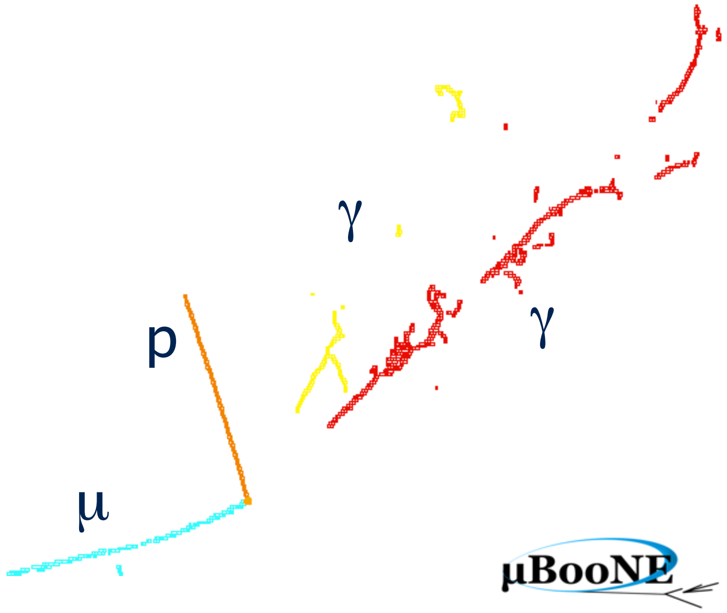


Pandora Exercise 4: Cluster Merging

J. S. Marshall for the Pandora Team

MicroBooNE Pandora Workshop

July 11-14th 2016, Cambridge

Cluster Merging

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.

Create a new algorithm to merge Clusters representing same particles:

- Examples of Cluster merging code, with associated lessons for careful use
- Visual debugging
- Start to assess Cluster purity and completeness

Add MyClusterMerging Algorithm

- Add a new algorithm, with a registered name such as “MyClusterMerging” (finding the right name for algs is an important part of managing a multi-algorithm approach!).
- The input to this new algorithm will be a list of Clusters. These could be formed by your earlier Clustering test algorithm, or you could choose to use an existing Pandora algorithm:

```
# Don’t forget you’ll need to re-run CMake after adding a new source file
```

```
<algorithm type = "LArClusteringParent">
    <algorithm type = "LArTrackClusterCreation" description = "ClusterFormation"/>
    <InputCaloHitListName>CaloHitListW</InputCaloHitListName>
    <ClusterListName>MyFirstClustersW</ClusterListName>
    <ReplaceCurrentCaloHitList>false</ReplaceCurrentCaloHitList>
    <ReplaceCurrentClusterList>true</ReplaceCurrentClusterList>
</algorithm>

<algorithm type = "LArVisualMonitoring">
    <CaloHitListNames>CaloHitListW</CaloHitListNames>
    <ClusterListNames>MyFirstClustersW</ClusterListNames>
</algorithm>

<algorithm type = "MyClusterMerging"/>

<algorithm type = "LArVisualMonitoring">
    <ClusterListNames>MyFirstClustersW</ClusterListNames>
</algorithm>
```

← LAr TPC TrackClusterCreation algorithm

← Cluster visualisation *before* merging alg

← New Cluster merging algorithm

← Cluster visualisation *after* merging alg

An Example Implementation

Merging of 2D Clusters from a single named list:

PandoraSettings_Workshop.xml

```
<algorithm type = "MyClusterMerging">
    <InputClusterListName>MyFirstClustersW</InputClusterListName>
</algorithm>
```

MyClusterMergingAlgorithm.cc

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

    ClusterVector sortedLongClusters;
    this->GetSortedLongClusters(pClusterList, sortedLongClusters);

    ClusterList defunctClusters;

    for (const Cluster *const pParentCluster : sortedLongClusters)
    {
        if (defunctClusters.count(pParentCluster))
            continue;

        for (const Cluster *const pDaughterCluster : sortedLongClusters)
        {
            if ((pParentCluster == pDaughterCluster) || defunctClusters.count(pDaughterCluster))
                continue;

            if (!this->AreClustersAssociated(pParentCluster, pDaughterCluster))
                continue;

            PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::MergeAndDeleteClusters(*this, pParentCluster, pDaughterCluster));
            (void) defunctClusters.insert(pDaughterCluster);
        }
    }

    return STATUS_CODE_SUCCESS;
}
```

Now look at this in some detail…

An Example Implementation

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

    ClusterVector sortedLongClusters;
    this->GetSortedLongClusters(pClusterList, sortedLongClusters);

    ClusterList defunctClusters;

    for (const Cluster *const pParentCluster : sortedLongClusters)
    {
        if (defunctClusters.count(pParentCluster))
            continue;

        for (const Cluster *const pDaughterCluster : sortedLongClusters)
        {
            if ((pParentCluster == pDaughterCluster) || defunctClusters.count(pDaughterCluster))
                continue;

            if (!this->AreClustersAssociated(pParentCluster, pDaughterCluster))
                continue;

            PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::MergeAndDeleteClusters(*this, pParentCluster, pDaughterCluster));
            (void) defunctClusters.insert(pDaughterCluster);
        }
    }

    return STATUS_CODE_SUCCESS;
}
```

Choose to read input Cluster list name via XML, then ask for named list.

Filter to focus on interesting Clusters, then sort local vector to put in well-defined state.

An Example Implementation

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

    ClusterVector sortedLongClusters;
    this->GetSortedLongClusters(pClusterList, sortedLongClusters);
```

```
/**
 *  @brief  Examine an input cluster list, providing a sorted container of long clusters
 *
 *  @param  pClusterList the address of the input cluster list
 *  @param  sortedLongClusters to receive the sorted vector of long clusters
 */
void GetSortedLongClusters(const pandora::ClusterList *const pClusterList, pandora::ClusterVector &sortedLongClusters) const;



void MyClusterMergingAlgorithm::GetSortedLongClusters(const ClusterList *const pClusterList, ClusterVector &sortedLongClusters) const
{
    for (const Cluster *const pCluster : *pClusterList)
    {
        if (pCluster->GetNCaloHits() > m_minClusterCaloHits)
            sortedLongClusters.push_back(pCluster);
    }

    std::sort(sortedLongClusters.begin(), sortedLongClusters.end(), LArClusterHelper::SortByNHits);
}
```

Choose to read input Cluster list name via XML, then ask for named list.

Filter to focus on interesting Clusters, then sort local vector to put in well-defined state.

An Example Implementation

A *local* problem:

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

    ClusterVector sortedLongClusters;
    this->GetSortedLongClusters(pClusterList, sortedLongClusters);

    ClusterList defunctClusters;

    for (const Cluster *const pParentCluster : sortedLongClusters)
    {
        if (defunctClusters.count(pParentCluster))
            continue;

        for (const Cluster *const pDaughterCluster : sortedLongClusters)
        {
            if ((pParentCluster == pDaughterCluster) || defunctClusters.count(pDaughterCluster))
                continue;

            if (!this->AreClustersAssociated(pParentCluster, pDaughterCluster))
                continue;

            PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::MergeAndDeleteClusters(*this, pParentCluster, pDaughterCluster));
            (void) defunctClusters.insert(pDaughterCluster);
        }
    }

    return STATUS_CODE_SUCCESS;
}
```

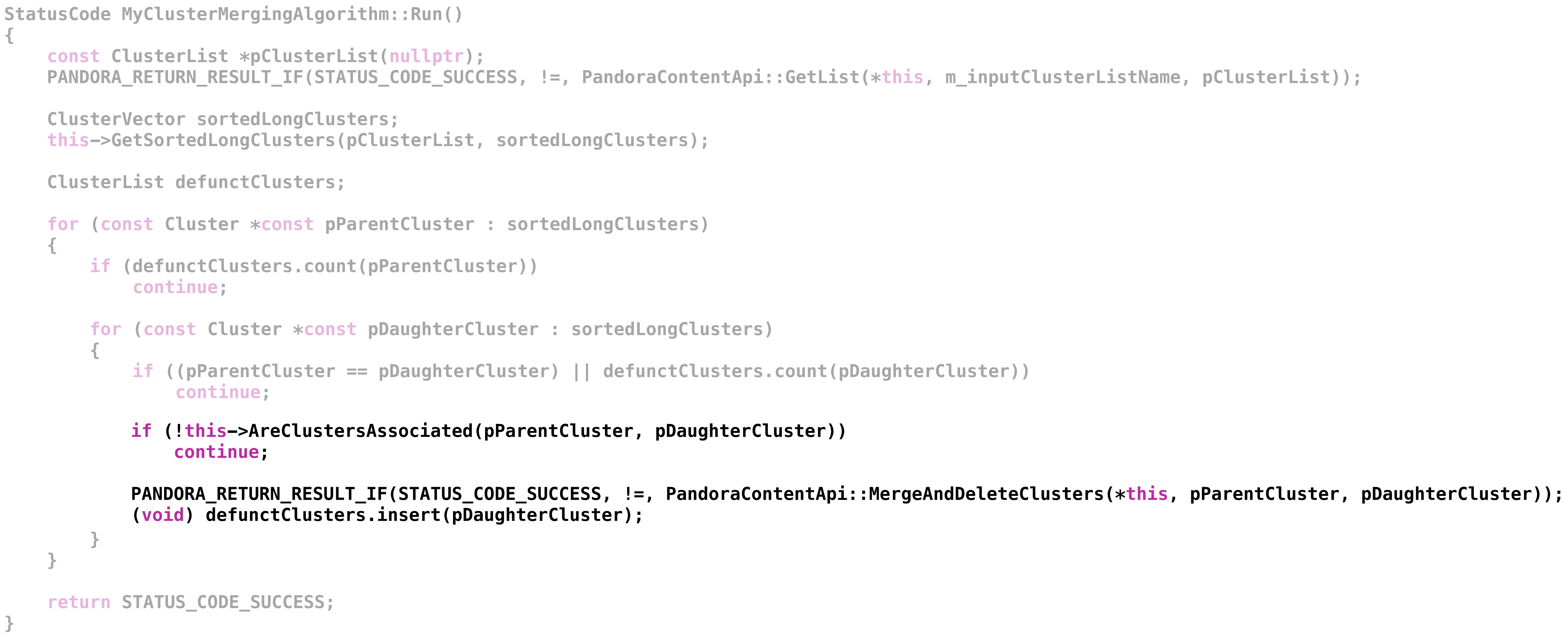
One approach: Keep track of local copies of dangling pointers

Or: check whether Cluster is still present in the manager-owned list, etc.

Care required in algs that delete Clusters from the manager-owned list. Likely that algorithm will hold local copy of addresses of Clusters, or may be iterating over the manager-owned list itself.

Don’t deference local copy of a now-dangling pointer. If iterating over the manager-owned list itself (unordered_set) note that any iterators pointing to a deleted element will be invalidated.

An Example Implementation

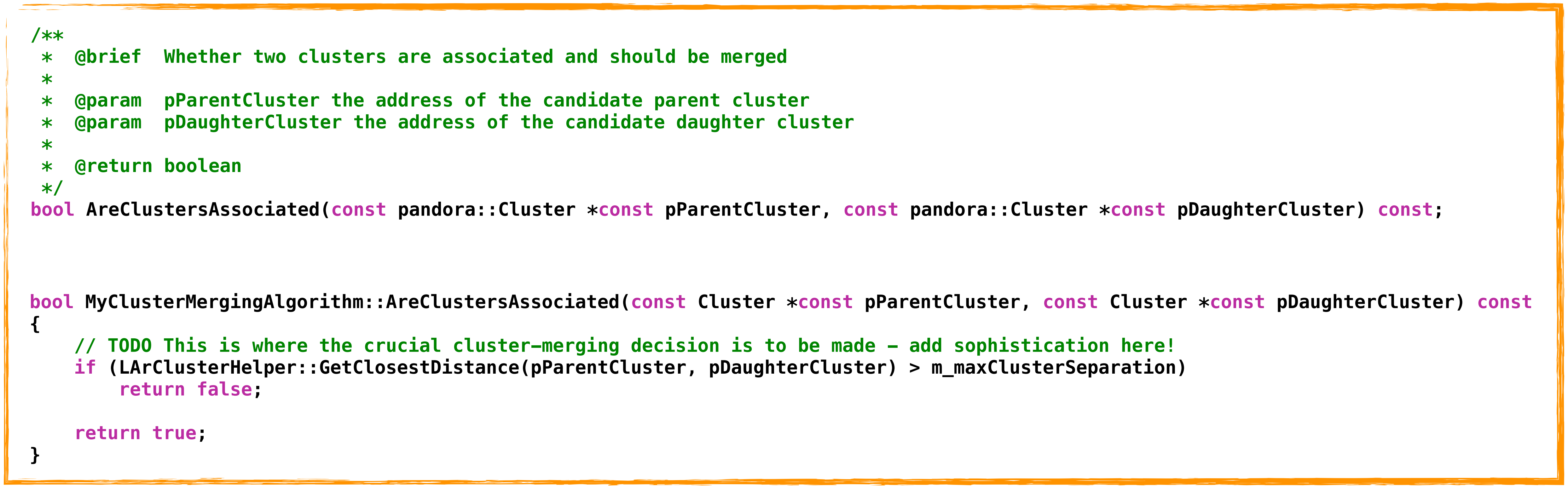


```
StatusCode MyClusterMergingAlgorithm::Run()
{
    const ClusterList *pClusterList(nullptr);
    PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::GetList(*this, m_inputClusterListName, pClusterList));

    ClusterVector sortedLongClusters;
    this->GetSortedLongClusters(pClusterList, sortedLongClusters);

    ClusterList defunctClusters;

    for (const Cluster *const pParentCluster : sortedLongClusters)
    {
        if (defunctClusters.count(pParentCluster))
            continue;

        for (const Cluster *const pDaughterCluster : sortedLongClusters)
        {
            if ((pParentCluster == pDaughterCluster) || defunctClusters.count(pDaughterCluster))
                continue;

            if (!this->AreClustersAssociated(pParentCluster, pDaughterCluster))
                continue;

            PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::MergeAndDeleteClusters(*this, pParentCluster, pDaughterCluster));
            (void) defunctClusters.insert(pDaughterCluster);
        }
    }

    return STATUS_CODE_SUCCESS;
}
```

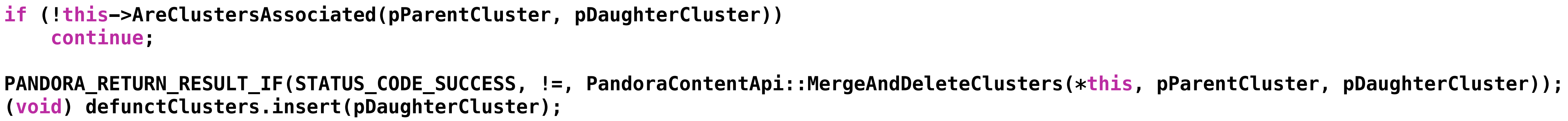
Key logic to be implemented to provide decision as to whether two Clusters are associated.

API call then requests that Cluster merge is actioned. Keep track of the now-dangling pointer!

An Example Implementation



```
/**
 *  @brief  Whether two clusters are associated and should be merged
 *
 *  @param  pParentCluster the address of the candidate parent cluster
 *  @param  pDaughterCluster the address of the candidate daughter cluster
 *
 *  @return boolean
 */
bool AreClustersAssociated(const pandora::Cluster *const pParentCluster, const pandora::Cluster *const pDaughterCluster) const;



bool MyClusterMergingAlgorithm::AreClustersAssociated(const Cluster *const pParentCluster, const Cluster *const pDaughterCluster) const
{
    // TODO This is where the crucial cluster-merging decision is to be made - add sophistication here!
    if (LArClusterHelper::GetClosestDistance(pParentCluster, pDaughterCluster) > m_maxClusterSeparation)
        return false;

    return true;
}
```

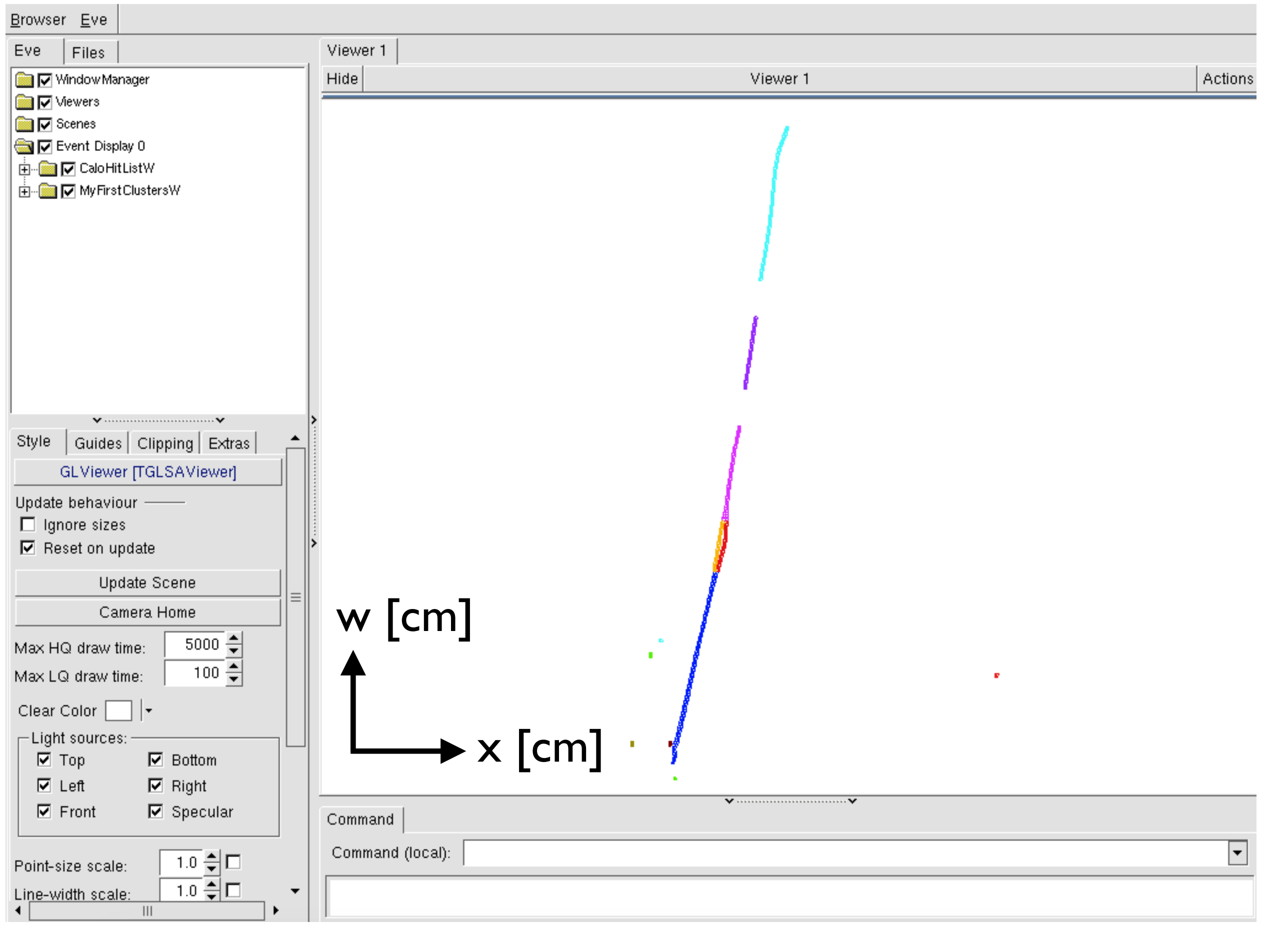


```
if (!this->AreClustersAssociated(pParentCluster, pDaughterCluster))
    continue;

PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::MergeAndDeleteClusters(*this, pParentCluster, pDaughterCluster));
(void) defunctClusters.insert(pDaughterCluster);
```

Key logic to be implemented to provide decision as to whether two Clusters are associated.

API call then requests that Cluster merge is actioned. Keep track of the now-dangling pointer!

An Example Implementation



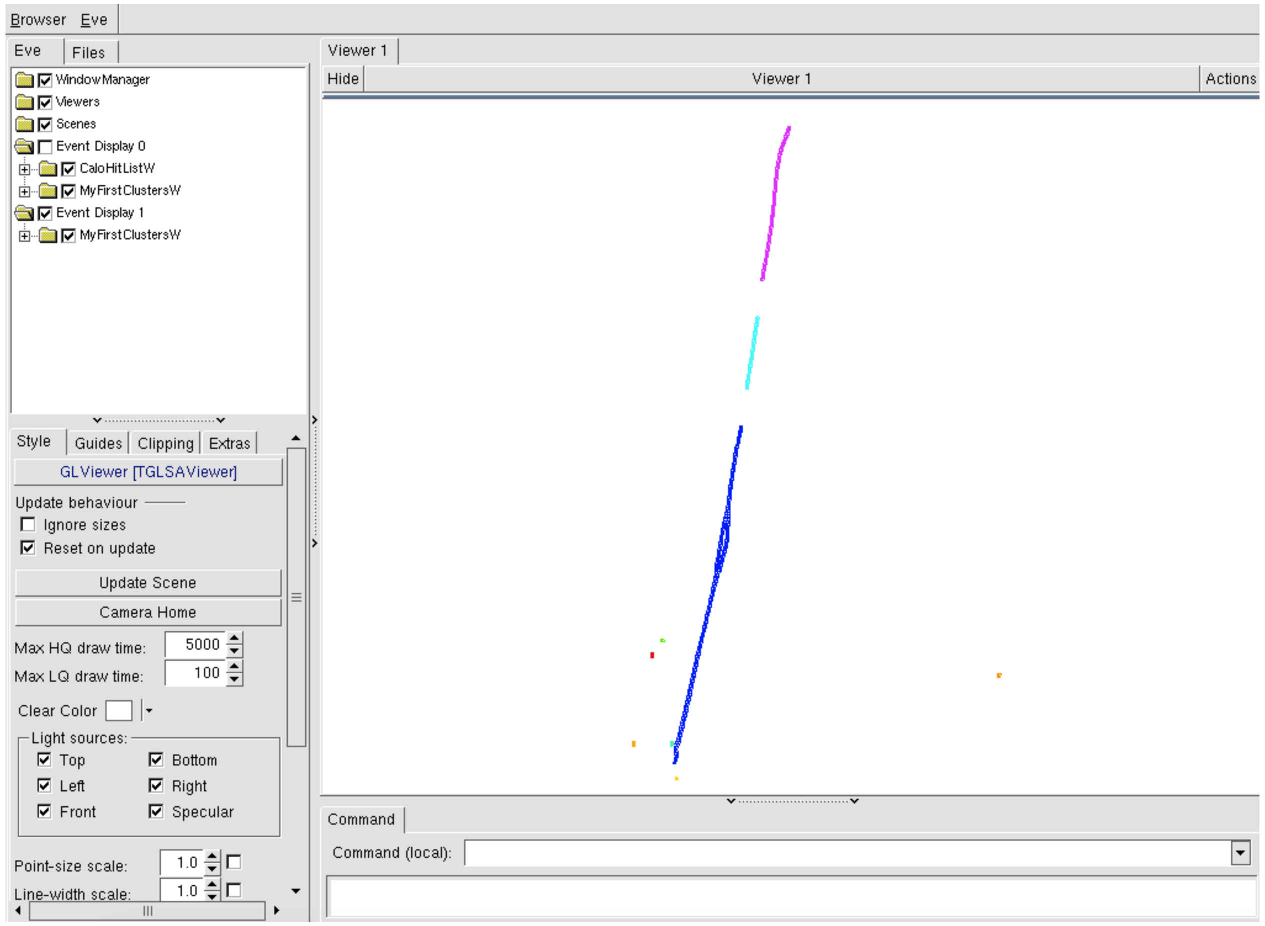
☑ WindowManager
☑ Viewers
☑ Scenes
☑ Event Display 0
☑ CaloHitListW
☑ MyFirstClustersW

Update behaviour
☐ Ignore sizes
☑ Reset on update

Max HQ draw time: 5000
Max LQ draw time: 100
Clear Color

Light sources:
☑ Top ☑ Bottom
☑ Left ☑ Right
☑ Front ☑ Specular

Point-size scale: 1.0 ☐
Line-width scale: 1.0 ☐



☑ WindowManager
☑ Viewers
☑ Scenes
☐ Event Display 0
☑ CaloHitListW
☑ MyFirstClustersW
☑ Event Display 1
☑ MyFirstClustersW

Update behaviour
☐ Ignore sizes
☑ Reset on update

Max HQ draw time: 5000
Max LQ draw time: 100
Clear Color

Light sources:
☑ Top ☑ Bottom
☑ Left ☑ Right
☑ Front ☑ Specular

Point-size scale: 1.0 ☐
Line-width scale: 1.0 ☐

Note the Cluster merges requested between the two LArVisualMonitoring algorithm instances

Different Approaches

- **Once merge has been identified, alg can choose whether to request merge immediately:**
  - In a dynamic approach, merges are made during the algorithm pattern-recognition operations: newly-enlarged Clusters can then feature in later algorithm operations and can grow again.
  - In a static approach, descriptions of individual merges are stored and then actioned together at the end of the algorithm (chains of merges can also be requested in this way).

- **Example implementation shows just one of a number of common variants for these algs:**
  - Loop over all parent/daughter combinations, choose best combination, merge then repeat (can choose to include newly-enlarged Cluster as a candidate parent and/or daughter).
  - For a given parent, loop over all daughters, find best combination, merge then repeat (can choose whether to repeat daughter loop for newly-enlarged parent or to just move on).
  - For a given parent, loop over daughters, merging as good combinations are identified (can choose whether to continue with newly-enlarged parent or break out of daughter loop).

Arguably the best approach is to evaluate all possible Cluster merges and store details in an association matrix, which can be interrogated to identify chains of associations, then updated.

Cluster Association

An example approach using LArPointingCluster instances to assess pointing/directional information:

```
#include "larpandoracontent/LArHelpers/LArPointingClusterHelper.h"

#include "larpandoracontent/LArObjects/LArPointingCluster.h"
```

```
bool MyClusterMergingAlgorithm::AreClustersAssociated(const Cluster *const pParentCluster, const Cluster *const pDaughterCluster) const
{
    try
    {
        // Useful constructs for pointing information
        const LArPointingCluster parentPointingCluster(pParentCluster);
        const LArPointingCluster daughterPointingCluster(pDaughterCluster);

        LArPointingCluster::Vertex closestVertexParent, closestVertexDaughter;
        LArPointingClusterHelper::GetClosestVertices(parentPointingCluster, daughterPointingCluster, closestVertexParent, closestVertexDaughter);

        // Impact parameters
        float parentDaughterImpactL(std::numeric_limits<float>::max()), parentDaughterImpactT(std::numeric_limits<float>::max());
        LArPointingClusterHelper::GetImpactParameters(closestVertexParent, closestVertexDaughter, parentDaughterImpactL, parentDaughterImpactT);

        float daughterParentImpactL(std::numeric_limits<float>::max()), daughterParentImpactT(std::numeric_limits<float>::max());
        LArPointingClusterHelper::GetImpactParameters(closestVertexDaughter, closestVertexParent, daughterParentImpactL, daughterParentImpactT);

        // Make decision
        if (((parentDaughterImpactL < m_maxImpactL) && (parentDaughterImpactT < m_maxImpactT)) ||
            ((daughterParentImpactL < m_maxImpactL) && (daughterParentImpactT < m_maxImpactT)))
        {
            return true;
        }
    }
    catch (const StatusCodeException &statusCodeException)
    {
        std::cout << "MyClusterMergingAlgorithm::AreClustersAssociated " << statusCodeException.ToString() << std::endl;
    }

    return false;
}
```

Visualise Cluster Associations

```cpp
bool MyClusterMergingAlgorithm::AreClustersAssociated(const Cluster *const pParentCluster, const Cluster *const pDaughterCluster) const
{
    try
    {
        // Useful constructs for pointing information
        const LArPointingCluster parentPointingCluster(pParentCluster);
        const LArPointingCluster daughterPointingCluster(pDaughterCluster);

        LArPointingCluster::Vertex closestVertexParent, closestVertexDaughter;
        LArPointingClusterHelper::GetClosestVertices(parentPointingCluster, daughterPointingCluster, closestVertexParent, closestVertexDaughter);

        // Impact parameters
        float parentDaughterImpactL(std::numeric_limits<float>::max()), parentDaughterImpactT(std::numeric_limits<float>::max());
        LArPointingClusterHelper::GetImpactParameters(closestVertexParent, closestVertexDaughter, parentDaughterImpactL, parentDaughterImpactT);

        float daughterParentImpactL(std::numeric_limits<float>::max()), daughterParentImpactT(std::numeric_limits<float>::max());
        LArPointingClusterHelper::GetImpactParameters(closestVertexDaughter, closestVertexParent, daughterParentImpactL, daughterParentImpactT);

        // Visualization and debug
        std::cout << "MyClusterMergingAlgorithm::AreClustersAssociated " << std::endl;
        std::cout << "parentDaughterImpactL: " << parentDaughterImpactL << ", parentDaughterImpactT " << parentDaughterImpactT << std::endl;
        std::cout << "daughterParentImpactL: " << daughterParentImpactL << ", daughterParentImpactT " << daughterParentImpactT << std::endl;

        ClusterList parentList, daughterList;
        parentList.insert(pParentCluster); daughterList.insert(pDaughterCluster);

        PandoraMonitoringApi::VisualizeClusters(this->GetPandora(), &parentList, "ParentCluster", RED);
        PandoraMonitoringApi::VisualizeClusters(this->GetPandora(), &daughterList, "DaughterCluster", BLUE);

        PandoraMonitoringApi::AddMarkerToVisualization(this->GetPandora(), &(closestVertexParent.GetPosition()), "ParentVertex", ORANGE, 2);
        PandoraMonitoringApi::AddMarkerToVisualization(this->GetPandora(), &(closestVertexDaughter.GetPosition()), "DaughterVertex", GREEN, 2);

        PandoraMonitoringApi::ViewEvent(this->GetPandora());

        // Make decision
        if (((parentDaughterImpactL < m_maxImpactL) && (parentDaughterImpactT < m_maxImpactT)) ||
            ((daughterParentImpactL < m_maxImpactL) && (daughterParentImpactT < m_maxImpactT)))
        {
            return true;
        }
    }
    catch (const StatusCodeException &statusCodeException)
    {
        std::cout << "MyClusterMergingAlgorithm::AreClustersAssociated " << statusCodeException.ToString() << std::endl;
    }

    return false;
}
```

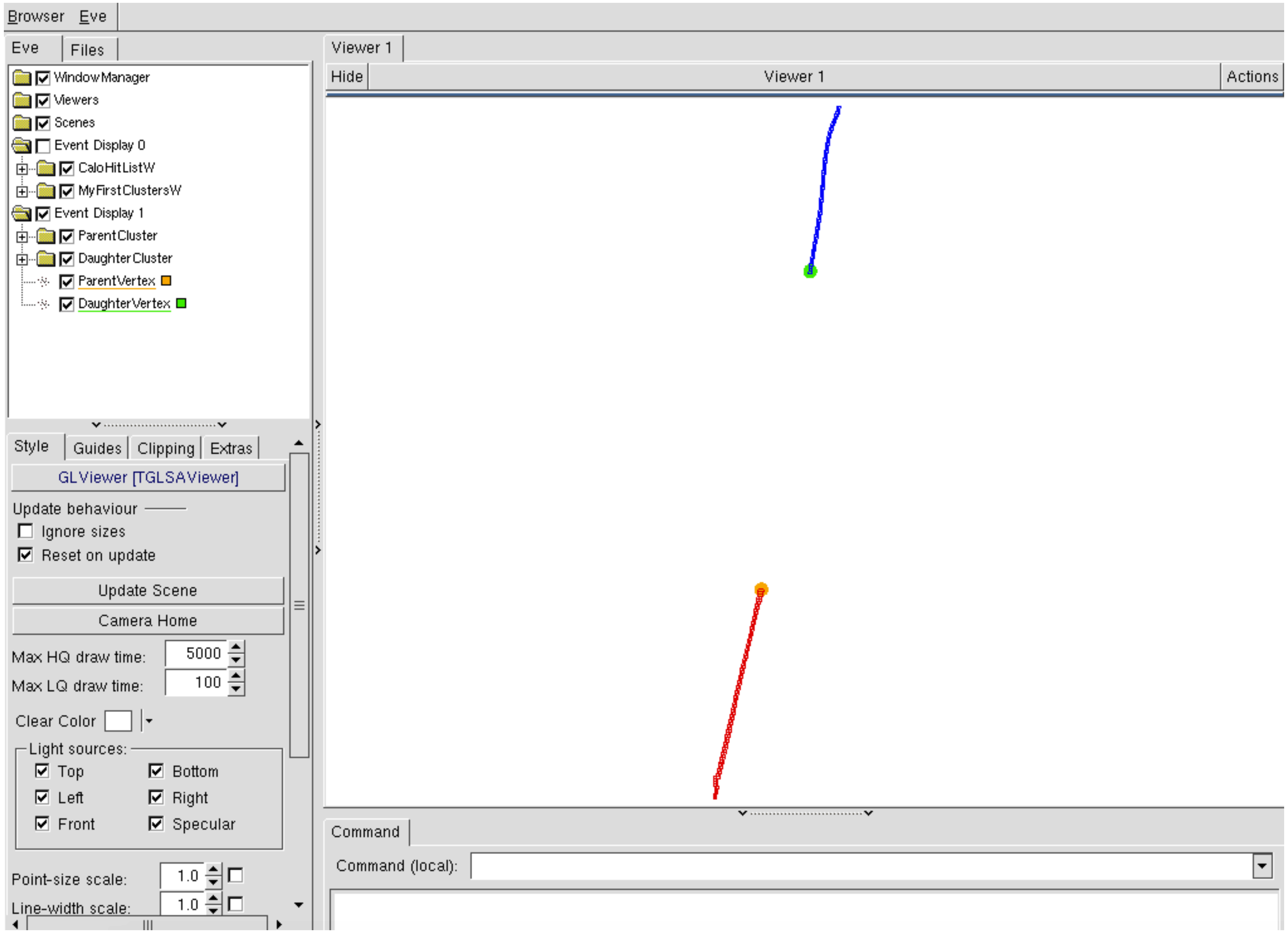
Add visualisation at the point of assessing Cluster association

Visualise Cluster Associations

```
bool MyClusterMergingAlgorithm::AreClustersAssociated(const Cluster *const pParentCluster, const Cluster *const pDaughterCluster) const
{
    try
    {
        // Useful constructs for pointing information
        const LArPointingCluster parentPointingCluster(pParentCluster);
        const LArPointingCluster daughterPointingCluster(pDaughterCluster);

        LArPointingCluster::Vertex closestVertexParent, closestVertexDaughter;
        LArPointingClusterHelper::GetClosestVertices(parentPointingCluster, daughterPointingCluster, closestVertexParent, closestVertexDaughter);

        // Impact parameters
        float parentDaughterImpactL(std::numeric_limits<float>::max()), parentDaughterImpactT(std::numeric_limits<float>::max());
        LArPointingClusterHelper::GetImpactParameters(closestVertexParent, closestVertexDaughter, parentDaughterImpactL, parentDaughterImpactT);

        float daughterParentImpactL(std::numeric_limits<float>::max()), daughterParentImpactT(std::numeric_limits<float>::max());
        LArPointingClusterHelper::GetImpactParameters(closestVertexDaughter, closestVertexParent, daughterParentImpactL, daughterParentImpactT);

        // Visualization and debug
        std::cout << "MyClusterMergingAlgorithm::AreClustersAssociated " << std::endl;
        std::cout << "parentDaughterImpactL: " << parentDaughterImpactL << ", parentDaughterImpactT " << parentDaughterImpactT << std::endl;
        std::cout << "daughterParentImpactL: " << daughterParentImpactL << ", daughterParentImpactT " << daughterParentImpactT << std::endl;

        ClusterList parentList, daughterList;
        parentList.insert(pParentCluster); daughterList.insert(pDaughterCluster);

        PandoraMonitoringApi::VisualizeClusters(this->GetPandora(), &parentList, "ParentCluster", RED);
        PandoraMonitoringApi::VisualizeClusters(this->GetPandora(), &daughterList, "DaughterCluster", BLUE);

        PandoraMonitoringApi::AddMarkerToVisualization(this->GetPandora(), &(closestVertexParent.GetPosition()), "ParentVertex", ORANGE, 2);
        PandoraMonitoringApi::AddMarkerToVisualization(this->GetPandora(), &(closestVertexDaughter.GetPosition()), "DaughterVertex", GREEN, 2);

        PandoraMonitoringApi::ViewEvent(this->GetPandora());

        // Make decision
        if (((parentDaughterImpactL < m_maxImpactL) && (parentDaughterImpactT < m_maxImpactT)) ||
            ((daughterParentImpactL < m_maxImpactL) && (daughterParentImpactT < m_maxImpactT)))
        {
            return true;
        }
    }
    catch (const StatusCodeException &statusCodeException)
    {
        std::cout << "MyClusterMergingAlgorithm::AreClustersAssociated " << statusCodeException.ToString() << std::endl;
    }

    return false;
}
```

Add visualisation at the point of assessing Cluster association

Visualise Cluster Associations



☑ WindowManager
☑ Viewers
☑ Scenes
□ Event Display 0
☑ CaloHitListW
☑ MyFirstClustersW
☑ Event Display 1
☑ ParentCluster
☑ DaughterCluster
☑ ParentVertex 🟧
☑ DaughterVertex 🟩

Update behaviour
□ Ignore sizes
☑ Reset on update

Max HQ draw time: 5000
Max LQ draw time: 100
Clear Color

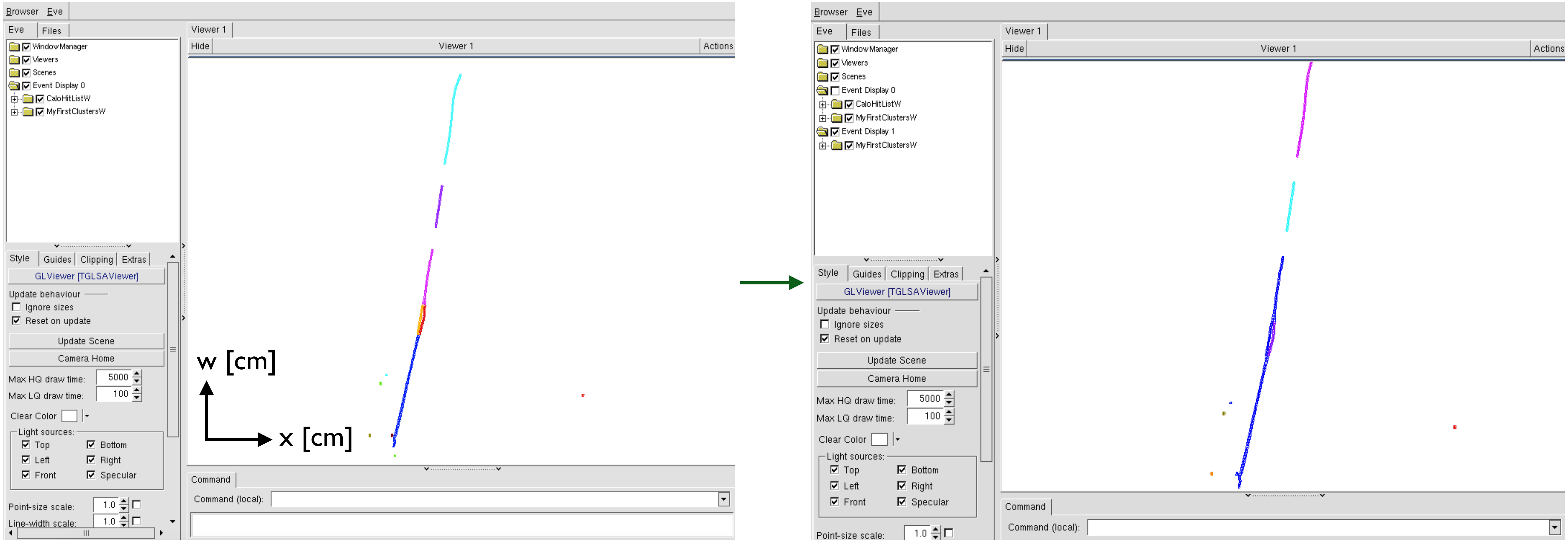
Light sources:
☑ Top ☑ Bottom
☑ Left ☑ Right
☑ Front ☑ Specular

Point-size scale: 1.0 □
Line-width scale: 1.0 □

Command (local):

```
MyClusterMergingAlgorithm::AreClustersAssociated
parentDaughterImpactL: 39.0444, parentDaughterImpactT 3.17649
daughterParentImpactL: 39.1654, daughterParentImpactT 0.791774
Press return to continue ...
```

Event Outcome using PointingClusters



Left view:

- ☑ WindowManager
- ☑ Viewers
- ☑ Scenes
- ☑ Event Display 0
  - ☑ CaloHitListW
  - ☑ MyFirstClustersW

☐ Ignore sizes
☑ Reset on update

Max HQ draw time: 5000
Max LQ draw time: 100
Clear Color

☑ Top ☑ Bottom
☑ Left ☑ Right
☑ Front ☑ Specular

Point-size scale: 1.0 ☐
Line-width scale: 1.0 ☐

Command (local):

Right view:

- ☑ WindowManager
- ☑ Viewers
- ☑ Scenes
- ☐ Event Display 0
  - ☑ CaloHitListW
  - ☑ MyFirstClustersW
- ☑ Event Display 1
  - ☑ MyFirstClustersW

☐ Ignore sizes
☑ Reset on update

Max HQ draw time: 5000
Max LQ draw time: 100
Clear Color

☑ Top ☑ Bottom
☑ Left ☑ Right
☑ Front ☑ Specular

Point-size scale: 1.0 ☐

Command (local):

Cluster Association

An example approach using LArTwoDSlidingFitResults instead (try using this to produce results):

See $MY_TEST_AREA/PandoraPFA/LArContent-v02_07_04/larpandoracontent/LArObjects/LArTwoDSlidingFitResult.h

```
/**
 *  @brief  Constructor using cluster extremal x-z positions to define primary axis
 *
 *  @param  pCluster address of the cluster
 *  @param  layerFitHalfWindow the layer fit half window
 *  @param  layerPitch the layer pitch, units cm
 */
TwoDSlidingFitResult(const pandora::Cluster *const pCluster, const unsigned int layerFitHalfWindow, const float layerPitch);
```

```
#include "larpandoracontent/LArHelpers/LArGeometryHelper.h"

#include "larpandoracontent/LArObjects/LArTwoDSlidingFitResult.h"
```

```
bool MyClusterMergingAlgorithm::AreClustersAssociated(const Cluster *const pParentCluster, const Cluster *const pDaughterCluster) const
{
    try
    {
        // Useful constructs for evaluating topological information
        const float slidingFitPitch(LArGeometryHelper::GetWireZPitch(this->GetPandora()));

        const TwoDSlidingFitResult parentFitResult(pParentCluster, m_slidingFitWindow, slidingFitPitch);
        const TwoDSlidingFitResult daughterFitResult(pDaughterCluster, m_slidingFitWindow, slidingFitPitch);

        // TODO - Make decisions

    }
    catch (const StatusCodeException &statusCodeException)
    {
        std::cout << "MyClusterMergingAlgorithm::AreClustersAssociated " << statusCodeException.ToString() << std::endl;
    }

    return false;
}
```

Performance Assessment

How well does your current set of algorithms perform?

```
<algorithm type = "LArClusteringParent">
    <algorithm type = "LArTrackClusterCreation" description = "ClusterFormation"/>
    <InputCaloHitListName>CaloHitListW</InputCaloHitListName>
    <ClusterListName>MyFirstClustersW</ClusterListName>
    <ReplaceCurrentCaloHitList>false</ReplaceCurrentCaloHitList>
    <ReplaceCurrentClusterList>true</ReplaceCurrentClusterList>
</algorithm>

<algorithm type = "MyClusterMerging">
    <InputClusterListName>MyFirstClustersW</InputClusterListName>
</algorithm>

<algorithm type = "LArTwoDParticleCreation">
    <OutputPfoListName>MyFirstParticlesW</OutputPfoListName>
    <ClusterListNameW>MyFirstClustersW</ClusterListNameW>
</algorithm>
```

Simple conversion of 2D Clusters to Particles, to provide input to EventValidation

```
<algorithm type = "LArEventValidation">
    <CaloHitListName>CaloHitListW</CaloHitListName>
    <MCParticleListName>MCParticleList3D</MCParticleListName>
    <PfoListName>MyFirstParticlesW</PfoListName>
    <NeutrinoInducedOnly>true</NeutrinoInducedOnly>
    <PrintAllToScreen>true</PrintAllToScreen>
    <PrintMatchingToScreen>true</PrintMatchingToScreen>
    <VisualizeMatching>false</VisualizeMatching>
    <MatchingMinPrimaryHits>15</MatchingMinPrimaryHits>
    <MatchingMinSharedHits>5</MatchingMinSharedHits>
    <WriteToTree>false</WriteToTree>
</algorithm>
```

Re-use pattern-recognition assessment alg from LArContent library

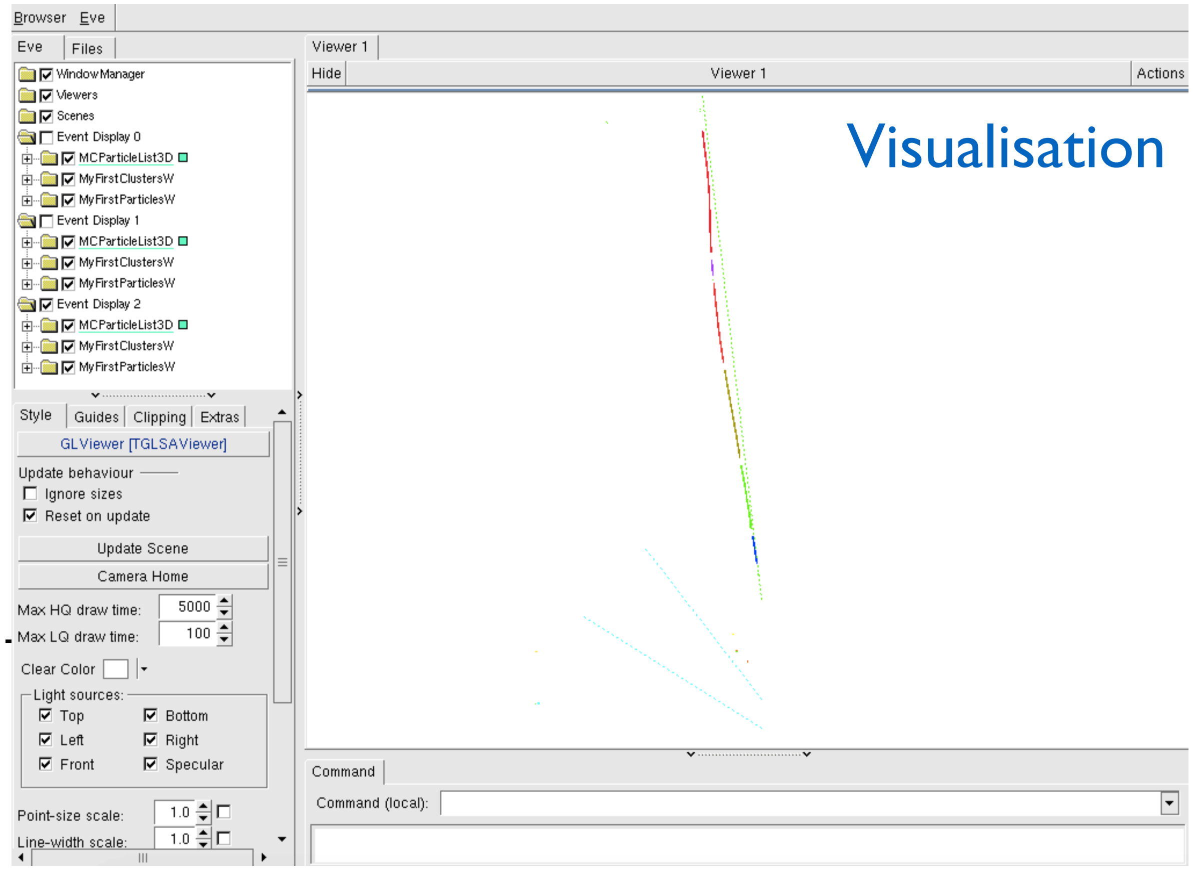
```
<algorithm type = "LArVisualMonitoring">
    <ClusterListNames>MyFirstClustersW</ClusterListNames>
    <PfoListNames>MyFirstParticlesW</PfoListNames>
    <MCParticleListNames>MCParticleList3D</MCParticleListNames>
    <SuppressMCParticles>22:0.01 2112:1.0</SuppressMCParticles>
</algorithm>
```

Summary event display

Performance Assessment

Screen output:

```
> Running Algorithm: 0x7f93afcfab20, LArListPreparation
> Running Algorithm: 0x7f93afcfad00, LArClusteringParent
----> Running Algorithm: 0x7f93afcfada0, LArTrackClusterCreation
> Running Algorithm: 0x7f93afcfaef0, MyClusterMerging
> Running Algorithm: 0x7f93afcfaf80, LArTwoDParticleCreation
> Running Algorithm: 0x7f93afcfb110, LArEventValidation
---RAW-MATCHING-OUTPUT------------------------------------------------------------
MCNeutrino, PDG 14, Nuance 1003

Primary 0, PDG 13, nMCHits 792 (0, 0, 792)
-MatchedPfo 0, PDG 22, nMatchedHits 242 (0, 0, 242), nPfoHits 242 (0, 0, 242)
-MatchedPfo 1, PDG 22, nMatchedHits 176 (0, 0, 176), nPfoHits 176 (0, 0, 176)
-MatchedPfo 2, PDG 22, nMatchedHits 160 (0, 0, 160), nPfoHits 160 (0, 0, 160)
-MatchedPfo 3, PDG 22, nMatchedHits 128 (0, 0, 128), nPfoHits 128 (0, 0, 128)
-MatchedPfo 4, PDG 22, nMatchedHits 54 (0, 0, 54), nPfoHits 54 (0, 0, 54)
-MatchedPfo 5, PDG 22, nMatchedHits 32 (0, 0, 32), nPfoHits 32 (0, 0, 32)

Primary 1, PDG 2112, nMCHits 10 (0, 0, 10)

Primary 2, PDG 2212, nMCHits 0 (0, 0, 0)

Primary 3, PDG 211, nMCHits 0 (0, 0, 0)
------------------------------------------------------------------------------------------------
---PROCESSED-MATCHING-OUTPUT--------------------------------------------------------------------

Primary 0, PDG 13, nMCHits 792 (0, 0, 792)
-MatchedPfo 0, PDG 22, nMatchedHits 242 (0, 0, 242), nPfoHits 242 (0, 0, 242)
-MatchedPfo 1, PDG 22, nMatchedHits 176 (0, 0, 176), nPfoHits 176 (0, 0, 176)
-MatchedPfo 2, PDG 22, nMatchedHits 160 (0, 0, 160), nPfoHits 160 (0, 0, 160)
-MatchedPfo 3, PDG 22, nMatchedHits 128 (0, 0, 128), nPfoHits 128 (0, 0, 128)
-MatchedPfo 4, PDG 22, nMatchedHits 54 (0, 0, 54), nPfoHits 54 (0, 0, 54)
-MatchedPfo 5, PDG 22, nMatchedHits 32 (0, 0, 32), nPfoHits 32 (0, 0, 32)

Is correct? 0
------------------------------------------------------------------------------------------------
```



Visualisation

Implement around 10+ such Cluster merging algorithms and should be converging towards 2D Clusters of target completeness (and hopefully purity too).

Try to add-in some of the 2D reconstruction algorithms from the LArContent library into your reconstruction and see if/how the reconstruction improves.

Next Exercise: Write a Cluster Matching and Particle Creation Algorithm