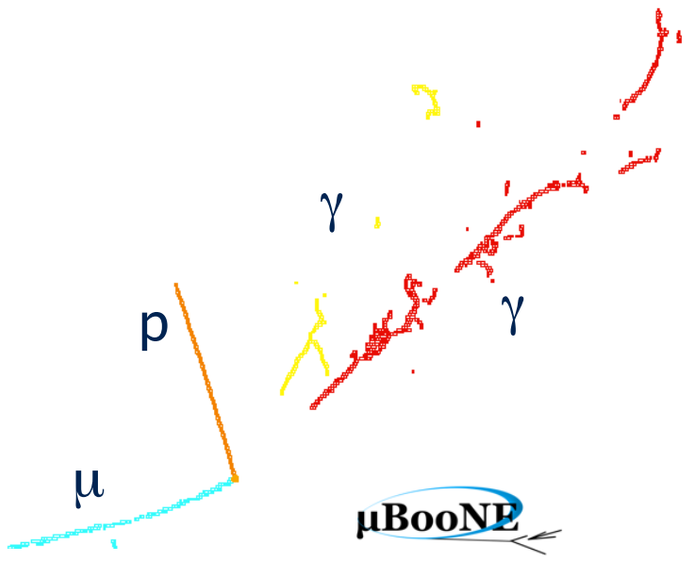


Pandora Exercise 7: Particle Merging

J. S. Marshall for the Pandora Team

MicroBooNE Pandora Workshop

July 11-14th 2016, Cambridge

Particle Merging

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.

Pre-requisite: Exercise 4 - write an algorithm to perform Cluster merging.

Particle Merging

Particle Merging is one of the hardest operations to perform in Pandora.

Particles are containers of multiple objects (Clusters, Vertices, daughter Particles, etc.).

When Particles are merged, decisions must be made about how to handle this.

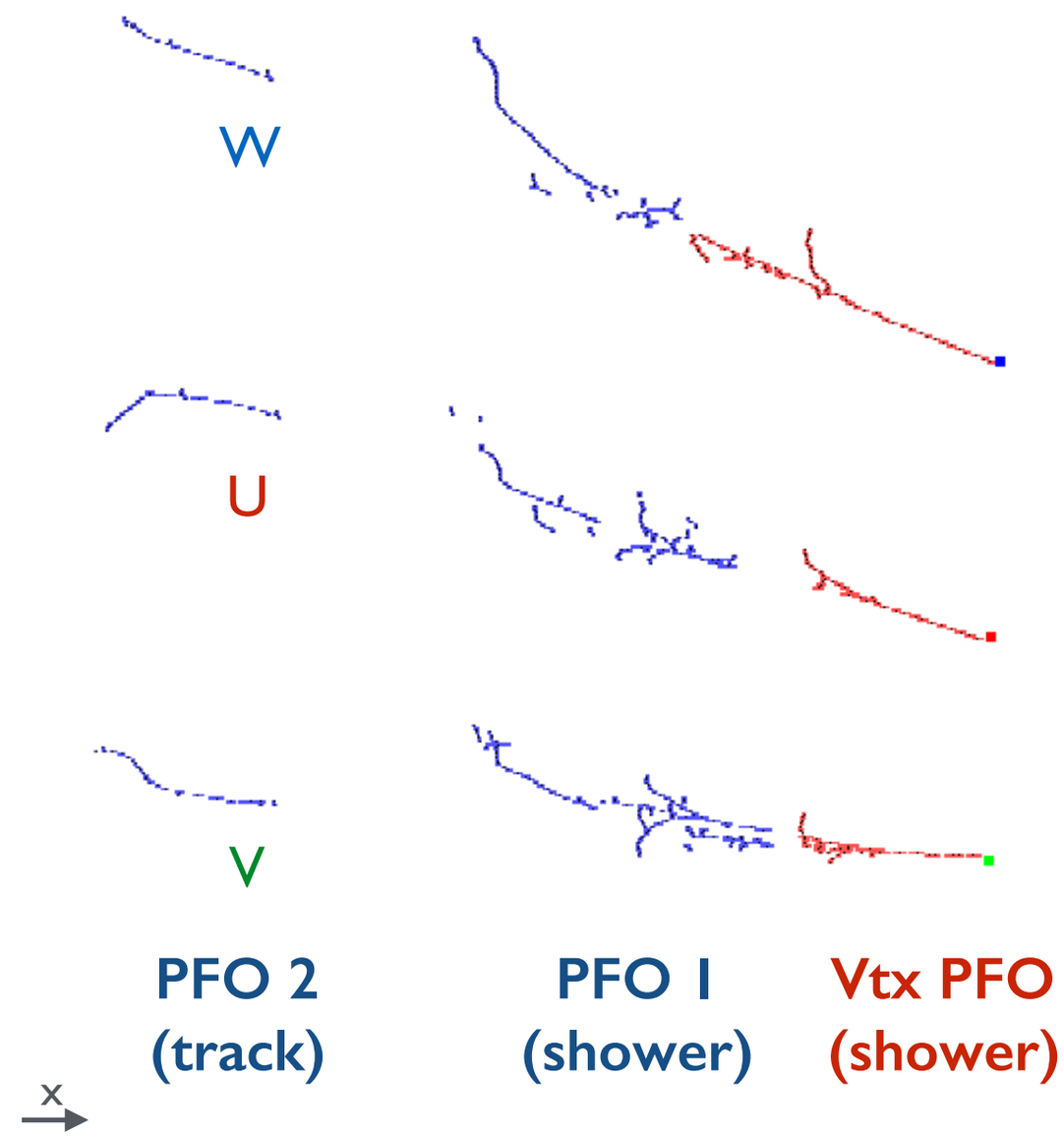
The objects contained in a Particle will likely be spread between multiple Pandora lists.

Object deletion also necessarily means algs must take care with local dangling pointers.

This document introduces a reasonably clean way to deal with Particle merging.

Logic to decide when to merge Particles, and which to enlarge or delete is up to you!

Example use-case: sparse showers reconstructed as multiple Particles:



PandoraSettings

Suggest that you start with `PandoraSettings_MicroBooNE_SingleNeutrino.xml` and add a new `MyParticleMerging` algorithm just before the neutrino/event building algs:

```
...

<!-- 3D hit reconstruction -->
<algorithm type = "LArThreeDHitCreation">
    <InputPfoListName>TrackParticles3D</InputPfoListName>
    <OutputCaloHitListName>TrackCaloHitList3D</OutputCaloHitListName>
    <OutputClusterListName>TrackClusters3D</OutputClusterListName>
    <HitCreationTools>
        <tool type = "LArClearTransverseTrackHits"><MinViews>3</MinViews></tool>
        <tool type = "LArClearLongitudinalTrackHits"><MinViews>3</MinViews></tool>
        <tool type = "LArMultiValuedLongitudinalTrackHits"><MinViews>3</MinViews></tool>
        <tool type = "LArMultiValuedTransverseTrackHits"><MinViews>3</MinViews></tool>
        <tool type = "LArClearTransverseTrackHits"><MinViews>2</MinViews></tool>
        <tool type = "LArClearLongitudinalTrackHits"><MinViews>2</MinViews></tool>
        <tool type = "LArMultiValuedLongitudinalTrackHits"><MinViews>2</MinViews></tool>
    </HitCreationTools>
</algorithm>
<algorithm type = "LArThreeDHitCreation">
    <InputPfoListName>ShowerParticles3D</InputPfoListName>
    <OutputCaloHitListName>ShowerCaloHitList3D</OutputCaloHitListName>
    <OutputClusterListName>ShowerClusters3D</OutputClusterListName>
    <HitCreationTools>
        <tool type = "LArThreeViewShowerHits"/>
        <tool type = "LArTwoViewShowerHits"/>
        <tool type = "LArDeltaRayShowerHits"/>
    </HitCreationTools>
</algorithm>

<!-- New Particle Merging attempts -->
<algorithm type = "MyParticleMerging">
    <InputPfoListNames>TrackParticles3D ShowerParticles3D</InputPfoListNames>
    <DaughterListNames>ClustersU ClustersV ClustersW TrackClusters3D ShowerClusters3D</DaughterListNames>
</algorithm>

<!-- Construct neutrino and daughter pfo hierarchy -->
<algorithm type = "LArNeutrinoCreation">
   <InputVertexListName>NeutrinoVertices3D</InputVertexListName>
   <NeutrinoPfoListName>NeutrinoParticles3D</NeutrinoPfoListName>
</algorithm>

...
```

Example Implementation

Key pieces of logic to provide (currently merges all top-level Pfos). Note warnings about dangling pointers after Pfo deletion. Here using local copy of manager-owned Pfo lists.

```
StatusCode MyParticleMergingAlgorithm::Run()
{
    PfoVector sortedPfos;
    this->GetSortedPfos(sortedPfos);

    for (const ParticleFlowObject *const pPfo1 : sortedPfos)
    {
        for (const ParticleFlowObject *const pPfo2 : sortedPfos)
        {
            // ATTN Check whether vector contents (size must remain fixed) is changing underneath; consider only top-level pfos
            if (!pPfo1 || !pPfo2 || (pPfo1 == pPfo2) || !pPfo1->GetParentPfoList().empty() || !pPfo2->GetParentPfoList().empty())
                continue;

            // TODO Add key pattern-recognition decision #1 here. Write a function to return this boolean.
            const bool shouldMergePfos(true);

            if (shouldMergePfos)
            {
                // TODO Add key pattern-recognition decision #2 here. Write a function to decide which pfo to enlarge.
                const ParticleFlowObject *const pPfoToEnlarge(pPfo1);
                const ParticleFlowObject *const pPfoToDelete(pPfo2);

                PfoList enlargePfoList, deletePfoList;
                enlargePfoList.insert(pPfoToEnlarge); deletePfoList.insert(pPfoToDelete);

                PandoraMonitoringApi::SetEveDisplayParameters(this->GetPandora(), false, DETECTOR_VIEW_XZ, -1.f, -1.f, 1.f);
                PandoraMonitoringApi::VisualizeParticleFlowObjects(this->GetPandora(), &enlargePfoList, "PfoToEnlarge", BLUE);
                PandoraMonitoringApi::VisualizeParticleFlowObjects(this->GetPandora(), &deletePfoList, "PfoToDelete", GREEN);
                PandoraMonitoringApi::ViewEvent(this->GetPandora());

                // ATTN Remove what will shortly become a dangling pointer from the container
                std::replace(sortedPfos.begin(), sortedPfos.end(), pPfoToDelete, static_cast<const ParticleFlowObject *>(nullptr));
                this->MergeAndDeletePfos(pPfoToEnlarge, pPfoToDelete);

                PfoList mergedPfoList;
                mergedPfoList.insert(pPfoToEnlarge);
                PandoraMonitoringApi::VisualizeParticleFlowObjects(this->GetPandora(), &mergedPfoList, "MergedPfo", RED);
                PandoraMonitoringApi::ViewEvent(this->GetPandora());
            }
        }
    }

    return STATUS_CODE_SUCCESS;
}
```

GetSortedPfos

Nothing particularly new in this implementation. Adds all Pfos in all specified input lists to a single vector, then sorts container to provide well-defined input state.

MyParticleMergingAlgorithm.h

```
/**
 *  @brief  Get the sorted list of input pfos
 *
 *  @param  sortedPfos to receive the sorted list of input pfos
 */
void GetSortedPfos(pandora::PfoVector &sortedPfos) const;
```

MyParticleMergingAlgorithm.cc

```
void MyParticleMergingAlgorithm::GetSortedPfos(PfoVector &sortedPfos) const
{
    for (const std::string &listName : m_inputPfoListNames)
    {
        const PfoList *pPfoList(nullptr);

        if (STATUS_CODE_SUCCESS == PandoraContentApi::GetList(*this, listName, pPfoList))
            sortedPfos.insert(sortedPfos.end(), pPfoList->begin(), pPfoList->end());
    }

    std::sort(sortedPfos.begin(), sortedPfos.end(), LArPfoHelper::SortByNHits);
}
```

MergeAndDeletePfos

Logic for Pfo merges; read carefully and note use of GetListName and GetParentCluster.

```
/**
 *  @brief  Merge and delete a pair of pfos, with a specific set of conventions for cluster merging, vertex use, etc.
 *
 *  @param  pPfoToEnlarge the address of the pfo to enlarge
 *  @param  pPfoToDelete the address of the pfo to delete (will become a dangling pointer)
 */
void MergeAndDeletePfos(const pandora::ParticleFlowObject *const pPfoToEnlarge, const pandora::ParticleFlowObject *const pPfoToDelete) const;
```

MyParticleMergingAlgorithm.h

```
void MyParticleMergingAlgorithm::MergeAndDeletePfos(const ParticleFlowObject *const pPfoToEnlarge, const ParticleFlowObject *const pPfoToDelete) const
{
    const PfoList daughterPfos(pPfoToDelete->GetDaughterPfoList());
    const ClusterVector daughterClusters(pPfoToDelete->GetClusterList().begin(), pPfoToDelete->GetClusterList().end());
    const VertexVector daughterVertices(pPfoToDelete->GetVertexList().begin(), pPfoToDelete->GetVertexList().end());

    PANDORA_THROW_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::Delete(*this, pPfoToDelete, this->GetListName(pPfoToDelete)));

    for (const ParticleFlowObject *const pDaughterPfo : daughterPfos)
    {
        PANDORA_THROW_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::SetPfoParentDaughterRelationship(*this, pPfoToEnlarge, pDaughterPfo));
    }

    for (const  Vertex *const pDaughterVertex : daughterVertices)
    {
        PANDORA_THROW_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::Delete(*this, pDaughterVertex, this->GetListName(pDaughterVertex)));
    }

    for (const Cluster *const pDaughterCluster : daughterClusters)
    {
        const HitType daughterHitType(LArClusterHelper::GetClusterHitType(pDaughterCluster));
        const Cluster *pParentCluster(this->GetParentCluster(pPfoToEnlarge->GetClusterList(), daughterHitType));

        if (pParentCluster)
        {
            PANDORA_THROW_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::MergeAndDeleteClusters(*this, pParentCluster, pDaughterCluster,
                this->GetListName(pParentCluster), this->GetListName(pDaughterCluster)));
        }
        else
        {
            PANDORA_THROW_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::AddToPfo(*this, pPfoToEnlarge, pDaughterCluster));
        }
    }
}
```

MyParticleMergingAlgorithm.cc

GetParentCluster

Used by MergeAndDeletePfos, choose most appropriate parent Cluster (in PfoToEnlarge) with which daughter Cluster (in PfoToDelete) should be merged.

MyParticleMergingAlgorithm.h

```
/**
 *  @brief  Select the parent cluster (same hit type and most hits) using a provided cluster list and hit type
 *
 *  @param  clusterList the cluster list
 *  @param  hitType the hit type
 *
 *  @return the address of the parent cluster
 */
const pandora::Cluster *GetParentCluster(const pandora::ClusterList &clusterList, const pandora::HitType hitType) const;
```

MyParticleMergingAlgorithm.cc

```
const Cluster *MyParticleMergingAlgorithm::GetParentCluster(const ClusterList &clusterList, const HitType hitType) const
{
    unsigned int mostHits(0);
    const Cluster *pBestParentCluster(nullptr);

    for (const Cluster *const pParentCluster : clusterList)
    {
        if (hitType != LArClusterHelper::GetClusterHitType(pParentCluster))
            continue;

        const unsigned int nParentHits(pParentCluster->GetNCaloHits());

        if (nParentHits > mostHits)
        {
            mostHits = nParentHits;
            pBestParentCluster = pParentCluster;
        }
    }

    return pBestParentCluster;
}
```

GetListName

Rather clean (if not efficient) solution to keeping track of manager-owned lists hosting each of the different objects associated to the Pfo. XML config provides list of relevant list names.

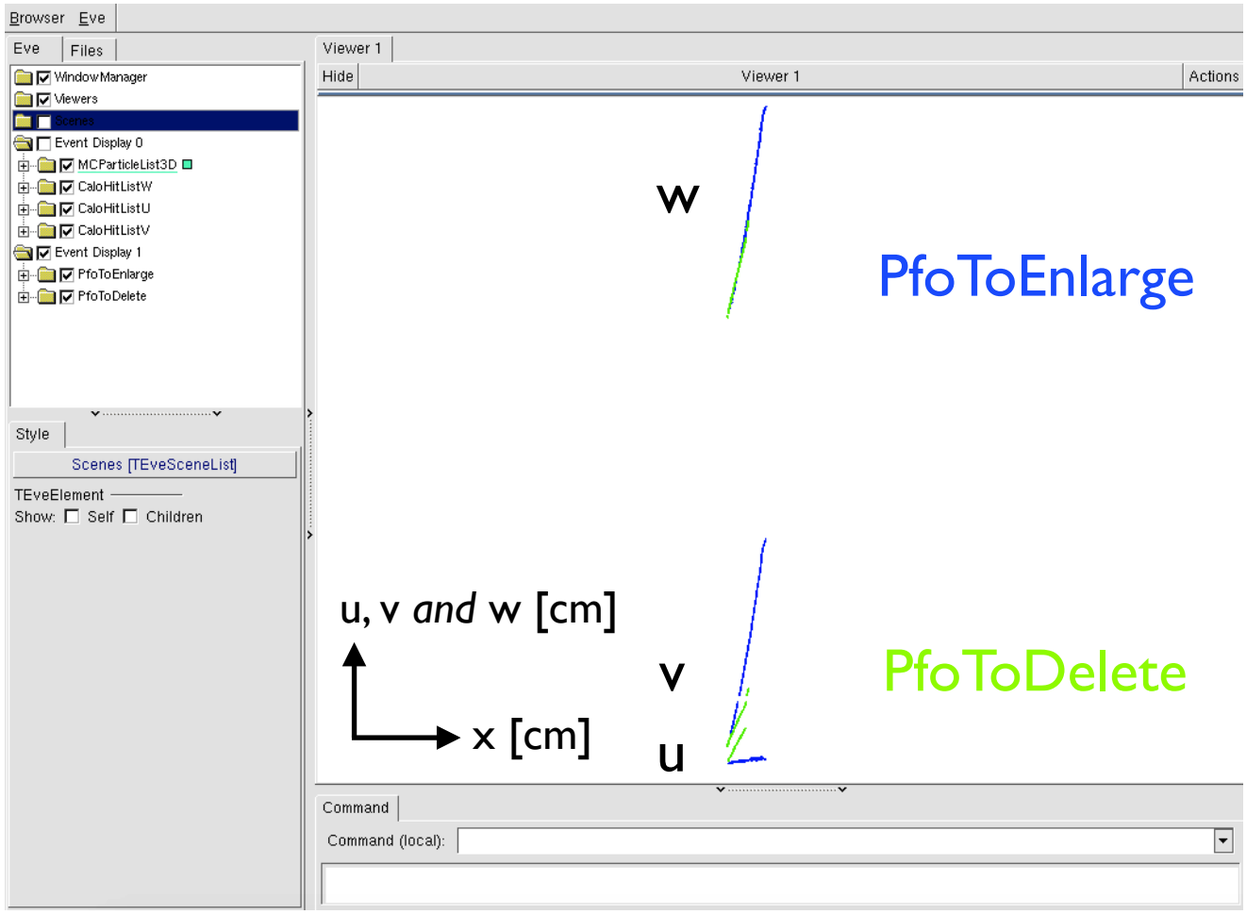
MyParticleMergingAlgorithm.h

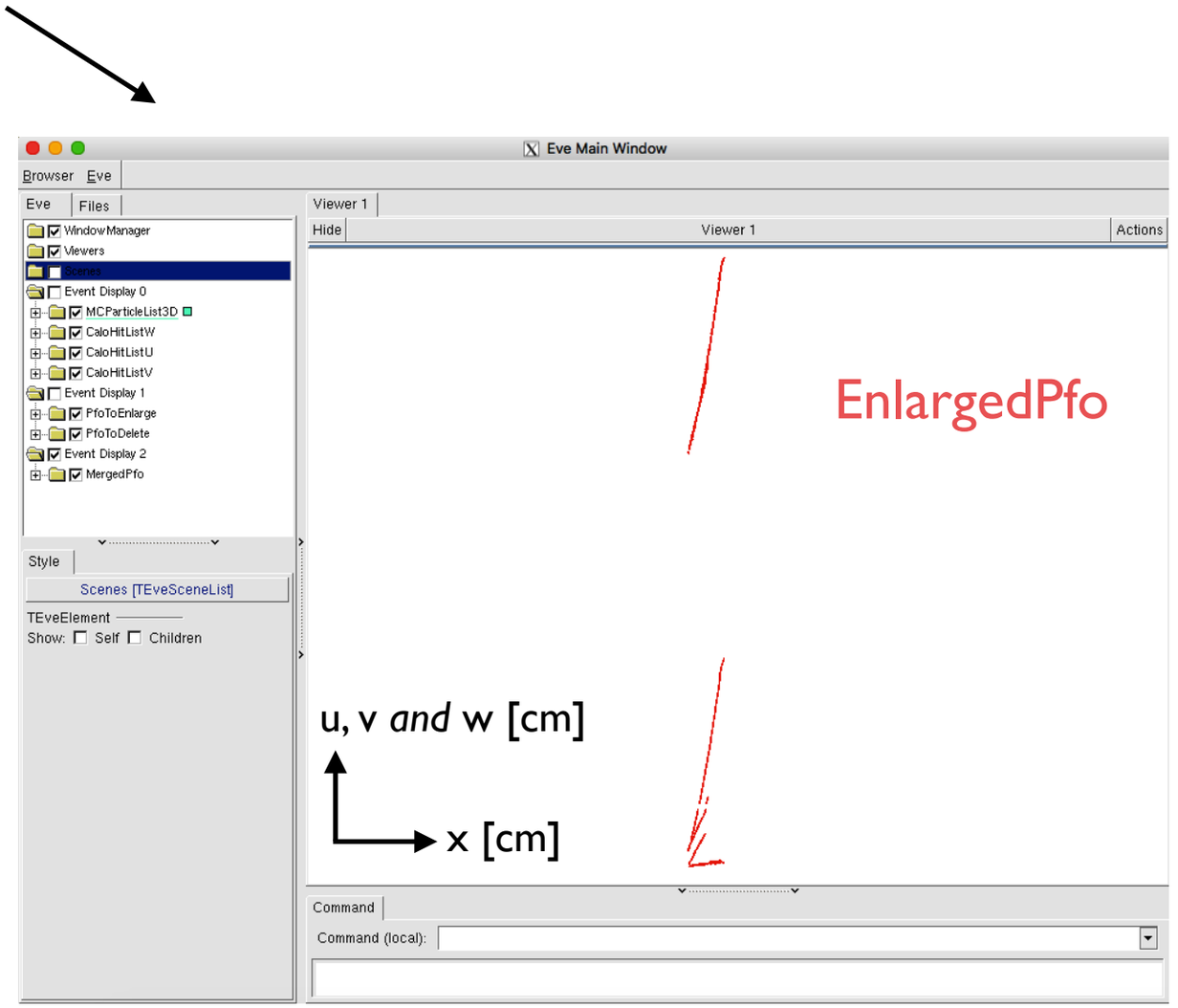
```
/**
 *  @brief  Find the name of the list hosting a specific object, by searching contents of named daughter lists
 *
 *  @param  pT the address of the object
 *
 *  @return the name of the list
 */
template <typename T>
const std::string GetListName(const T *const pT) const;

pandora::StringVector   m_inputPfoListNames;    ///< The list of input pfo list names
pandora::StringVector   m_daughterListNames;    ///< The list of daughter list names (input pfo list names will be added to this list too)
```

MyParticleMergingAlgorithm.cc

```
template <typename T>
const std::string MyParticleMergingAlgorithm::GetListName(const T *const pT) const
{
    std::string currentListName;
    const std::MANAGED_CONTAINER<const T*> *pCurrentList(nullptr);
    (void) PandoraContentApi::GetCurrentList(*this, pCurrentList, currentListName);

    if (pCurrentList && (pCurrentList->count(pT)))
        return currentListName;

    for (const std::string &listName : m_daughterListNames)
    {
        const std::MANAGED_CONTAINER<const T*> *pList(nullptr);
        (void) PandoraContentApi::GetList(*this, listName, pList);

        if (pList && (pList->count(pT)))
            return listName;
    }

    throw StatusCodeException(STATUS_CODE_NOT_FOUND);
}
```

Example Visualisation





Next Exercise: Perform Track vs. Shower Identification