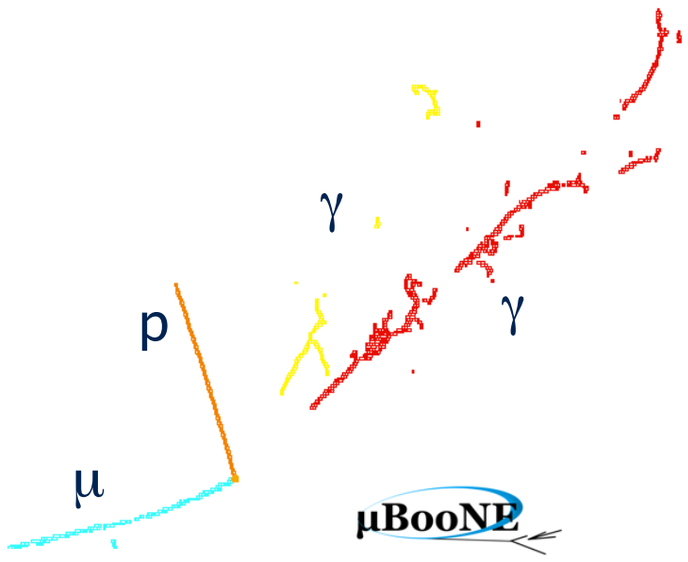


Pandora Exercise 8: Track/Shower Id

J. S. Marshall for the Pandora Team

MicroBooNE Pandora Workshop

July 11-14th 2016, Cambridge

Track vs. Shower Identification

This document provides just a starting template for further exploration. It is assumed that the reader will have already worked through the following exercises:

Pre-requisite: Exercise 2 - setup Pandora environment and add a new algorithm.

Pre-requisite: Exercise 3 - configure a new algorithm, use APIs and build first Clusters.

Track vs. Shower Identification

This document introduces an example implementation to develop topological selection cuts to perform track/shower id.

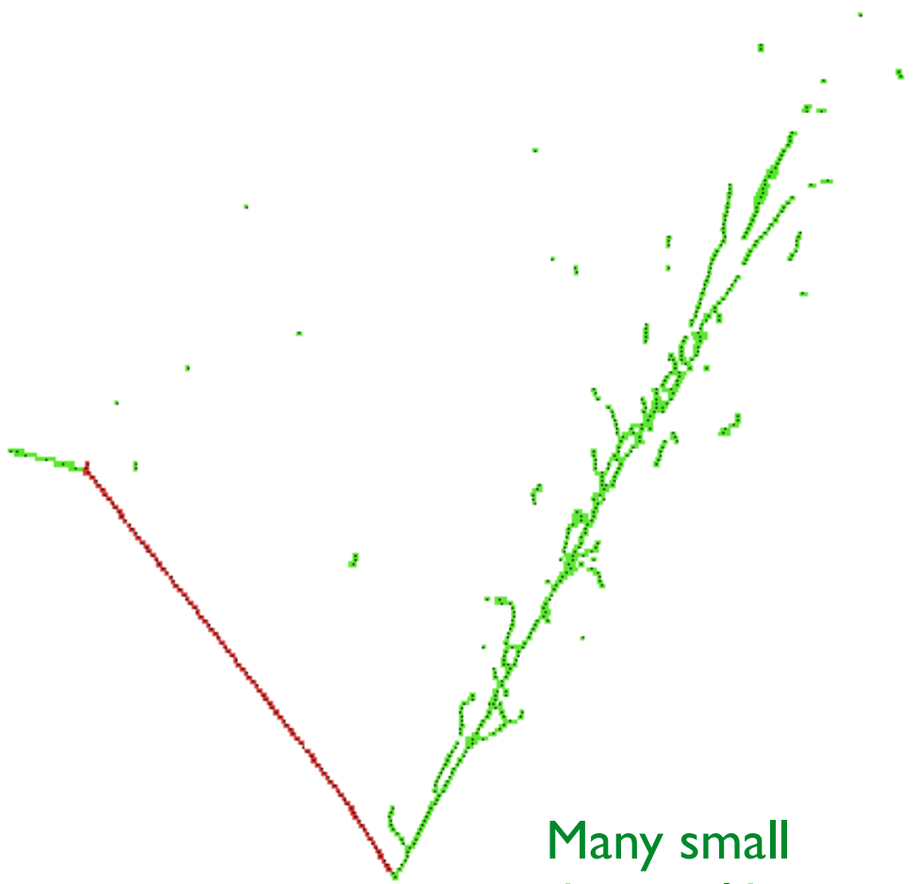
The example investigates 2D Clusters and uses MC information to determine their true/ generated particle origin.

A few example topological properties are then calculated and the information is all written to a ROOT TTree using PandoraMonitoring.

Any cuts developed can be used to flag 2D Clusters with an indication of particle type.

Use of 3x2D cuts can allow PDG codes to be specified for output Particles.

2D Clusters



PandoraSettings

Suggest that you start with `PandoraSettings_MicroBooNE_SingleNeutrino.xml` and add a new `MyTrackShowerId` alg to look at W Clusters immediately after 2D reco:

```
...
<!-- 2D track reconstruction, W View -->
<algorithm type = "LArClusteringParent">
    <algorithm type = "LArTrackClusterCreation" description = "ClusterFormation"/>
    <InputCaloHitListName>CaloHitListW</InputCaloHitListName>
    <ClusterListName>ClustersW</ClusterListName>
    <ReplaceCurrentCaloHitList>false</ReplaceCurrentCaloHitList>
    <ReplaceCurrentClusterList>true</ReplaceCurrentClusterList>
</algorithm>
<algorithm type = "LArLayerSplitting"/>
<algorithm type = "LArLongitudinalAssociation"/>
<algorithm type = "LArTransverseAssociation"/>
<algorithm type = "LArLongitudinalExtension"/>
<algorithm type = "LArTransverseExtension"/>
<algorithm type = "LArCrossGapsAssociation"/>
<algorithm type = "LArCrossGapsExtension"/>
<algorithm type = "LArOvershootSplitting"/>
<algorithm type = "LArBranchSplitting"/>
<algorithm type = "LArKinkSplitting"/>
<algorithm type = "LArTrackConsolidation">
    <algorithm type = "LArSimpleClusterCreation" description = "ClusterRebuilding"/>
</algorithm>

<algorithm type = "LArVisualMonitoring">
    <CaloHitListNames>CaloHitListW</CaloHitListNames>
    <ClusterListNames>ClustersW</ClusterListNames>
    <MCParticleListNames>MCParticleList3D</MCParticleListNames>
    <SuppressMCParticles>22:0.01 2112:1.0</SuppressMCParticles>
    <ShowDetector>true</ShowDetector>
</algorithm>

<!-- New Track/Shower identification attempts -->
<algorithm type = "MyTrackShowerId">
    <InputClusterListName>ClustersW</InputClusterListName>
    <WriteToTree>true</WriteToTree>
    <OutputTree>MyTree</OutputTree>
    <OutputFile>MyTrackShowerId.root</OutputFile>
</algorithm>
...
```

Example Algorithm

```
/**
 *  @brief  MyTrackShowerIdAlgorithm class
 */
class MyTrackShowerIdAlgorithm : public pandora::Algorithm
{
public:
    /**
     *  @brief  Factory class for instantiating algorithm
     */
    class Factory : public pandora::AlgorithmFactory
    {
    public:
        pandora::Algorithm *CreateAlgorithm() const;
    };

    /**
     *  @brief  Default constructor
     */
    MyTrackShowerIdAlgorithm();

    /**
     *  @brief  Destructor
     */
    ~MyTrackShowerIdAlgorithm();

private:
    pandora::StatusCode Run();
    pandora::StatusCode ReadSettings(const pandora::TiXmlHandle xmlHandle);

    /**
     *  @brief  Whether cluster is identified as a clear track
     *
     *  @param  pCluster address of the relevant cluster
     *
     *  @return boolean
     */
    bool IsClearTrack(const pandora::Cluster *const pCluster) const;

    std::string     m_inputClusterListName;         ///< The input cluster list name
    bool            m_writeToTree;                  ///< Whether to write monitoring details to tree
    std::string     m_treeName;                     ///< Name of output tree
    std::string     m_fileName;                     ///< Name of output file
};
```

Note explicit destructor: point at which ROOT TTree is written.

Example Implementation

```
MyTrackShowerIdAlgorithm::MyTrackShowerIdAlgorithm() :
    m_writeToTree(false)
{
}

//------------------------------------------------------------------------------------------------------------------------

MyTrackShowerIdAlgorithm::~MyTrackShowerIdAlgorithm()
{
    if (m_writeToTree)
        PandoraMonitoringApi::SaveTree(this->GetPandora(), m_treeName.c_str(), m_fileName.c_str(), "UPDATE");
}

//------------------------------------------------------------------------------------------------------------------------

StatusCode MyTrackShowerIdAlgorithm::Run()
{
    const ClusterList *pClusterList(nullptr);
    PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::GetList(*this, m_inputClusterListName, pClusterList));

    ClusterVector sortedClusters(pClusterList->begin(), pClusterList->end());
    std::sort(sortedClusters.begin(), sortedClusters.end(), LArClusterHelper::SortByNHits);

    for (const Cluster *const pCluster : sortedClusters)
    {
        if (this->IsClearTrack(pCluster))
        {
            PandoraContentApi::Cluster::Metadata metadata;
            metadata.m_particleId = MU_MINUS;
            PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::AlterMetadata(*this, pCluster, metadata));
        }
        else
        {
            PandoraContentApi::Cluster::Metadata metadata;
            metadata.m_particleId = E_MINUS;
            PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::AlterMetadata(*this, pCluster, metadata));
        }
    }

    return STATUS_CODE_SUCCESS;
}
```

PandoraMonitoring manages ROOT TTree; request write upon alg destruction

Function that does all the work here; evaluate topological variables, fill tree, return decision

Can choose to alter particle id flag; metadata that can be attached to Cluster

IsClearTrack

```
bool MyTrackShowerIdAlgorithm::IsClearTrack(const Cluster *const pCluster) const
{
    int nHits(pCluster->GetNCaloHits()), isTrueTrack(-1);
    FloatVector xPositions, zPositions;

    try
    {
        // ATTN Slightly curious definition of a clear track, but this is most-likely what is needed for shower-growing
        const MCParticle *const pMCParticle(MCParticleHelper::GetMainMCParticle(pCluster));

        if ((PHOTON != pMCParticle->GetParticleId()) && (E_MINUS != std::abs(pMCParticle->GetParticleId())))
        {
            isTrueTrack = 1;
        }
        else
        {
            isTrueTrack = 0;
        }
    }
    catch (StatusCodeException &)
    {
        isTrueTrack = 0;
    }

    CaloHitList caloHitList;
    pCluster->GetOrderedCaloHitList().GetCaloHitList(caloHitList);

    for (const CaloHit *const pCaloHit : caloHitList)
    {
        xPositions.push_back(pCaloHit->GetPositionVector().GetX());
        zPositions.push_back(pCaloHit->GetPositionVector().GetZ());
    }

    PandoraMonitoringApi::SetTreeVariable(this->GetPandora(), m_treeName.c_str(), "nHits", nHits);
    PandoraMonitoringApi::SetTreeVariable(this->GetPandora(), m_treeName.c_str(), "isTrueTrack", isTrueTrack);
    PandoraMonitoringApi::SetTreeVariable(this->GetPandora(), m_treeName.c_str(), "xPositions", &xPositions);
    PandoraMonitoringApi::SetTreeVariable(this->GetPandora(), m_treeName.c_str(), "zPositions", &zPositions);
    PandoraMonitoringApi::FillTree(this->GetPandora(), m_treeName.c_str());

    // TODO return value motivated by selection on topological properties
    return true;
}
```

Default values for variables written to TTree

Access MCParticle information to determine target track/shower id

Store all Hit positions for each Cluster; can draw at ROOT command line

Name in TTree

Fill the TTree with in-scope int and FloatVector instances (no Boolean or unsigned support)

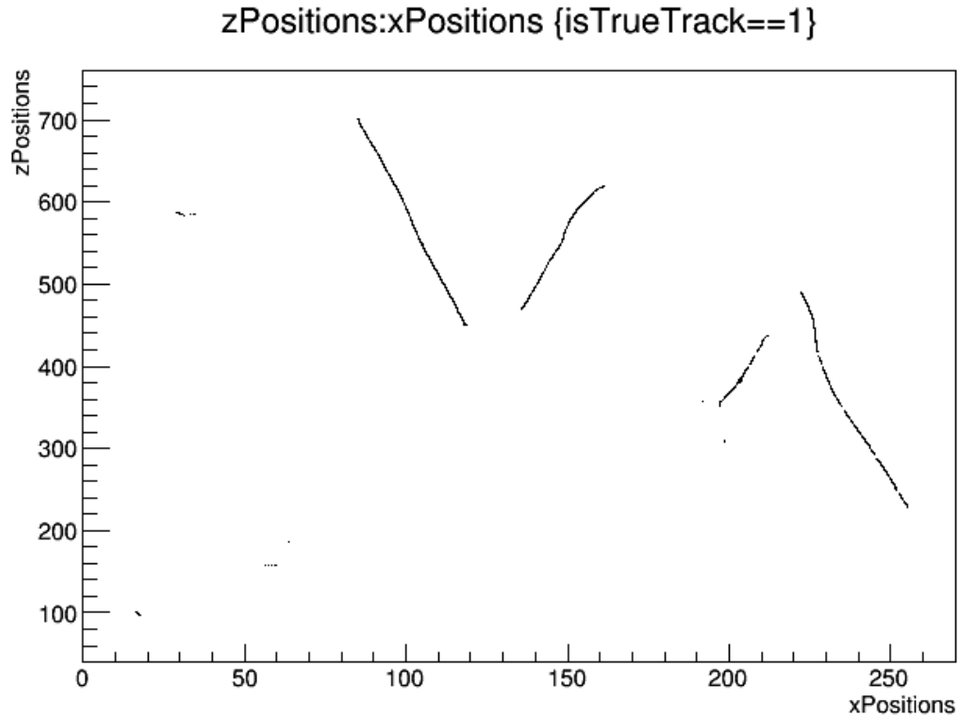
IsClearTrack

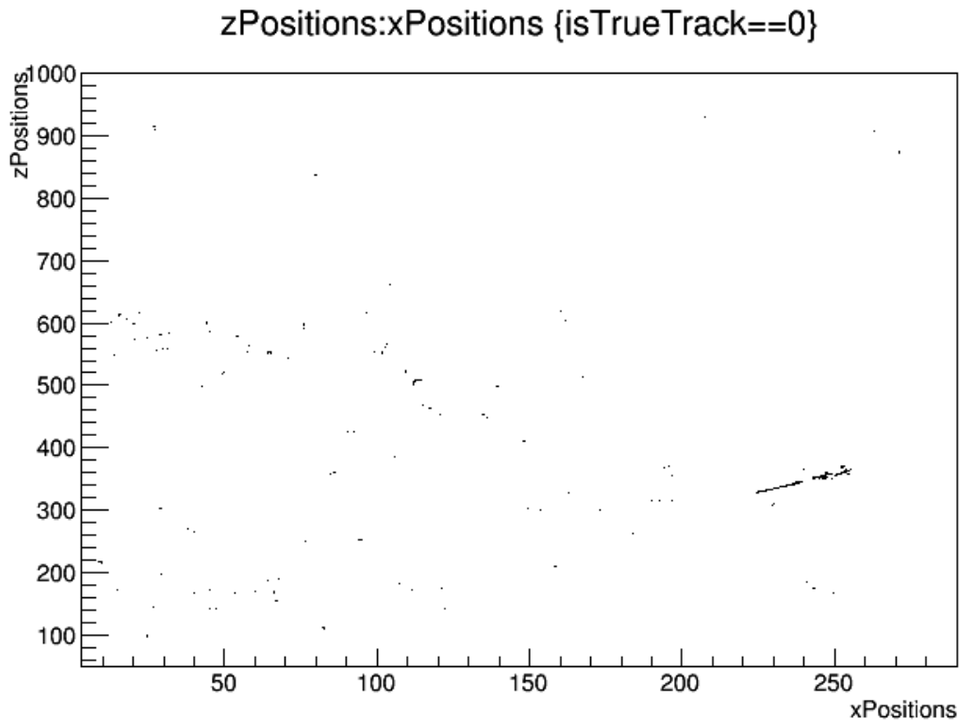
What's missing? More sophisticated variables, such as Cluster lengths and widths.
Try experimenting with something along the following lines, or just use the Hits as you see fit:

```
float straightLinePathLength(-1.f), widthSum(-1.f);

try
{
    // Sliding fit to i) entire Cluster ('shw'), ii) positive ('pos') and iii) negative ('neg') shower edges
    const float slidingFitPitch(LArGeometryHelper::GetWireZPitch(this->GetPandora()));
    const TwoDSlidingShowerFitResult showerFitResult(pCluster, m_slidingFitWindow, slidingFitPitch);

    const LayerFitResultMap &layerFitResultMapShw(showerFitResult.GetShowerFitResult().GetLayerFitResultMap());
    const LayerFitResultMap &layerFitResultMapPos(showerFitResult.GetPositiveEdgeFitResult().GetLayerFitResultMap());
    const LayerFitResultMap &layerFitResultMapNeg(showerFitResult.GetNegativeEdgeFitResult().GetLayerFitResultMap());

    // Use sliding fit to entire Cluster to define Cluster length
    CartesianVector globalMinLayerPositionOnAxis(0.f, 0.f, 0.f);
    showerFitResult.GetShowerFitResult().GetGlobalPosition(layerFitResultMapShw.begin()->second.GetL(), 0.f, globalMinLayerPositionOnAxis);

    CartesianVector globalMaxLayerPositionOnAxis(0.f, 0.f, 0.f);
    showerFitResult.GetShowerFitResult().GetGlobalPosition(layerFitResultMapShw.rbegin()->second.GetL(), 0.f, globalMaxLayerPositionOnAxis);

    straightLinePathLength = ((globalMaxLayerPositionOnAxis - globalMinLayerPositionOnAxis).GetMagnitude());

    // Use sliding fits to positive and negative shower edges to define Cluster width - query edge fits in layers defined by full Cluster fit
    widthSum = 0.f;

    for (const LayerFitResultMap::value_type &shwFitResultEntry : layerFitResultMapShw)
    {
        LayerFitResultMap::const_iterator iterPos = layerFitResultMapPos.find(shwFitResultEntry.first);
        LayerFitResultMap::const_iterator iterNeg = layerFitResultMapNeg.find(shwFitResultEntry.first);

        if ((layerFitResultMapPos.end() == iterPos) || (layerFitResultMapNeg.end() == iterNeg))
            continue;

        widthSum += std::fabs(iterPos->second.GetFitT() - iterNeg->second.GetFitT());
    }
}
catch (const StatusCodeException &statusCodeException)
{
    std::cout << "MyTrackShowerIdAlgorithm::IsClearTrack: " << statusCodeException.ToString() << std::endl;
}

PandoraMonitoringApi::SetTreeVariable(this->GetPandora(), m_treeName.c_str(), "straightLinePathLength", straightLinePathLength);
PandoraMonitoringApi::SetTreeVariable(this->GetPandora(), m_treeName.c_str(), "widthSum", widthSum);
PandoraMonitoringApi::FillTree(this->GetPandora(), m_treeName.c_str());
```

Investigation of ROOT TTree

```
bash-3.2$ root -l MyTrackShowerId.root

root [0]
Attaching file MyTrackShowerId.root as _file0...
root [1] .ls
TFile**             MyTrackShowerId.root
 TFile*         MyTrackShowerId.root
  KEY: TTree    MyTree;1        MyTree
root [2] MyTree->Show(0)
======> EVENT:0
 nHits           = 272
 isTrueTrack     = 1
 xPositions      = (vector<float>*)0x7f9ce2e667d0
 zPositions      = (vector<float>*)0x7f9ce2e67220
 straightLinePathLength = 87.3
 widthSum        = 24.5155
root [3] MyTree->Draw("zPositions:xPositions", "isTrueTrack==1")
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(Long64_t)2443
root [4] MyTree->Draw("zPositions:xPositions", "isTrueTrack==0")
(Long64_t)459
```





Next Exercise: Work through the ExampleContent Library