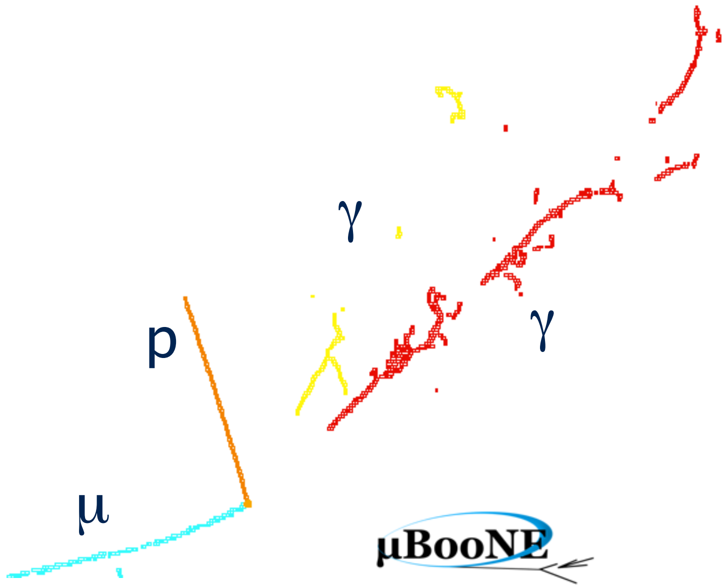


Pandora Exercise 5: Cluster Matching

J. S. Marshall for the Pandora Team

MicroBooNE Pandora Workshop

July 11-14th 2016, Cambridge

Cluster Matching Between Views

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 create Pandora Particles, containing Clusters from different views:

- Repeat 2D reconstruction for each input view
- Start to associate Clusters between views, using coordinate transformation plugins
- Visual debugging
- Particle creation

Add MyParticleCreation Algorithm

- Add a new algorithm, with a registered name such as “MyParticleCreation”.
- The input to this new algorithm will be three lists of Clusters, formed by earlier algorithms.
- So far, we have only performed Clustering in one view. Now need to apply algs to all views.

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

3 x 2D Reconstruction

- **Strategies for applying 2D reconstruction to 3 x 2D CaloHit lists are:**

  1. Repeat config in PandoraSettings XML file, with input list names, or use of current list.
  2. Write a parent algorithm which steers lists of daughter algorithms as required.

- For simplicity, we will go with strategy 1. To see an example of strategy 2, please look at larpandoracontent/LArUtility/NeutrinoParentAlgorithm.cc and .h

- The NeutrinoParent algorithm is used to accommodate slicing of input Hits into separate interactions, and re-uses multiple lists of algorithms.
- See different configurations in $MY_TEST_AREA/WorkshopContent/scripts/uboone/
  PandoraSettings_MicroBooNE_Neutrino.xml vs.
  PandoraSettings_MicroBooNE_SingleNeutrino.xml

3 x 2D Reconstruction

- For the 2D reconstruction, can either use algorithms created during this workshop, or can drop-in algorithms from the LArContent library to do the job.

```
<!-- 2D track reconstruction, U View -->
<algorithm type = "LArClusteringParent">
    <algorithm type = "LArTrackClusterCreation" description = "ClusterFormation"/>
    <InputCaloHitListName>CaloHitListU</InputCaloHitListName>
    <ClusterListName>ClustersU</ClusterListName>
    <ReplaceCurrentCaloHitList>true</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>
```

2D reconstruction for Hits in U view. Note input and output list names, then careful use of "current" list in later algs.

```
<!-- 2D track reconstruction, V View AS FOR U VIEW-->
...

<!-- 2D track reconstruction, W View AS FOR U VIEW -->
...
```

Copy and *edit* to perform V, W 2D reco

```
<algorithm type = "MyParticleCreation"/>
```

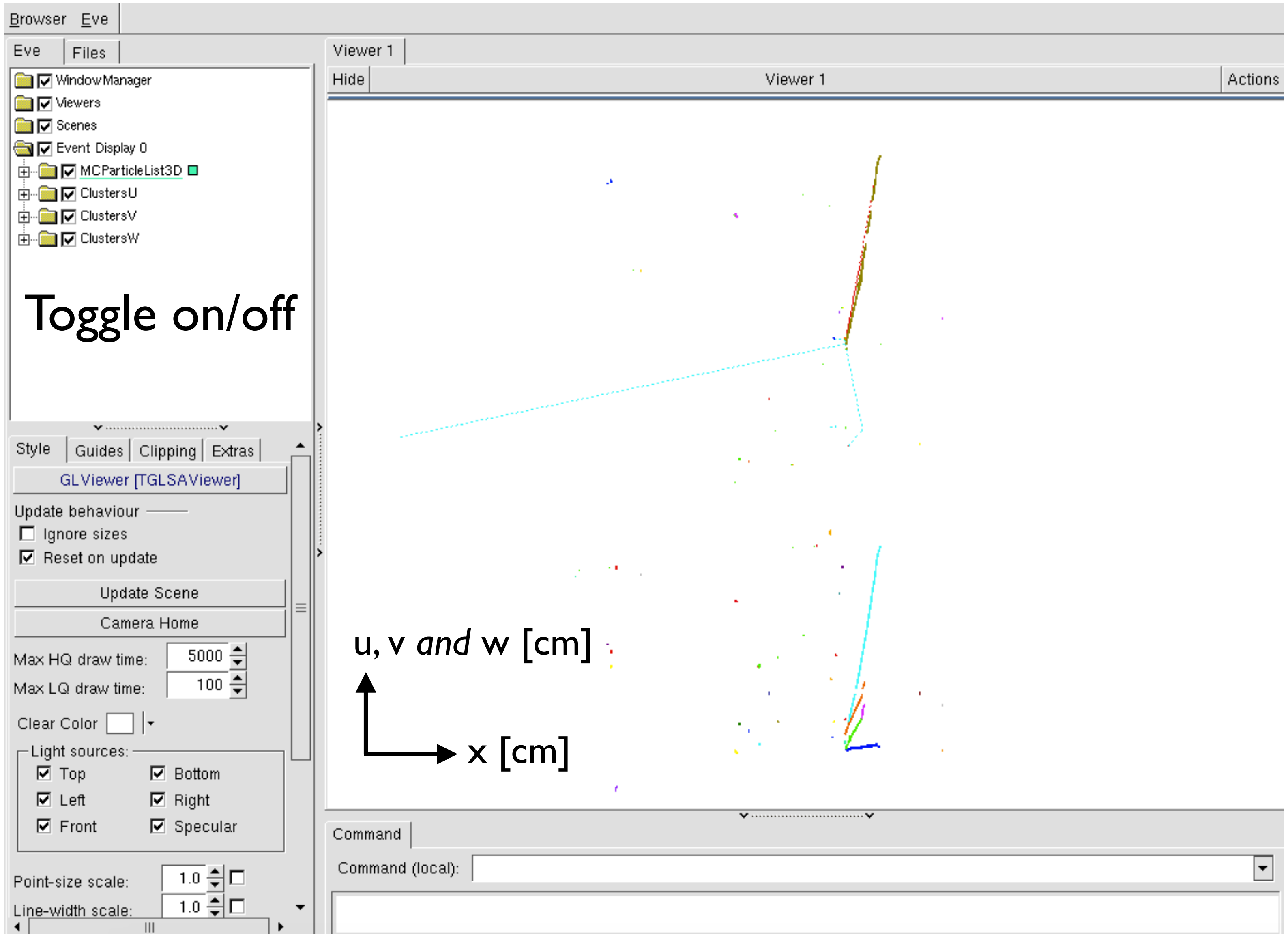
New Particle creation algorithm

```
<algorithm type = "LArVisualMonitoring">
    <ClusterListNames>ClustersU ClustersV ClustersW</ClusterListNames>
    <PfoListNames>MyParticles</PfoListNames>
    <MCParticleListNames>MCParticleList3D</MCParticleListNames>
    <SuppressMCParticles>22:0.01 2112:1.0</SuppressMCParticles>
</algorithm>
```

Visualisation at end of algorithm chain

Visualisation

Run with XML description on previous slide and should see something similar to that below:



Toggle on/off

u, v *and* w [cm]

x [cm]

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 ☐

```
> Running Algorithm: 0x7f9933644130, LArEventReading
> Running Algorithm: 0x7f993fc57b40, LArListPreparation
> Running Algorithm: 0x7f993fc57cf0, LArClusteringParent
----> Running Algorithm: 0x7f993fc57d90, LArTrackClusterCreation
> Running Algorithm: 0x7f993fc57ee0, LArLayerSplitting
> Running Algorithm: 0x7f993fc57f80, LArLongitudinalAssociation
> Running Algorithm: 0x7f993fc580d0, LArTransverseAssociation
> Running Algorithm: 0x7f993fc58180, LArLongitudinalExtension
> Running Algorithm: 0x7f993fc58250, LArTransverseExtension
> Running Algorithm: 0x7f993fc582e0, LArCrossGapsAssociation
> Running Algorithm: 0x7f993fc58500, LArCrossGapsExtension
> Running Algorithm: 0x7f993fc58050, LArOvershootSplitting
> Running Algorithm: 0x7f993fc585a0, LArBranchSplitting
> Running Algorithm: 0x7f993fc58630, LArKinkSplitting
> Running Algorithm: 0x7f993fc586d0, LArTrackConsolidation
> Running Algorithm: 0x7f993fc58810, LArClusteringParent
----> Running Algorithm: 0x7f993fc588b0, LArTrackClusterCreation
> Running Algorithm: 0x7f993fc589a0, LArLayerSplitting
> Running Algorithm: 0x7f993fc58a40, LArLongitudinalAssociation
> Running Algorithm: 0x7f993fc58380, LArTransverseAssociation
> Running Algorithm: 0x7f993fc58460, LArLongitudinalExtension
> Running Algorithm: 0x7f993fc58e70, LArTransverseExtension
> Running Algorithm: 0x7f993fc58f00, LArCrossGapsAssociation
> Running Algorithm: 0x7f993fc58fa0, LArCrossGapsExtension
> Running Algorithm: 0x7f993fc59040, LArOvershootSplitting
> Running Algorithm: 0x7f993fc590e0, LArBranchSplitting
> Running Algorithm: 0x7f993fc59170, LArKinkSplitting
> Running Algorithm: 0x7f993fc59210, LArTrackConsolidation
> Running Algorithm: 0x7f993fc59350, LArClusteringParent
----> Running Algorithm: 0x7f993fc593f0, LArTrackClusterCreation
> Running Algorithm: 0x7f993fc594e0, LArLayerSplitting
> Running Algorithm: 0x7f993fc59580, LArLongitudinalAssociation
> Running Algorithm: 0x7f993fc59640, LArTransverseAssociation
> Running Algorithm: 0x7f993fc59750, LArLongitudinalExtension
> Running Algorithm: 0x7f993fc59820, LArTransverseExtension
> Running Algorithm: 0x7f993fc598b0, LArCrossGapsAssociation
> Running Algorithm: 0x7f993fc58b00, LArCrossGapsExtension
> Running Algorithm: 0x7f993fc58ba0, LArOvershootSplitting
> Running Algorithm: 0x7f993fc58c40, LArBranchSplitting
> Running Algorithm: 0x7f993fc58cd0, LArKinkSplitting
> Running Algorithm: 0x7f993fc58d70, LArTrackConsolidation
> Running Algorithm: 0x7f993fc59a00, MyParticleCreation
> Running Algorithm: 0x7f993fc59ae0, LArVisualMonitoring
```

An Example Implementation

PandoraSettings_Workshop.xml

```xml
<algorithm type = "MyParticleCreation">
    <InputClusterListNameU>ClustersU</InputClusterListNameU>
    <InputClusterListNameV>ClustersV</InputClusterListNameV>
    <InputClusterListNameW>ClustersW</InputClusterListNameW>
    <OutputPfoListName>MyParticles</OutputPfoListName>
</algorithm>
```

MyParticleCreationAlgorithm.cc

```cpp
StatusCode MyParticleCreationAlgorithm::Run()
{
    ClusterVector sortedLongClustersU, sortedLongClustersV, sortedLongClustersW;
    this->GetSortedLongClusters(sortedLongClustersU, sortedLongClustersV, sortedLongClustersW);

    const PfoList *pTemporaryList(nullptr); std::string temporaryListName;
    PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::CreateTemporaryListAndSetCurrent(*this, pTemporaryList, temporaryListName));

    const Cluster *pBestClusterU(nullptr), *pBestClusterV(nullptr), *pBestClusterW(nullptr);
    while (this->GetBestParticle(sortedLongClustersU, sortedLongClustersV, sortedLongClustersW, pBestClusterU, pBestClusterV, pBestClusterW))
    {
        this->CreateParticle(pBestClusterU, pBestClusterV, pBestClusterW);
    }

    if (!pTemporaryList->empty())
    {
        PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::SaveList<Pfo>(*this, m_outputPfoListName));
        PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::ReplaceCurrentList<Pfo>(*this, m_outputPfoListName));
    }

    return STATUS_CODE_SUCCESS;
}
```

Now look at this in some detail…

List Management Operations

Request temporary list to receive new Particles

```
StatusCode MyParticleCreationAlgorithm::Run()
{
    ClusterVector sortedLongClustersU, sortedLongClustersV, sortedLongClustersW;
    this->GetSortedLongClusters(sortedLongClustersU, sortedLongClustersV, sortedLongClustersW);

    const PfoList *pTemporaryList(nullptr); std::string temporaryListName;
    PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::CreateTemporaryListAndSetCurrent(*this, pTemporaryList, temporaryListName));

    const Cluster *pBestClusterU(nullptr), *pBestClusterV(nullptr), *pBestClusterW(nullptr);
    while (this->GetBestParticle(sortedLongClustersU, sortedLongClustersV, sortedLongClustersW, pBestClusterU, pBestClusterV, pBestClusterW))
    {
        this->CreateParticle(pBestClusterU, pBestClusterV, pBestClusterW);
    }

    if (!pTemporaryList->empty())
    {
        PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::SaveList<Pfo>(*this, m_outputPfoListName));
        PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::ReplaceCurrentList<Pfo>(*this, m_outputPfoListName));
    }

    return STATUS_CODE_SUCCESS;
}
```

Choose to save all the Particles, which would otherwise remain in a temporary list at the end of algorithm operations and so be deleted.

GetSortedLongClusters

MyParticleCreationAlgorithm.h

```cpp
/**
 *  @brief  Use the provided list names to read input cluster lists, select clusters passing cuts and store in sorted containers
 *
 *  @param  sortedLongClustersU to receive the sorted list of long clusters in the u view
 *  @param  sortedLongClustersV to receive the sorted list of long clusters in the v view
 *  @param  sortedLongClustersW to receive the sorted list of long clusters in the w view
 */
void GetSortedLongClusters(pandora::ClusterVector &sortedLongClustersU, pandora::ClusterVector &sortedLongClustersV,
    pandora::ClusterVector &sortedLongClustersW) const;

/**
 *  @brief  Use the provided list name to read input cluster lists, select clusters passing cuts and store in sorted container
 *
 *  @param  inputClusterListName the input cluster list name
 *  @param  sortedLongClustersV to receive the sorted list of long clusters
 */
void GetSortedLongClusters(const std::string &inputClusterListName, pandora::ClusterVector &sortedLongClusters) const;
```

MyParticleCreationAlgorithm.cc

```cpp
void MyParticleCreationAlgorithm::GetSortedLongClusters(ClusterVector &sortedLongClustersU, ClusterVector &sortedLongClustersV,
    ClusterVector &sortedLongClustersW) const
{
    this->GetSortedLongClusters(m_inputClusterListNameU, sortedLongClustersU);
    this->GetSortedLongClusters(m_inputClusterListNameV, sortedLongClustersV);
    this->GetSortedLongClusters(m_inputClusterListNameW, sortedLongClustersW);
}

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

void MyParticleCreationAlgorithm::GetSortedLongClusters(const std::string &inputClusterListName, ClusterVector &sortedLongClusters) const
{
    const ClusterList *pClusterList(nullptr);
    PANDORA_THROW_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::GetList(*this, inputClusterListName, pClusterList));

    for (const Cluster *const pCluster : *pClusterList)
    {
        if (pCluster->GetNCaloHits() > m_minClusterCaloHits)
            sortedLongClusters.push_back(pCluster);
    }

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

← Avoid repeated implementation!

← Increase sophistication as req’d

GetBestParticle

```
/**
 *  @brief  Find the combination of u, v and w clusters that form the best, most plausible candidate particle
 *
 *  @param  sortedLongClustersU the sorted list of long clusters in the u view
 *  @param  sortedLongClustersV the sorted list of long clusters in the v view
 *  @param  sortedLongClustersW the sorted list of long clusters in the w view
 *  @param  pBestClusterU to receive the address of the u cluster identified as part of the best candidate particle
 *  @param  pBestClusterV to receive the address of the v cluster identified as part of the best candidate particle
 *  @param  pBestClusterW to receive the address of the w cluster identified as part of the best candidate particle
 *
 *  @return whether a candidate particle has been identified
 */
bool GetBestParticle(const pandora::ClusterVector &sortedLongClustersU, const pandora::ClusterVector &sortedLongClustersV,
    const pandora::ClusterVector &sortedLongClustersW, const pandora::Cluster *&pBestClusterU, const pandora::Cluster *&pBestClusterV,
    const pandora::Cluster *&pBestClusterW) const;
```

MyParticleCreationAlgorithm.h

```
bool MyParticleCreationAlgorithm::GetBestParticle(const ClusterVector &sortedLongClustersU, const ClusterVector &sortedLongClustersV,
    const ClusterVector &sortedLongClustersW, const Cluster *&pBestClusterU, const Cluster *&pBestClusterV, const Cluster *&pBestClusterW) const
{
    float bestOverlapFigureOfMerit(std::numeric_limits<float>::epsilon());
    pBestClusterU = nullptr; pBestClusterV = nullptr; pBestClusterW = nullptr;

    for (const Cluster *const pClusterU : sortedLongClustersU)
    {
        for (const Cluster *const pClusterV : sortedLongClustersV)
        {
            for (const Cluster *const pClusterW : sortedLongClustersW)
            {
                if (!PandoraContentApi::IsAvailable(*this, pClusterU) || !PandoraContentApi::IsAvailable(*this, pClusterV) ||
                    !PandoraContentApi::IsAvailable(*this, pClusterW))
                {
                    continue;
                }

                const float overlapFigureOfMerit(this->GetOverlapFigureOfMerit(pClusterU, pClusterV, pClusterW));

                if (overlapFigureOfMerit > bestOverlapFigureOfMerit)
                {
                    bestOverlapFigureOfMerit = overlapFigureOfMerit;
                    pBestClusterU = pClusterU; pBestClusterV = pClusterV; pBestClusterW = pClusterW;
                }
            }
        }
    }
    return (pBestClusterU && pBestClusterV && pBestClusterW);
}
```

Check whether any Cluster already used in an existing Particle

Key pattern-recognition operations all in this function

MyParticleCreationAlgorithm.cc

GetOverlapFigureOfMerit

MyParticleCreationAlgorithm.h

```
/**
 *  @brief  Get a figure of merit characterising the overlap agreement between a combination of u, v and w clusters
 *
 *  @param  pClusterU the address of the u cluster
 *  @param  pClusterV the address of the v cluster
 *  @param  pClusterW the address of the w cluster
 *
 *  @return the figure of merit
 */
float GetOverlapFigureOfMerit(const pandora::Cluster *const pClusterU, const pandora::Cluster *const pClusterV,
    const pandora::Cluster *const pClusterW) const;
```

MyParticleCreationAlgorithm.cc

```
float MyParticleCreationAlgorithm::GetOverlapFigureOfMerit(const Cluster *const pClusterU, const Cluster *const pClusterV,
    const Cluster *const pClusterW) const
{
    try
    {
        const float slidingFitPitch(LArGeometryHelper::GetWireZPitch(this->GetPandora()));
        const TwoDSlidingFitResult fitResultU(pClusterU, m_slidingFitWindow, slidingFitPitch);
        const TwoDSlidingFitResult fitResultV(pClusterV, m_slidingFitWindow, slidingFitPitch);
        const TwoDSlidingFitResult fitResultW(pClusterW, m_slidingFitWindow, slidingFitPitch);

        // ATTN Presence of more than one “fit segment” means complicated trajectory, winding back and forth in x (don’t treat here)
        if ((1 != fitResultU.GetFitSegmentList().size()) ||
            (1 != fitResultV.GetFitSegmentList().size()) ||
            (1 != fitResultW.GetFitSegmentList().size()))
        {
            return 0.f;
        }

        // TODO - Make decisions

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

    return 0.f;
}
```

Note

Focus of later slide: providing this all-important figure of merit

CreateParticle

MyParticleCreationAlgorithm.h

```
/**
 *  @brief  Create a new particle containing the provided combination of u, v and w clusters
 *
 *  @param  pClusterU the address of the u cluster for inclusion in the particle
 *  @param  pClusterV the address of the v cluster for inclusion in the particle
 *  @param  pClusterW the address of the w cluster for inclusion in the particle
 */
void CreateParticle(const pandora::Cluster *const pClusterU, const pandora::Cluster *const pClusterV,
    const pandora::Cluster *const pClusterW) const;
```

MyParticleCreationAlgorithm.cc

```
void MyParticleCreationAlgorithm::CreateParticle(const Cluster *const pClusterU, const Cluster *const pClusterV,
    const Cluster *const pClusterW) const
{
    PandoraContentApi::ParticleFlowObject::Parameters pfoParameters;

    pfoParameters.m_particleId = MU_MINUS; // ATTN Placeholder values only - assume track
    pfoParameters.m_charge = PdgTable::GetParticleCharge(pfoParameters.m_particleId.Get());
    pfoParameters.m_mass = PdgTable::GetParticleMass(pfoParameters.m_particleId.Get());
    pfoParameters.m_energy = 0.f;
    pfoParameters.m_momentum = CartesianVector(0.f, 0.f, 0.f);

    pfoParameters.m_clusterList.insert(pClusterU);
    pfoParameters.m_clusterList.insert(pClusterV);
    pfoParameters.m_clusterList.insert(pClusterW);

    const ParticleFlowObject *pPfo(nullptr);
    PANDORA_THROW_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::ParticleFlowObject::Create(*this, pfoParameters, pPfo));
}
```

← Placeholder metadata

← Specify Clusters

← Request Particle creation

GetOverlapFigureOfMerit

Start with a relatively simple approach:

1. Obtain minimum common x-coordinate
2. Extract fitted positions for u and v Clusters at this x-coordinate
3. Predict position of w Cluster at this x-coordinate, u,v→w
4. Add markers at these positions.

```
const FitSegment &fitSegmentU(fitResultU.GetFitSegmentList().front());
const FitSegment &fitSegmentV(fitResultV.GetFitSegmentList().front());
const FitSegment &fitSegmentW(fitResultW.GetFitSegmentList().front());

const float x(std::max(fitSegmentU.GetMinX(), std::max(fitSegmentV.GetMinX(), fitSegmentW.GetMinX())));

CartesianVector fitUVector(0.f, 0.f, 0.f), fitVVector(0.f, 0.f, 0.f), fitWVector(0.f, 0.f, 0.f);
CartesianVector fitUDirection(0.f, 0.f, 0.f), fitVDirection(0.f, 0.f, 0.f), fitWDirection(0.f, 0.f, 0.f);

if ((STATUS_CODE_SUCCESS != fitResultU.GetTransverseProjection(x, fitSegmentU, fitUVector, fitUDirection)) ||
    (STATUS_CODE_SUCCESS != fitResultV.GetTransverseProjection(x, fitSegmentV, fitVVector, fitVDirection)))
{
    return 0.f;
}

const float u(fitUVector.GetZ()), v(fitVVector.GetZ());
const float uv2w(LArGeometryHelper::MergeTwoPositions(this->GetPandora(), TPC_VIEW_U, TPC_VIEW_V, u, v));
const CartesianVector predictionW(x, 0.f, uv2w);

PandoraMonitoringApi::SetEveDisplayParameters(this->GetPandora(), false, DETECTOR_VIEW_XZ, -1.f, -1.f, 1.f);
PandoraMonitoringApi::AddMarkerToVisualization(this->GetPandora(), &fitUVector, "FitU", RED, 2);
PandoraMonitoringApi::AddMarkerToVisualization(this->GetPandora(), &fitVVector, "FitV", GREEN, 2);
PandoraMonitoringApi::AddMarkerToVisualization(this->GetPandora(), &predictionW, "PredictionW", BLUE, 2);

ClusterList clusterListU, clusterListV, clusterListW;
clusterListU.insert(pClusterU); clusterListV.insert(pClusterV); clusterListW.insert(pClusterW);
PandoraMonitoringApi::VisualizeClusters(this->GetPandora(), &clusterListU, "ClusterU", RED);
PandoraMonitoringApi::VisualizeClusters(this->GetPandora(), &clusterListV, "ClusterV", GREEN);
PandoraMonitoringApi::VisualizeClusters(this->GetPandora(), &clusterListW, "ClusterW", BLUE);

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

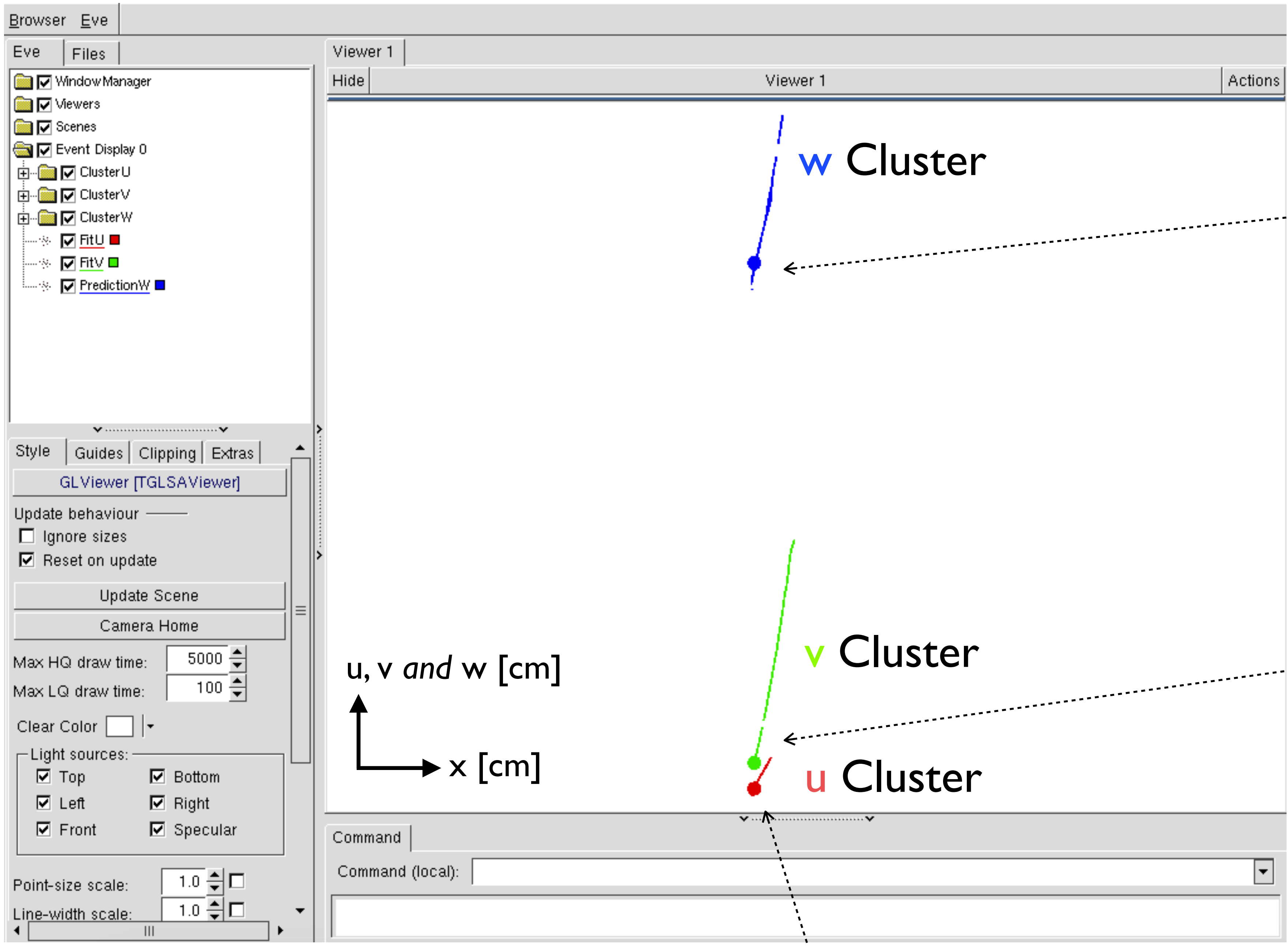
Get u, v sliding fit positions and directions at specified x

Use to predict position of w Cluster at same x

Add markers

GetOverlapFigureOfMerit

Resulting visualisation:



Update behaviour
☐ 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):

Blue marker: u,v→w prediction at minimum common x-coordinate

Green marker: fit to v Cluster at minimum common x-coordinate

Red marker: fit to u Cluster at minimum common x-coordinate

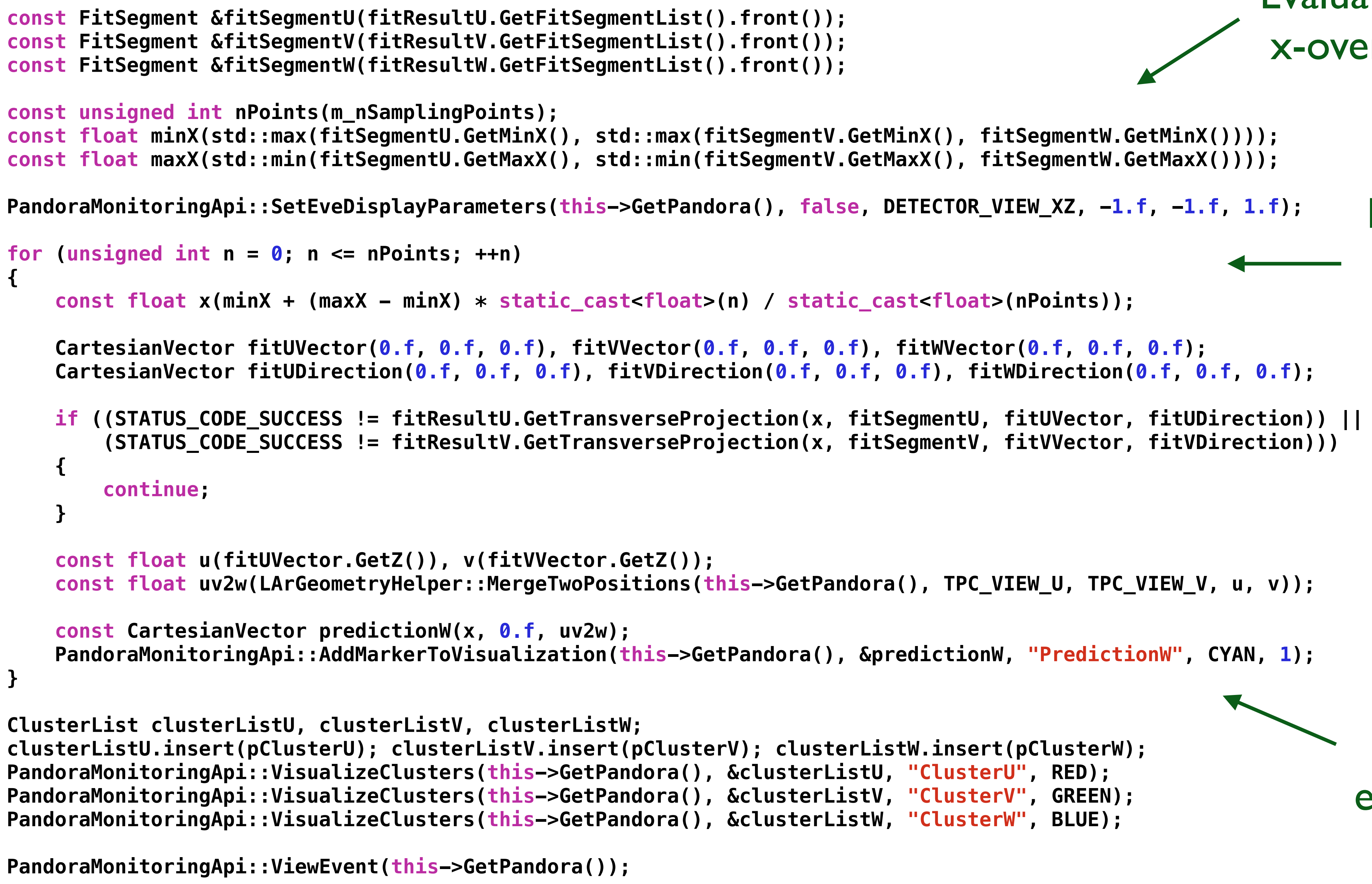
GetOverlapFigureOfMerit

Extend: sample u and v Clusters at points across the trajectory, predicting w Cluster position

Evaluate common x-overlap region

Fixed no. of sampling points to begin with (later: adaptive)

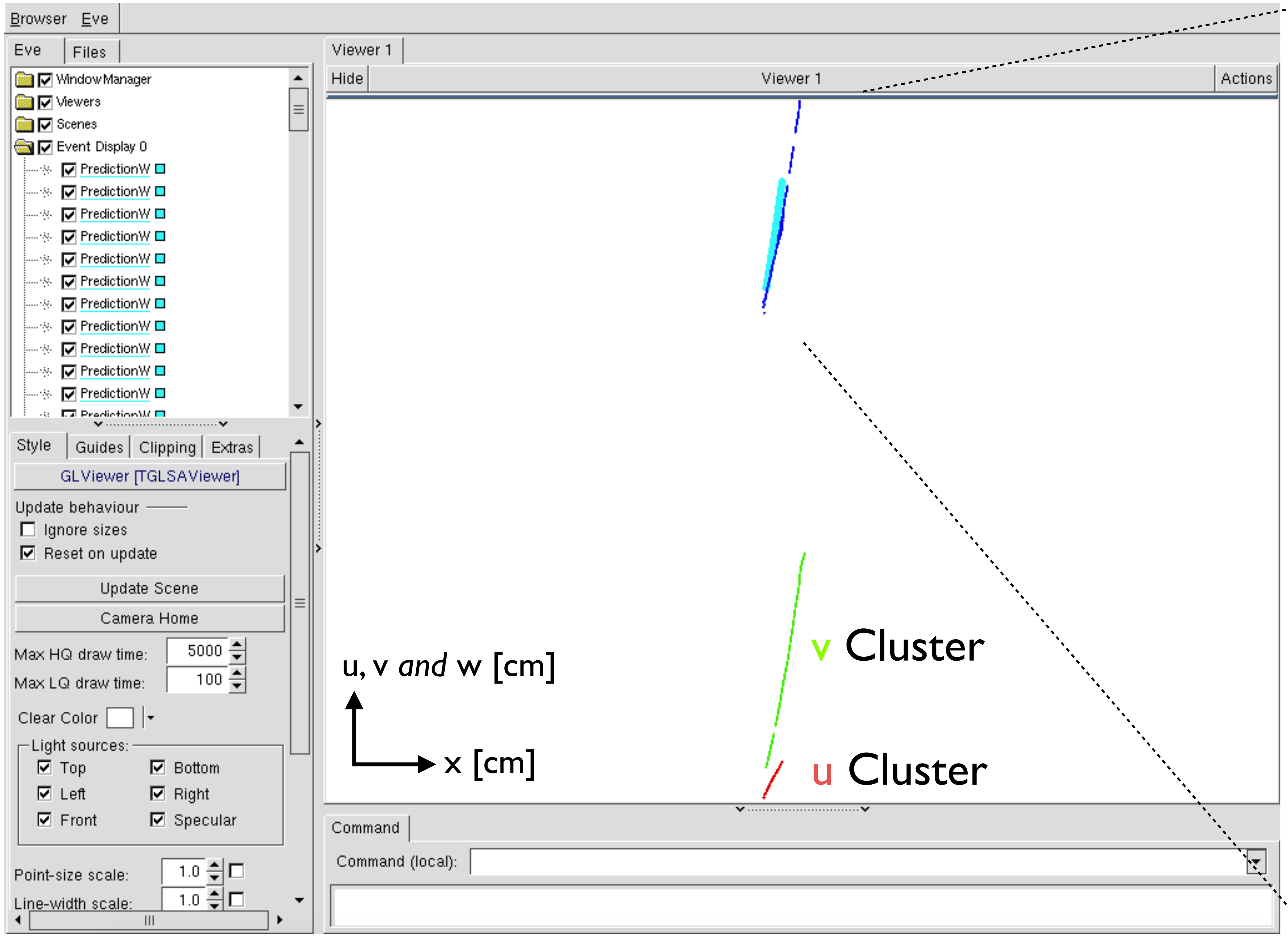
Add predicted w Cluster positions at each sampling x-value



```cpp
const FitSegment &fitSegmentU(fitResultU.GetFitSegmentList().front());
const FitSegment &fitSegmentV(fitResultV.GetFitSegmentList().front());
const FitSegment &fitSegmentW(fitResultW.GetFitSegmentList().front());

const unsigned int nPoints(m_nSamplingPoints);
const float minX(std::max(fitSegmentU.GetMinX(), std::max(fitSegmentV.GetMinX(), fitSegmentW.GetMinX())));
const float maxX(std::min(fitSegmentU.GetMaxX(), std::min(fitSegmentV.GetMaxX(), fitSegmentW.GetMaxX())));

PandoraMonitoringApi::SetEveDisplayParameters(this->GetPandora(), false, DETECTOR_VIEW_XZ, -1.f, -1.f, 1.f);

for (unsigned int n = 0; n <= nPoints; ++n)
{
    const float x(minX + (maxX - minX) * static_cast<float>(n) / static_cast<float>(nPoints));

    CartesianVector fitUVector(0.f, 0.f, 0.f), fitVVector(0.f, 0.f, 0.f), fitWVector(0.f, 0.f, 0.f);
    CartesianVector fitUDirection(0.f, 0.f, 0.f), fitVDirection(0.f, 0.f, 0.f), fitWDirection(0.f, 0.f, 0.f);

    if ((STATUS_CODE_SUCCESS != fitResultU.GetTransverseProjection(x, fitSegmentU, fitUVector, fitUDirection)) ||
        (STATUS_CODE_SUCCESS != fitResultV.GetTransverseProjection(x, fitSegmentV, fitVVector, fitVDirection)))
    {
        continue;
    }

    const float u(fitUVector.GetZ()), v(fitVVector.GetZ());
    const float uv2w(LArGeometryHelper::MergeTwoPositions(this->GetPandora(), TPC_VIEW_U, TPC_VIEW_V, u, v));

    const CartesianVector predictionW(x, 0.f, uv2w);
    PandoraMonitoringApi::AddMarkerToVisualization(this->GetPandora(), &predictionW, "PredictionW", CYAN, 1);
}

ClusterList clusterListU, clusterListV, clusterListW;
clusterListU.insert(pClusterU); clusterListV.insert(pClusterV); clusterListW.insert(pClusterW);
PandoraMonitoringApi::VisualizeClusters(this->GetPandora(), &clusterListU, "ClusterU", RED);
PandoraMonitoringApi::VisualizeClusters(this->GetPandora(), &clusterListV, "ClusterV", GREEN);
PandoraMonitoringApi::VisualizeClusters(this->GetPandora(), &clusterListW, "ClusterW", BLUE);

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

GetOverlapFigureOfMerit

Resulting visualisation:



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):

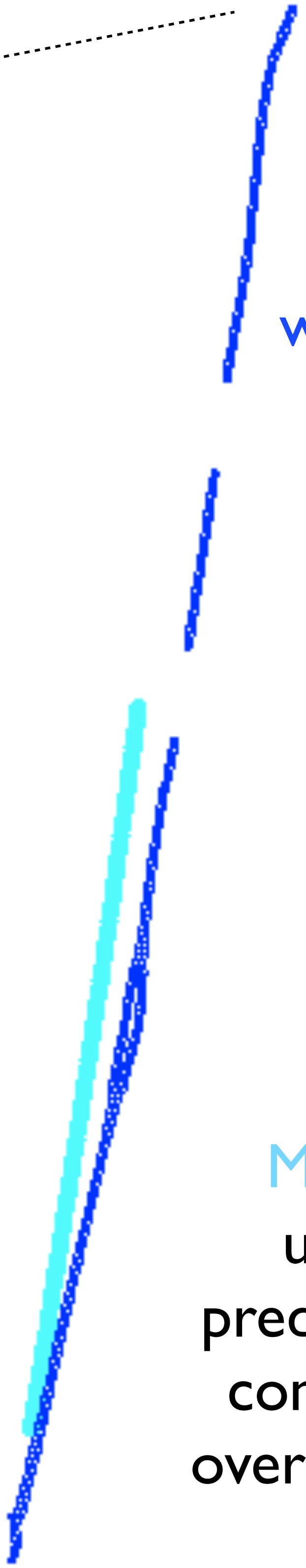
u, v *and* w [cm]

x [cm]

v Cluster

u Cluster

zoom in

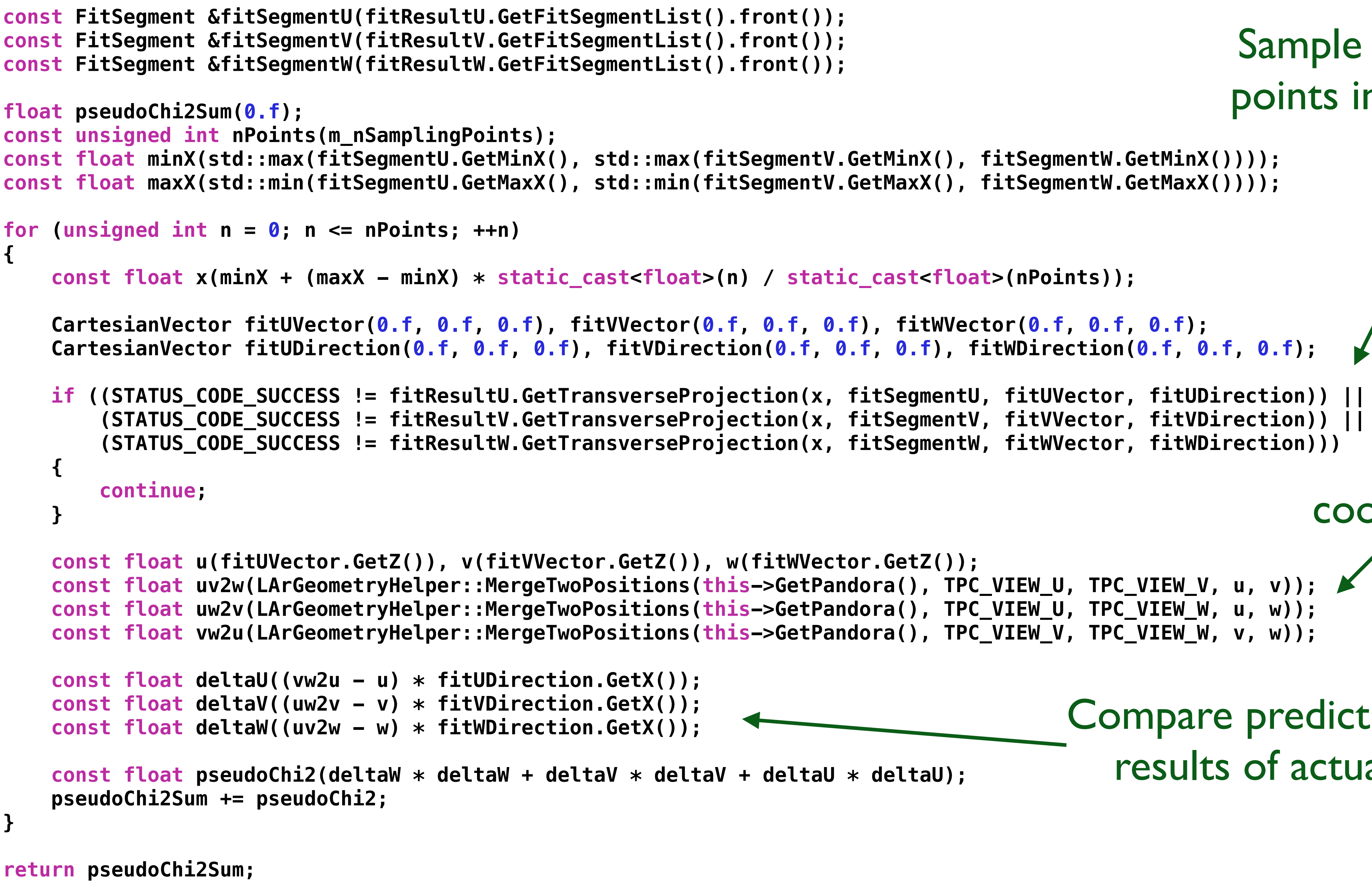


w Cluster

Markers:
u,v→w
predictions in
common x-
overlap region

GetOverlapFigureOfMerit

End up with simple version of TrackOverlapResult calculated in LArThreeDTransverseTrack alg:



```
const FitSegment &fitSegmentU(fitResultU.GetFitSegmentList().front());
const FitSegment &fitSegmentV(fitResultV.GetFitSegmentList().front());
const FitSegment &fitSegmentW(fitResultW.GetFitSegmentList().front());

float pseudoChi2Sum(0.f);
const unsigned int nPoints(m_nSamplingPoints);
const float minX(std::max(fitSegmentU.GetMinX(), std::max(fitSegmentV.GetMinX(), fitSegmentW.GetMinX())));
const float maxX(std::min(fitSegmentU.GetMaxX(), std::min(fitSegmentV.GetMaxX(), fitSegmentW.GetMaxX())));

for (unsigned int n = 0; n <= nPoints; ++n)
{
    const float x(minX + (maxX - minX) * static_cast<float>(n) / static_cast<float>(nPoints));

    CartesianVector fitUVector(0.f, 0.f, 0.f), fitVVector(0.f, 0.f, 0.f), fitWVector(0.f, 0.f, 0.f);
    CartesianVector fitUDirection(0.f, 0.f, 0.f), fitVDirection(0.f, 0.f, 0.f), fitWDirection(0.f, 0.f, 0.f);

    if ((STATUS_CODE_SUCCESS != fitResultU.GetTransverseProjection(x, fitSegmentU, fitUVector, fitUDirection)) ||
        (STATUS_CODE_SUCCESS != fitResultV.GetTransverseProjection(x, fitSegmentV, fitVVector, fitVDirection)) ||
        (STATUS_CODE_SUCCESS != fitResultW.GetTransverseProjection(x, fitSegmentW, fitWVector, fitWDirection)))
    {
        continue;
    }

    const float u(fitUVector.GetZ()), v(fitVVector.GetZ()), w(fitWVector.GetZ());
    const float uv2w(LArGeometryHelper::MergeTwoPositions(this->GetPandora(), TPC_VIEW_U, TPC_VIEW_V, u, v));
    const float uw2v(LArGeometryHelper::MergeTwoPositions(this->GetPandora(), TPC_VIEW_U, TPC_VIEW_W, u, w));
    const float vw2u(LArGeometryHelper::MergeTwoPositions(this->GetPandora(), TPC_VIEW_V, TPC_VIEW_W, v, w));

    const float deltaU((vw2u - u) * fitUDirection.GetX());
    const float deltaV((uw2v - v) * fitVDirection.GetX());
    const float deltaW((uv2w - w) * fitWDirection.GetX());

    const float pseudoChi2(deltaW * deltaW + deltaV * deltaV + deltaU * deltaU);
    pseudoChi2Sum += pseudoChi2;
}

return pseudoChi2Sum;
```

Sample u, v and w sliding fits at points in common x coordinate

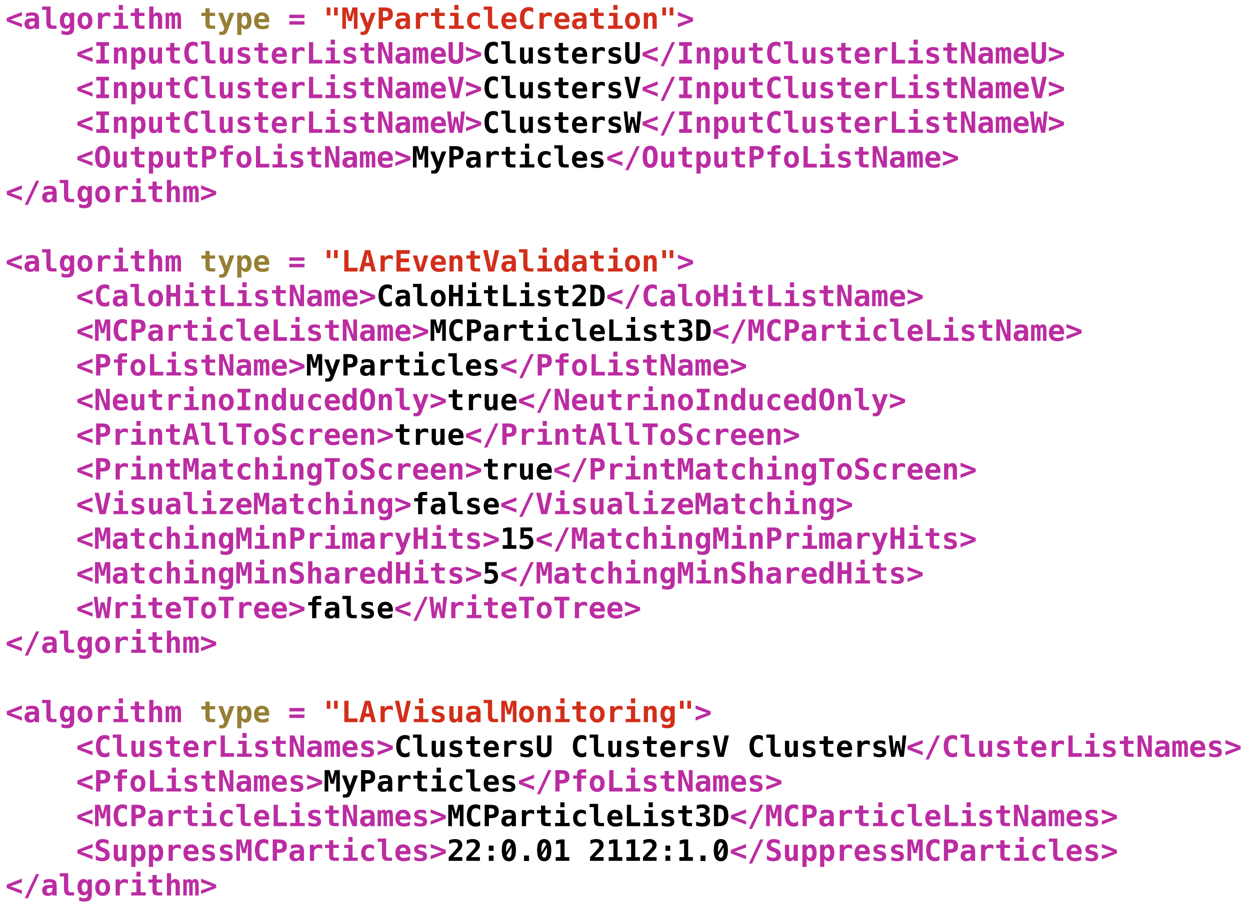
coordinate transforms

Compare predictions e.g. u,v→w, with results of actually sampling e.g. w

Try to extract more (sophisticated) information and return it to `GetBestParticle` to make more informed decision: e.g. number of "matched" sampling points, x-overlap, etc.

Performance Assessment

How well does your current set of algorithms perform?



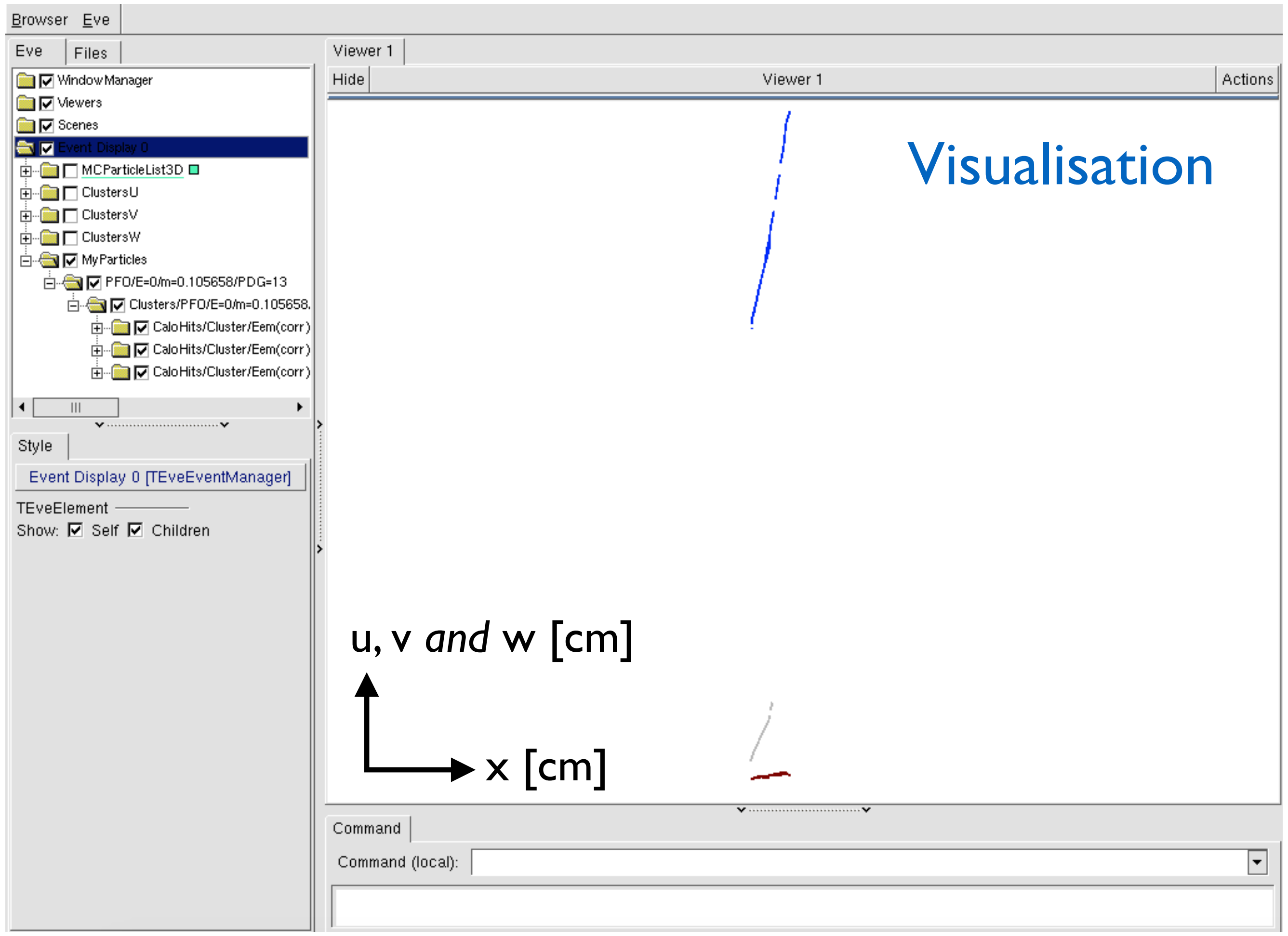
```xml
<algorithm type = "MyParticleCreation">
    <InputClusterListNameU>ClustersU</InputClusterListNameU>
    <InputClusterListNameV>ClustersV</InputClusterListNameV>
    <InputClusterListNameW>ClustersW</InputClusterListNameW>
    <OutputPfoListName>MyParticles</OutputPfoListName>
</algorithm>

<algorithm type = "LArEventValidation">
    <CaloHitListName>CaloHitList2D</CaloHitListName>
    <MCParticleListName>MCParticleList3D</MCParticleListName>
    <PfoListName>MyParticles</PfoListName>
    <NeutrinoInducedOnly>true</NeutrinoInducedOnly>
    <PrintAllToScreen>true</PrintAllToScreen>
    <PrintMatchingToScreen>true</PrintMatchingToScreen>
    <VisualizeMatching>false</VisualizeMatching>
    <MatchingMinPrimaryHits>15</MatchingMinPrimaryHits>
    <MatchingMinSharedHits>5</MatchingMinSharedHits>
    <WriteToTree>false</WriteToTree>
</algorithm>

<algorithm type = "LArVisualMonitoring">
    <ClusterListNames>ClustersU ClustersV ClustersW</ClusterListNames>
    <PfoListNames>MyParticles</PfoListNames>
    <MCParticleListNames>MCParticleList3D</MCParticleListNames>
    <SuppressMCParticles>22:0.01 2112:1.0</SuppressMCParticles>
</algorithm>
```

← Particle creation

← Re-use pattern-recognition assessment alg from LArContent library

← Summary event display

Performance Assessment



Screen output:

```
> Running Algorithm: 0x7fef3fd1e090, LArEventValidation
---RAW-MATCHING-OUTPUT-----------------------------------------------------
MCNeutrino, PDG 14, Nuance 1092

Primary 0, PDG 2212, nMCHits 422 (28, 255, 139)
-MatchedPfo 0, PDG 13, nMatchedHits 167 (28, 0, 139), nPfoHits 370 (28, 70, 272)

Primary 1, PDG 2112, nMCHits 350 (94, 102, 154)
-MatchedPfo 0, PDG 13, nMatchedHits 203 (0, 70, 133), nPfoHits 370 (28, 70, 272)

Primary 2, PDG 2112, nMCHits 89 (27, 34, 28)
------------------------------------------------------------------------------------------------
---PROCESSED-MATCHING-OUTPUT--------------------------------------------------------------------

Primary 0, PDG 2212, nMCHits 422 (28, 255, 139)

Primary 1, PDG 2112, nMCHits 350 (94, 102, 154)
-MatchedPfo 0, PDG 13, nMatchedHits 203 (0, 70, 133), nPfoHits 370 (28, 70, 272)

Primary 2, PDG 2112, nMCHits 89 (27, 34, 28)

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

Try to add-in some of the 3D track particle creation algorithms from the LArContent library into your reconstruction and see if/how the reconstruction improves.

Next Exercise: Handle Pandora Outputs in LArSoft