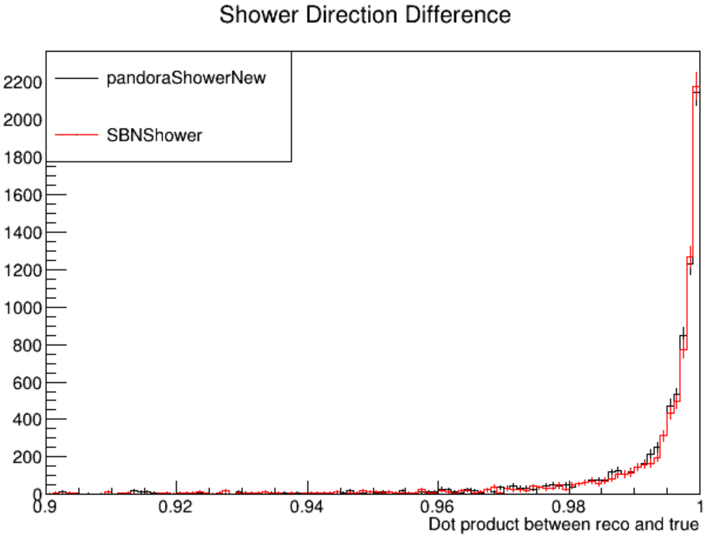
Shower dEdx


Shower dEdx
SBNShower
SBNShowerCheat
SBNShowerPandoraCheat
3000
2500
2000
1500
1000
500
0
0
1
2
3
4
5
6
7
8
9
10
dEdx (MeV/cm)

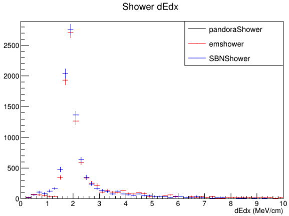
Shower Energy Ratio

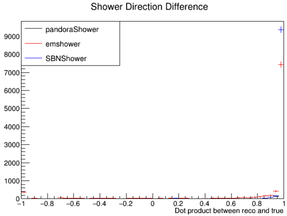


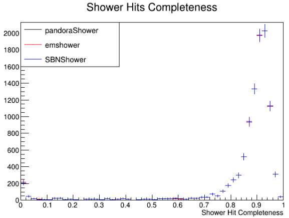
Direction Difference Zoomed in

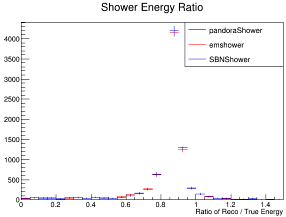


Shower Start Position $< 5cm$


Shower dEdx


Shower Direction Difference


Shower Hits Completeness


Shower Energy Ratio