A New Monte-Carlo Code system for Particles Transport

M. Fouka

CRAAG, Algiers Observatory, Algiers, Algeria

First Pan-African Astro-Particle and Collider Physics Workshop

Wednesday March 22 2022

Outlines

1. General remarks on Particles Transport
2. The Global design of **PTM**
3. Stuff : Materials, Geometry, Particles, Physical processes
4. Simulation examples

General remarks on Particles Transport

Basic concepts/quantities :

- Total and differential cross sections ($\sigma(E, Z)$, $d\sigma/d\Omega$, $d\sigma/dE$)

General remarks on Particles Transport

Basic concepts/quantities :

- Total and differential cross sections ($\sigma(E, Z)$, $d\sigma/d\Omega$, $d\sigma/dE$)
- Macroscopic cross section $\Sigma = n\sigma$ (in case of a single element material)

General remarks on Particles Transport

Basic concepts/quantities :

- Total and differential cross sections ($\sigma(E, Z)$, $d\sigma/d\Omega$, $d\sigma/dE$)
- Macroscopic cross section $\Sigma = n\sigma$ (in case of a single element material)
- Interaction Mean free path $< \lambda >= \Sigma^{-1}$

General remarks on Particles Transport

Basic concepts/quantities :

- Total and differential cross sections ($\sigma(E, Z)$, $d\sigma/d\Omega$, $d\sigma/dE$)
- Macroscopic cross section $\Sigma = n\sigma$ (in case of a single element material)
- Interaction Mean free path $< \lambda >= \Sigma^{-1}$
- Decay mean free path $< \lambda >_{\text{decay}} = \gamma\beta c\tau$ (τ being the mean life time of the decaying particle)
- True interaction length $\lambda = x < \lambda >$ with x sampled by Monte-Carlo according to $\text{PDF}(x) = \exp(-x)$

General remarks on Particles Transport

Basic concepts/quantities :

- Total and differential cross sections ($\sigma(E, Z)$, $d\sigma/d\Omega$, $d\sigma/dE$)
- Macroscopic cross section $\Sigma = n\sigma$ (in case of a single element material)
- Interaction Mean free path $< \lambda >= \Sigma^{-1}$
- Decay mean free path $< \lambda >_{\text{decay}} = \gamma\beta c\tau$ (τ being the mean life time of the decaying particle)
- True interaction length $\lambda = x < \lambda >$ with x sampled by Monte-Carlo according to $\text{PDF}(x) = \exp(-x)$

General remarks on Particles Transport

Basic concepts/quantities :

- Total and differential cross sections ($\sigma(E, Z)$, $d\sigma/d\Omega$, $d\sigma/dE$)
- Macroscopic cross section $\Sigma = n\sigma$ (in case of a single element material)
- Interaction Mean free path $< \lambda >= \Sigma^{-1}$
- Decay mean free path $< \lambda >_{\text{decay}}= \gamma\beta c\tau$ (τ being the mean life time of the decaying particle)
- True interaction length $\lambda = x < \lambda >$ with x sampled by Monte-Carlo according to $\text{PDF}(x) = \exp(-x)$

The Global design of PTM

The **PTM** (stands for Particles Through Matter) is a Full Monte-Carlo Particles Transport C++ Code.

Why C++ for PTM ?

- C++ is the fastest programming language (HPC),
- C++ is an OOP language,
- C++ is widely used by HEP community (e.g., Geant4, ROOT, CLHEP, ...)

PTM content :

```
data      fwk      Makefile   particles  ptm.sh.in  utl
EXAMPLES  install  materials  physics    shapes     vis
```

The Global design of PTM

PTM is using some External C++ Libraries :

- CLHEP (Computing Libraries for High Energy Physics)
- GSL (Gnu Scientific Library)
- BOOST
- ROOT

The Global design of PTM

The **PTM** runs in two run modes :

- Processing mode managed by `RunManager` a singleton class
- Post-Processing mode managed by `AnalysisManager` a singleton class

Both classes have three main methods :

- `void Init(argc,argv)`
- `void Run()`
- `void Finish()`

The Global design of PTM

The **PTM** runs in two run modes :

- Processing mode managed by `RunManager` a singleton class
- Post-Processing mode managed by `AnalysisManager` a singleton class

Both classes have three main methods :

- `void Init(argc,argv)`
- `void Run()`
- `void Finish()`

The Global design of PTM : Processing mode

For the Processing mode one has to assign some pointers to `RunManager` instance :

- **Detector** = A collection of `PhysicalVolume` instances

The Global design of PTM : Processing mode

For the Processing mode one has to assign some pointers to `RunManager` instance :

- **Detector** = A collection of `PhysicalVolume` instances
- **PrimariesGenerator** = A collection of primary particles (one or more)

The Global design of PTM : Processing mode

For the Processing mode one has to assign some pointers to `RunManager` instance :

- **Detector** = A collection of `PhysicalVolume` instances
- **PrimariesGenerator** = A collection of primary particles (one or more)
- **PhysicsDefinition** = A collection of `ParticlePhysics` instances

The Global design of PTM : Processing mode

For the Processing mode one has to assign some pointers to `RunManager` instance :

- **Detector** = A collection of `PhysicalVolume` instances
- **PrimariesGenerator** = A collection of primary particles (one or more)
- **PhysicsDefinition** = A collection of `ParticlePhysics` instances
- **Visualizer** = `VRML` the only visualizer provided so far

The Global design of PTM : Processing mode

For the Processing mode one has to assign some pointers to `RunManager` instance :

- **Detector** = A collection of `PhysicalVolume` instances
- **PrimariesGenerator** = A collection of primary particles (one or more)
- **PhysicsDefinition** = A collection of `ParticlePhysics` instances
- **Visualizer** = `VRML` the only visualizer provided so far
- **VisHandler** = A concrete class the user can develope to altere colors and shapes of volumes, select particles/hits to visualize

The Global design of PTM : Processing mode

For the Processing mode one has to assign some pointers to `RunManager` instance :

- **Detector** = A collection of `PhysicalVolume` instances
- **PrimariesGenerator** = A collection of primary particles (one or more)
- **PhysicsDefinition** = A collection of `ParticlePhysics` instances
- **Visualizer** = `VRML` the only visualizer provided so far
- **VisHandler** = A concrete class the user can develope to altere colors and shapes of volumes, select particles/hits to visualize
- **TransportationManager** = `DefaultTransportationManager` provided, but the user can develop his/her concrete class

The Global design of PTM : Processing mode

For the Processing mode one has to assign some pointers to `RunManager` instance :

- **Detector** = A collection of `PhysicalVolume` instances
- **PrimariesGenerator** = A collection of primary particles (one or more)
- **PhysicsDefinition** = A collection of `ParticlePhysics` instances
- **Visualizer** = `VRML` the only visualizer provided so far
- **VisHandler** = A concrete class the user can develope to altere colors and shapes of volumes, select particles/hits to visualize
- **TransportationManager** = `DefaultTransportationManager` provided, but the user can develop his/her concrete class
- **Writer** (For persistency ; not mandatory) = `TextFileWriter` the only writer provided so far
- ...

The Global design of PTM : Processing mode

For the Processing mode one has to assign some pointers to `RunManager` instance :

- **Detector** = A collection of `PhysicalVolume` instances
- **PrimariesGenerator** = A collection of primary particles (one or more)
- **PhysicsDefinition** = A collection of `ParticlePhysics` instances
- **Visualizer** = `VRML` the only visualizer provided so far
- **VisHandler** = A concrete class the user can develope to altere colors and shapes of volumes, select particles/hits to visualize
- **TransportationManager** = `DefaultTransportationManager` provided, but the user can develop his/her concrete class
- **Writer** (For persistency ; not mandatory) = `TextFileWriter` the only writer provided so far
- ...

The Global design of PTM : Processing mode

```
#include "fwk/RunManager.h"
#include "vis/VRML.h"
#include "fwk/TextFileWriter.h"

#include "MyProject/TestDetector2.h"
#include "MyProject/PrimariesGenerator.h"
#include "MyProject/PhysicsDefinition.h"

using namespace fwk;
using namespace vis;
using namespace physics;

int main(int argc, char** argv) {

    RunManager& manager = RunManager::GetInstance();

    VPrimariesGenerator* primaries_generator = new PrimariesGenerator();
    Detector* det = new TestDetector2();
    VPhysicsDefinition* physicsDef = new PhysicsDefinition();
    Visualizer* vis = new VRML();
    VWriter* writer = new TextFileWriter();

    manager.SetDetector(det);
    manager.SetPrimaryParticlesGenerator(primaries_generator);
    manager.SetPhysicsDefinition(physicsDef);
    manager.SetVisualizer(vis);
    manager.SetWriter(writer);

    manager.Init(argc,argv);
    manager.Run();
    manager.Finish();

    return 0;
}
```

Needed stuffs for a complete particle transport code

- Particles : γ, optical photon, $e^{\pm}$, $\mu^{\pm}$, ν, ions, proton, neutron, hadrons,
- Detector : Physical volumes $\rightarrow$ materials $\rightarrow$ chemical elements,

Needed stuffs for a complete particle transport code

- Particles : γ, optical photon, $e^{\pm}$, $\mu^{\pm}$, ν, ions, proton, neutron, hadrons,
- Detector : Physical volumes $\rightarrow$ materials $\rightarrow$ chemical elements,
- Managers : RunManager, AnalysisManager, TransportationManager, ...

Needed stuffs for a complete particle transport code

- Particles : γ, optical photon, $e^{\pm}$, $\mu^{\pm}$, ν, ions, proton, neutron, hadrons,
- Detector : Physical volumes $\rightarrow$ materials $\rightarrow$ chemical elements,
- Managers : RunManager, AnalysisManager, TransportationManager, ...
- Visualizers : e.g., VRML,

Needed stuffs for a complete particle transport code

- Particles : γ, optical photon, $e^{\pm}$, $\mu^{\pm}$, ν, ions, proton, neutron, hadrons,
- Detector : Physical volumes $\rightarrow$ materials $\rightarrow$ chemical elements,
- Managers : RunManager, AnalysisManager, TransportationManager, ...
- Visualizers : e.g., VRML,
- Geometrical operations when propagating particle (Navigation, intersection with surfaces),

Needed stuffs for a complete particle transport code

- Particles : γ, optical photon, $e^{\pm}$, $\mu^{\pm}$, ν, ions, proton, neutron, hadrons,
- Detector : Physical volumes $\rightarrow$ materials $\rightarrow$ chemical elements,
- Managers : RunManager, AnalysisManager, TransportationManager, ...
- Visualizers : e.g., VRML,
- Geometrical operations when propagating particle (Navigation, intersection with surfaces),
- Monte-Carlo techniques : Handle competition between processes (e.g., interaction vs decay), atomic target

Needed stuffs for a complete particle transport code

- Particles : γ, optical photon, $e^{\pm}$, $\mu^{\pm}$, ν, ions, proton, neutron, hadrons,
- Detector : Physical volumes $\rightarrow$ materials $\rightarrow$ chemical elements,
- Managers : RunManager, AnalysisManager, TransportationManager, ...
- Visualizers : e.g., VRML,
- Geometrical operations when propagating particle (Navigation, intersection with surfaces),
- Monte-Carlo techniques : Handle competition between processes (e.g., interaction vs decay), atomic target
- Validate every thing

Needed stuffs for a complete particle transport code

- Particles : γ, optical photon, $e^{\pm}$, $\mu^{\pm}$, ν, ions, proton, neutron, hadrons,
- Detector : Physical volumes $\rightarrow$ materials $\rightarrow$ chemical elements,
- Managers : RunManager, AnalysisManager, TransportationManager, ...
- Visualizers : e.g., VRML,
- Geometrical operations when propagating particle (Navigation, intersection with surfaces),
- Monte-Carlo techniques : Handle competition between processes (e.g., interaction vs decay), atomic target
- Validate every thing

Chemical Elements

Chemical elements are defined in data bases. User may define his proper chemical elements to use them latter to create materials. A data base in saved in an SNL (Simple Node Language) file.

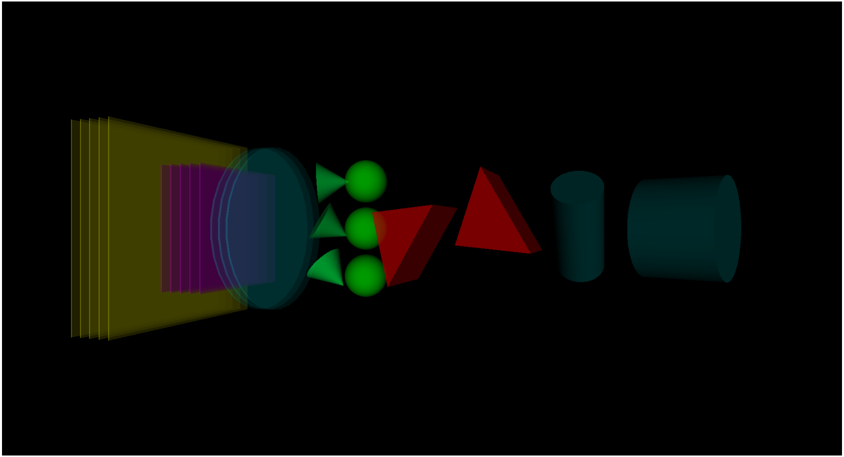
```
chemical_element:
      Z = "18"    symbol = "Ar"
      isotope:
            A = "36"    number_fraction = "0.003336"
      isotope:
            A = "38"    number_fraction = "0.000629"
      isotope:
            A = "40"    number_fraction = "0.996035"


chemical_element:
      Z = "19"    symbol = "K"
      isotope:
            A = "39"    number_fraction = "0.932581"
      isotope:
            A = "40"    number_fraction = "0.000117"
      isotope:
            A = "41"    number_fraction = "0.067302"

chemical_element:
      Z = "20"    symbol = "Ca"
      isotope:
            A = "40"    number_fraction = "0.96941"
      isotope:
            A = "42"    number_fraction = "0.00647"
      isotope:
            A = "43"    number_fraction = "0.00135"
      isotope:
            A = "44"    number_fraction = "0.02086"
      isotope:
            A = "46"    number_fraction = "4e-05"
      isotope:
            A = "48"    number_fraction = "0.00187"
```

Materials

Materials are defined in data bases. User may define his proper materials. A data base in saved in an SNL (Simple Node Language) file.

```
material:
     name = "BERYLLIUM_OXIDE"
     chemical_formula = "NONE"
     matter_state = "Solid"
     density = "3.01*g/cm3"
     mean_ionization = "93.2*eV"

     composition:
          chemical_element:
               Z = "4"  mass_fraction = "0.36032"  db_name= "NIST"

          chemical_element:
               Z = "8"  mass_fraction = "0.63968"  db_name= "NIST"

     sternheimer_parameters:
          a= "0.10755"  m= "3.4927"  x0= "0.0241"  x1= "2.5846"  C= "2.9801"  delta0= "0"

material:
     name = "BGO"
     chemical_formula = "NONE"
     matter_state = "Solid"
     density = "7.13*g/cm3"
     mean_ionization = "534.1*eV"

     composition:
          chemical_element:
               Z = "8"  mass_fraction = "0.154126"  db_name= "NIST"

          chemical_element:
               Z = "32"  mass_fraction = "0.17482"  db_name= "NIST"

          chemical_element:
               Z = "83"  mass_fraction = "0.671054"  db_name= "NIST"

     sternheimer_parameters:
          a= "0.09569"  m= "3.0781"  x0= "0.0456"  x1= "3.7816"  C= "5.7409"  delta0= "0"
```

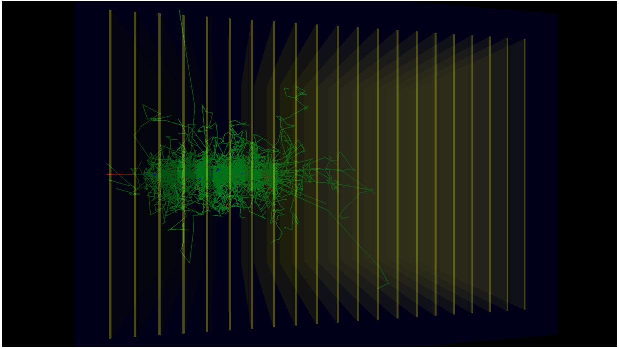
Geometry : shapes



Some defined shapes in the toolkit.

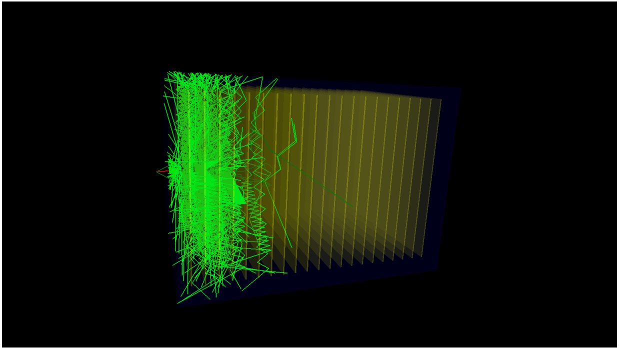
Simulation Results : EM Calorimeter

Scintillation material = Liquid Argon, 20 Lead Foils with thickness = 5mm
Injected electron with $T = 1$ GeV. Scintillation switched off

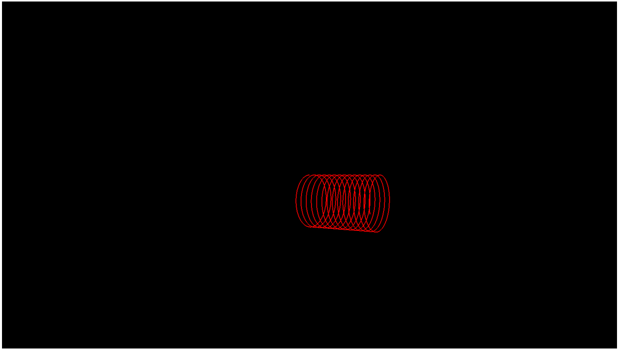


Simulation Results : EM Calorimeter

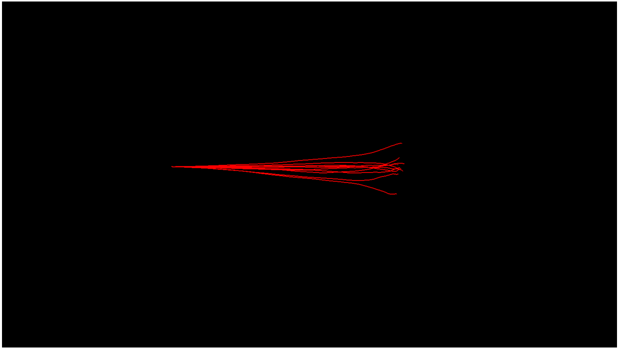
Scintillation material = Liquid Argon, 20 Lead Foils with thickness = 5mm
Injected electron with $T = 100$ MeV. Scintillation switched on



Simulation Results : Charged Particle in a Uniform Magnetic Field

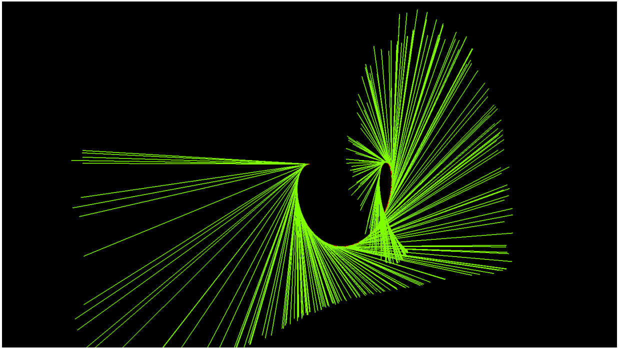


Simulation Results : Multiple Coulomb Scattering

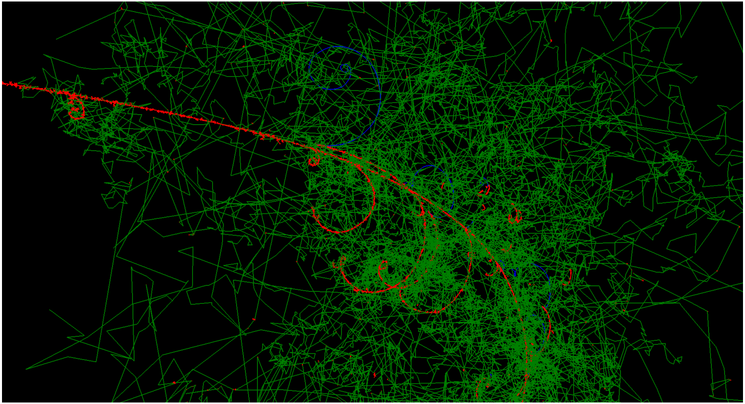


Simulation Results : Charged Particle in a Uniform Magnetic Field

Cherenkov activated ...

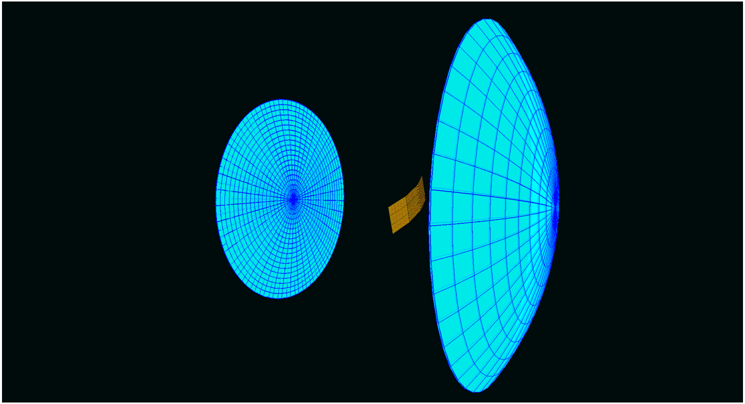


Simulation Results : Charged Particle in a Uniform Magnetic Field



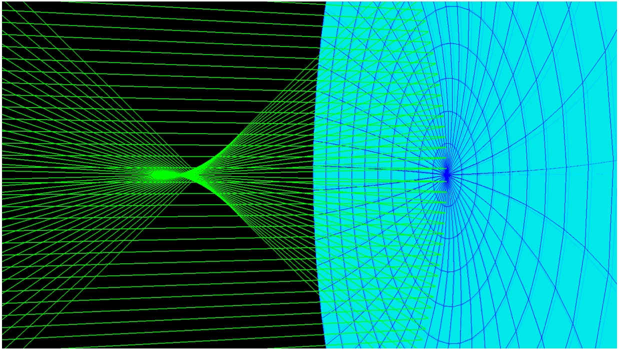
Simulation Results : JEM-SPB2

SPB2 = Corrector Plate + Primary Mirror + Camera (3 PDMs)



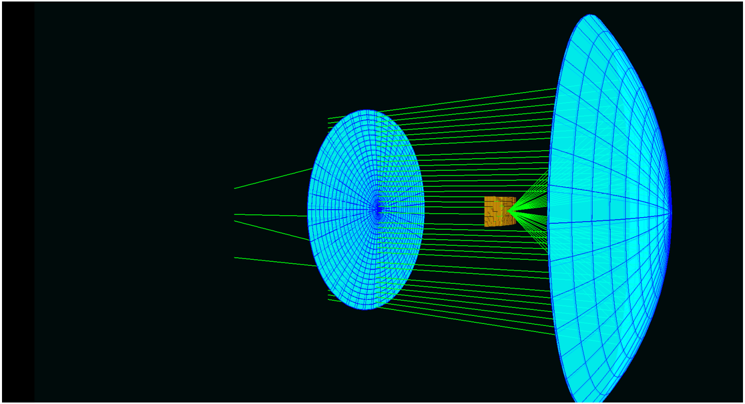
Simulation Results : JEM-SPB2

SPB2 = Corrector Plate + Primary Mirror + Camera (3 PDMs)
Primary mirror without Corrector Plate → Spherical Aberrations !



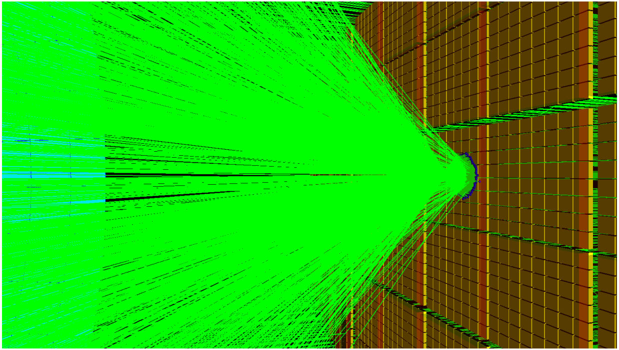
Simulation Results : JEM-SPB2

Corrector Plate + Primary mirror without Corrector Plate $\rightarrow$ Spherical Aberrations reduced significantly.

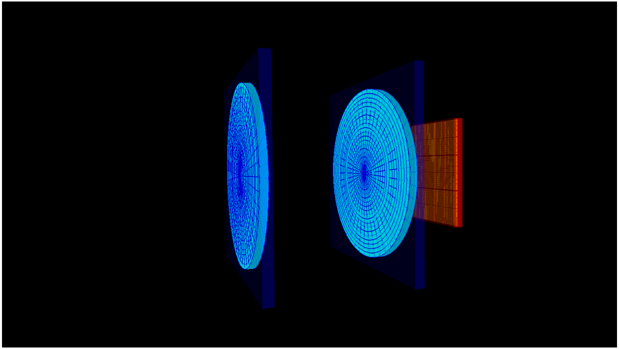


Simulation Results : JEM-SPB2

2000 injected Optical Photon at 10 degrees polar angle



Mini-EUSO = Front Fresnel Lens + Rear Fresnel Lens + Camera (1 PDM)



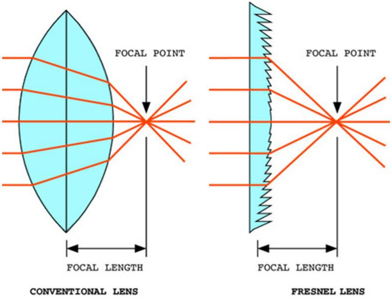
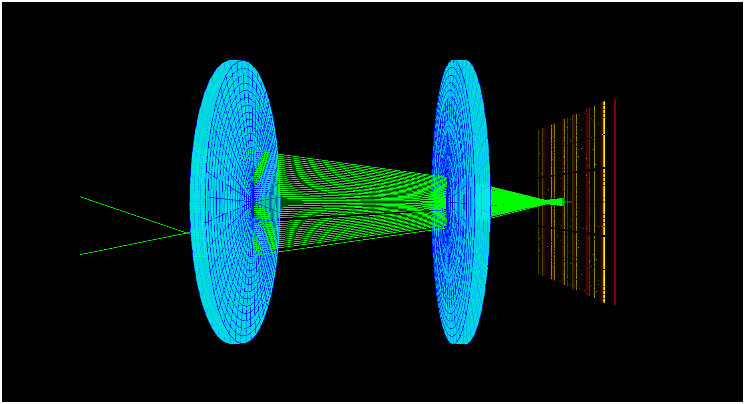
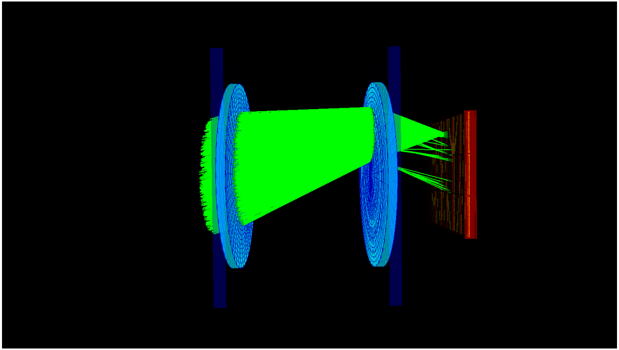


Figure – Fresnel lenses design and ray tracing

Mini-EUSO ray-traying ...



Mini-EUSO ray-traying : Reflections switched off



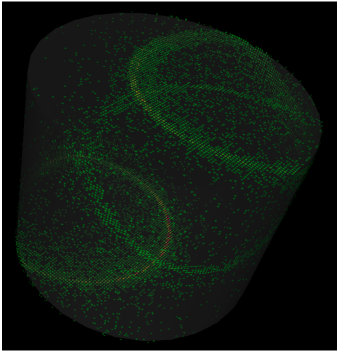


Figure – Proton decay event display in HyperK $p \to e^+\pi^0$

Atmospheric neutrinos propagation through the earth

Using atmospheric neutrinos one can discover neutrino mass hierarchy : mission of KM3NeT and IceCube/DeepCore

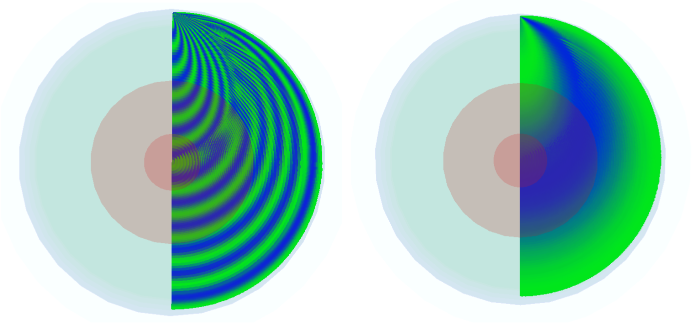


Figure – Atmospheric neutrinos propagation and oscillation through the Earth, at different zenith angles. $E_\nu = 1$ GeV for the left figure, and $E_\nu = 10$ for the right one.

KM3NeT : simulation results

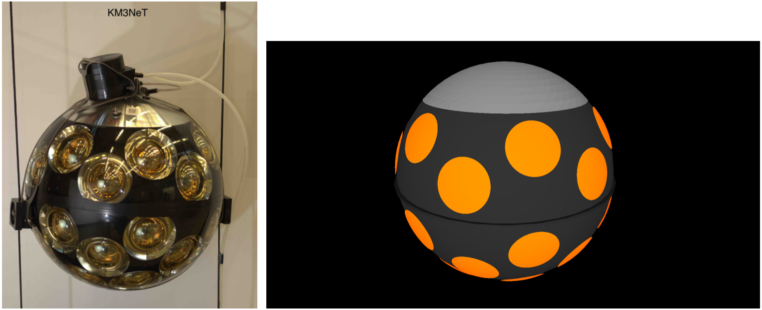
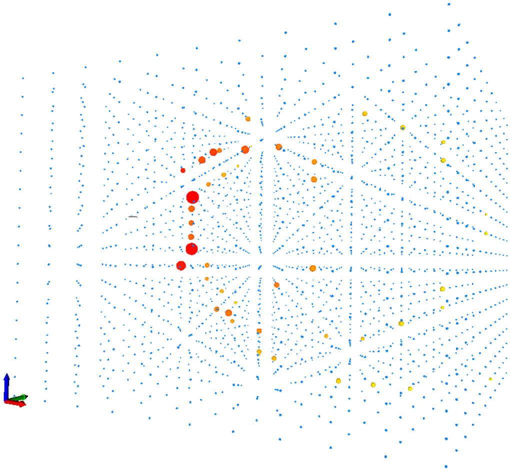


Figure – KM3NeT real DOM vs simulated DOM by the PTM code.

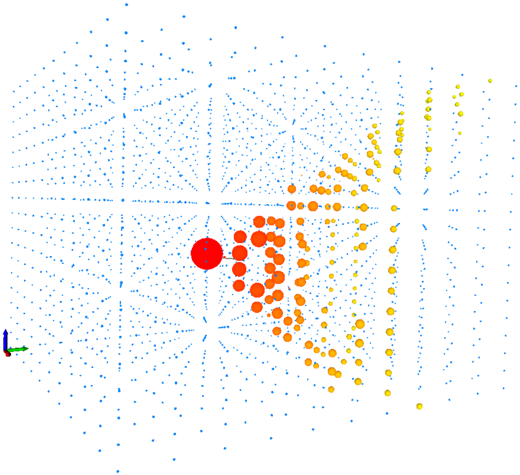
Study of ORCA sensibility to neutrino mass hierarchy

Muonic event displays ... $E_\mu = 1$ GeV.



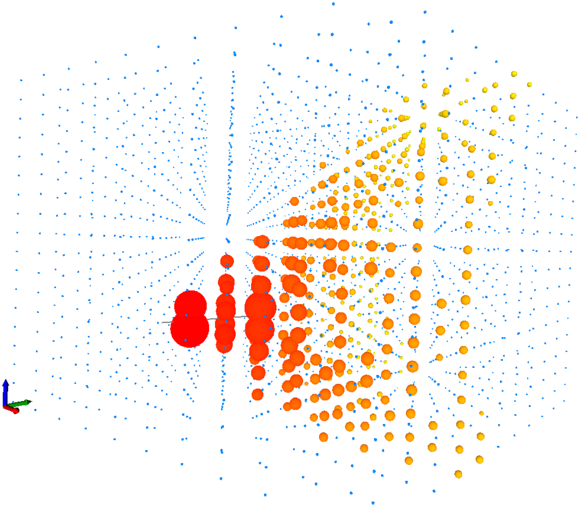
Study of ORCA sensibility to neutrino mass hierarchy

Muonic event displays ... $E_\mu = 5$ GeV.



Study of ORCA sensibility to neutrino mass hierarchy

Muonic event displays ... $E_\mu = 10$ GeV.



Thank you for you attention

BACKUPS

Particles

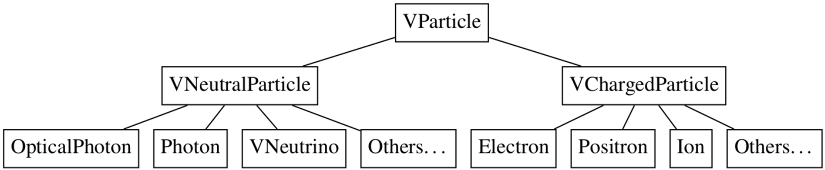


Figure – Particle inheritance tree.

The Global design of PTM : Post-Processing mode

For the Post-Processing mode one may assign four pointers to `AnalysisManager` object :

- **Detector** (not mandatory) : needed for visualization

The Global design of PTM : Post-Processing mode

For the Post-Processing mode one may assign four pointers to `AnalysisManager` object :

- **Detector** (not mandatory) : needed for visualization
- **Visualizer** (not mandatory)

The Global design of PTM : Post-Processing mode

For the Post-Processing mode one may assign four pointers to `AnalysisManager` object :

- **Detector** (not mandatory) : needed for visualization
- **Visualizer** (not mandatory)
- **Reader** (mandatory) : `TextFileReader` the only reader provided so far (corresponds to `TextFileWriter`)
- **Analyser** (not mandatory) : The user has to write its proper `UserAnalyser` by implementing `void Analyse()` method.

The Global design of PTM : Post-Processing mode

For the Post-Processing mode one may assign four pointers to `AnalysisManager` object :

- **Detector** (not mandatory) : needed for visualization
- **Visualizer** (not mandatory)
- **Reader** (mandatory) : `TextFileReader` the only reader provided so far (corresponds to `TextFileWriter`)
- **Analyser** (not mandatory) : The user has to write its proper `UserAnalyser` by implementing `void Analyse()` method.

The Global design of PTM : Post-Processing mode

```
#include "MyProject/TestDetector2.h"
#include "MyProject/Analyser.h"

#include "fwk/AnalysisManager.h"
#include "fwk/TextFileReader.h"
#include "vis/VRML.h"

using namespace fwk;
using namespace vis;

int main(int argc, char** argv) {

    AnalysisManager* manager = AnalysisManager::GetPointer();

    TextFileReader* reader = new TextFileReader();
    Detector* det = new TestDetector2();
    VRML* vis = new VRML();
    VAnalyser* analyser = new Analyser();

    manager->SetReader(reader);
    manager->SetVisualizer(vis);
    manager->SetDetector(det);
    manager->SetAnalyser(analyser);

    manager->Init(argc,argv);
    manager->Run();
    manager->Finish();

    return 0;
}
```

Chemical Elements

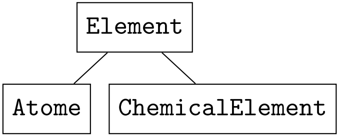


Figure – `Atome` and `ChemicalElement` inheritance from a top abstract class `Element`.

Materials

Materials are described by the `Material` class = A collection of chemical elements.

A material may need some properties :

- An Optical Model : described by the abstract class `VOpticalModel` → `TabulatedOpticalModel` is one of its concrete classes
- A Scintillation Model : described by the abstract class `VScintillationModel` → `BasicScintillationModel` is one of its concrete classes (A scintillation model may need two components : Fast and Slow)
- ...

Materials

Materials are described by the `Material` class = A collection of chemical elements.

A material may need some properties :

- An Optical Model : described by the abstract class `VOpticalModel` → `TabulatedOpticalModel` is one of its concrete classes
- A Scintillation Model : described by the abstract class `VScintillationModel` → `BasicScintillationModel` is one of its concrete classes (A scintillation model may need two components : Fast and Slow)
- ...

Optical Model properties :

- Refractive Index, $n(\epsilon)$,
- Absorption Length, ABSL(ϵ),
- Wave Length Shifting, WLS(ϵ) if the material is a shifter

Materials

Materials are described by the `Material` class = A collection of chemical elements.

A material may need some properties :

- An Optical Model : described by the abstract class `VOpticalModel` $\rightarrow$ `TabulatedOpticalModel` is one of its concrete classes
- A Scintillation Model : described by the abstract class `VScintillationModel` $\rightarrow$ `BasicScintillationModel` is one of its concrete classes (A scintillation model may need two components : Fast and Slow)
- ...

Optical Model properties :

- Refractive Index, $n(\epsilon)$,
- Absorption Length, ABSL(ϵ),
- Wave Length Shifting, WLS(ϵ) if the material is a shifter

Geometry Objects

Some **Geometry classes** :

- `utl/Vector`
- `utl/Point` ≡ `Vector`
- `utl/CoordinateSystem`
- `shapes/Volume` a collection of surfaces
- `shapes/VSurface.h` top abstract class of surfaces
- `materials/VSurfaceOpticalModel` handles reflection/absorption on optical surfaces
- `materials/VSurfaceModel` describes the effect of micro-structure on optical photon reflection
- ...

`Volume` = A Collection of surfaces. Operations are, then, performed on its surfaces.

Geometry Objects

Some **Geometry classes** :

- `utl/Vector`
- `utl/Point` ≡ `Vector`
- `utl/CoordinateSystem`
- `shapes/Volume` a collection of surfaces
- `shapes/VSurface.h` top abstract class of surfaces
- `materials/VSurfaceOpticalModel` handles reflection/absorption on optical surfaces
- `materials/VSurfaceModel` describes the effect of micro-structure on optical photon reflection
- ...

`Volume` = A Collection of surfaces. Operations are, then, performed on its surfaces.

Particles

All particles inherent from the top abstract class `VParticle`.
Main attributes :

- `TimeStamp`
- `Position`
- `Momentum`
- `TrueStepLength`
- `Trajectory` = Collection of `SpaceTimeMomentumm`
- `Children` = Sub-shower (by recursivity) of the particle.
- `Parent` (is `NULL` for primaries)
- ...

Particles

Main methods :

- `void Propagate()`
- `void Transport()`
- `void AddChild()`
- `void Interaction()`
- `void Decay()`
- `double GetRandomInteractionLength()`
- `double GetRandomDecayLength()`
- `void IsAddedToPhysicsDefinition()`
- `void GetLocalCoordinateSystem()`
- ...

void Propagate() method

```
void
VParticle::Propagate() {

   if(!IsAddedToPhysicsDefinition())
      return;

   TransportationManager* transportManager = TransportationManager::GetPointer();

   if(transportManager->GetCutStatus(this))
      return;

   Detector& det = RunManager::GetInstance().GetDetector();
   bool do_decay, do_interaction;
   double random_interaction_length, random_decay_length;
   bool& isTransportFinished = IsTransportFinished();

   while(!isTransportFinished) {
        do_interaction = false;
        do_decay = false;

        if(transportManager->GetCutStatus(this))
           break;

        random_interaction_length = GetInteractionRandomLength();
        random_decay_length = GetDecayRandomLength();

        if(random_interaction_length < random_decay_length) {
           do_interaction = true;
           SetTrueStepLength(random_interaction_length);
        } else {
           do_decay = true;
           SetTrueStepLength(random_decay_length);
        }

        Transport();
   }

   if(transportManager->GetCutStatus(this))
       return;

   if(IsTransportStoped())
       return;

   if(do_decay)
      Decay();
   else if(do_interaction)
      Interaction();
}
```

void Interaction()

```
void
VParticle::Interaction() {

    TransportationManager* transportManager = TransportationManager::GetPointer();
    if(transportManager->GetCutStatus(this))
       return;

    InteractionProcessCollection interaction_processes = GetInteractionProcesses();

    vector<double> Probabilities(interaction_processes.size());

    for(unsigned i=0; i<interaction_processes.size(); i++) {
        const double macro_sigma = GetMacroscopicCrossSection(interaction_processes[i]);
        Probabilities[i] = macro_sigma;
    }

    const int i_selected = RandomEngine::GetInstance().SelectCase(Probabilities);
    if(i_selected<0)
       return;

    VInteractionProcess* interaction_process = interaction_processes[i_selected];
    Atome* atomic_target = GetAtomicTarget(interaction_process);

    ParticleCollection secondaries = interaction_process->GenerateSecondaries(this, atomic_target);

    for(ParticleIterator it=secondaries.begin(); it<secondaries.end(); ++it)
        AddChild(*it);
}
```

void Decay()

```
void
VParticle::Decay()  {

    if(IsStable() || !GetDecayProcess())
        return;

    ParticleCollection particles = GetDecayProcess()->GetSecondaries(this);

    for(ParticleIterator it=particles.begin(); it<particles.end(); ++it)
       AddChild(*it);
}
```

Recursive mechanism : `void AddChild()` method

```cpp
void
VParticle::AddChild(VParticle* particle) {

    TransportationManager* transportManager = TransportationManager::GetPointer();

    if(!particle->IsAddedToPhysicsDefinition() || transportManager->GetCutStatus(particle) ) {
       particle->DepositEnergy();
       delete particle;
       return;
    }

    fChildren.push_back(particle);

    particle->SetParent(this);
    particle->SetLevel( GetLevel()+1 );
    particle->Propagate();
}
```

Processes/Physics Definition and Transport

Physical processes are defined by the top Abstract class `VProcess`, from which the following abstract daughters are defined :

- VInteractionProcess,
- VDecayProcess,
- VEnergyLossProcess,
- VMultipleScatteringProcess,
- VScintillationProcess,
- VCherenkovProcess,
- VTransitionRadiationProcess,
- ...

Interaction Processes

- VGammaConversion,
- VPhotoElectricEffect,
- VComptonScattering,
- VRayleighScattering,
- VPositronAnnihilation,
- VElectronBremsstrahlung,
- VMuonBremsstrahlung,
- VSingleScattering,
- ElectronIonizationStandard,
- PositronIonizationStandard,
- ...

Processes/Physics Definition and Transport

VPhysicsDefinition is the top abstract class for physics definition with the pure virtual method `virtual void Define() = 0;`

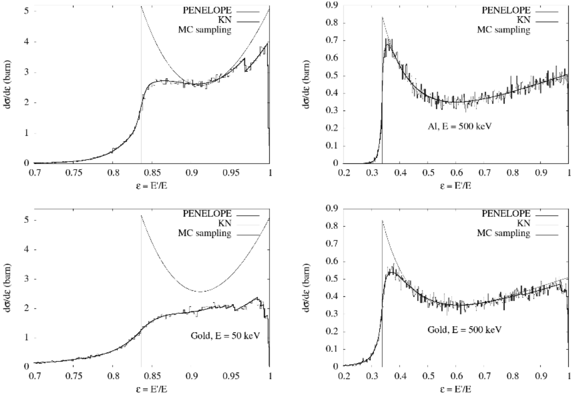
VPhysicsDefinition = A Collection of ParticlePhysics instances

```
void
PhysicsDefinition::Define()
{

     OpticalPhotonPhysics* optical_photon_phys =
         new OpticalPhotonPhysics(/*reflection_enabled*/                1,
                                  /*total_internal_reflection_enabled*/ 1,
                                  /*refraction_enabled*/                1,
                                  /*surface_absorption_enabled*/        1,
                                  /*volume_absorption_enabled*/         1,
                                  /*polarization_enabled*/              1);

     Add(optical_photon_phys);

     ParticlePhysics* photon_physics = new ParticlePhysics(new Photon);
     ParticlePhysics* electron_physics = new ParticlePhysics(new Electron);
     ParticlePhysics* positron_physics = new ParticlePhysics(new Positron);

     ParticlePhysics* proton_physics = new ParticlePhysics(new Proton);
     ParticlePhysics* muon_physics = new ParticlePhysics(new MuonPlus);

     EnergyLossStandard* energyLoss = EnergyLossStandard::GetPointer();
     GammaConversionStandard* gammaConversion = GammaConversionStandard::GetPointer();
     ComptonScatteringStandard* compton = ComptonScatteringStandard::GetPointer();
     PhotoElectricEffectPenelope* photoElectric = PhotoElectricEffectPenelope::GetPointer();
     RayleighScatteringPenelope* rayleigh = RayleighScatteringPenelope::GetPointer();

     //more stuff ...
```

Processes/Physics Definition and Transport

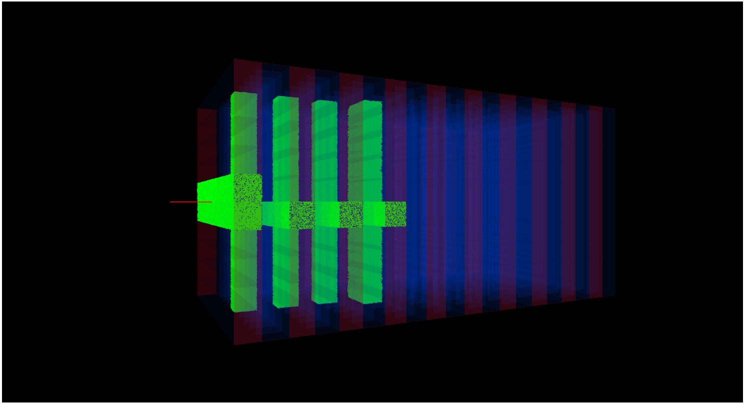
```
//...

ElectronBremsstrahlungStandard* eBremss = ElectronBremsstrahlungStandard::GetPointer();
MultipleScatteringGaussian* multScattering = MultipleScatteringGaussian::GetPointer();
ElectronCoulombSingleScattering* eSingleScattering = ElectronCoulombSingleScattering::GetPointer();

ElectronIonizationStandard* elIonization = ElectronIonizationStandard::GetPointer();
   elIonization->EnableDeltaRayProduction();
   elIonization->SetDeltaElectronCutEnergy(100*eV);
PositronIonizationStandard* posiIonization = PositronIonizationStandard::GetPointer();
   posiIonization->EnableDeltaRayProduction();
   posiIonization->SetDeltaElectronCutEnergy(100*eV);
PositronAnnihilationStandard* posiAnnihilation = PositronAnnihilationStandard::GetPointer();

MuonDecayStandard* muonDecay = new MuonDecayStandard();

BasicScintillationProcess* scintillation_process = new BasicScintillationProcess(1e-2);
BasicCherenkovProcess* cherenkov_process = new BasicCherenkovProcess(1e-3, Color(0.5,1,0));
BasicTransitionRadiationProcess* transition_radiation_process = new BasicTransitionRadiationProcess();


//Photon physics
photon_physics->AddProcess(compton);
photon_physics->AddProcess(rayleigh);
photon_physics->AddProcess(photoElectric);
photon_physics->AddProcess(gammaConversion);

//more stuff ...
```

Processes/Physics Validation tests : Photon processes



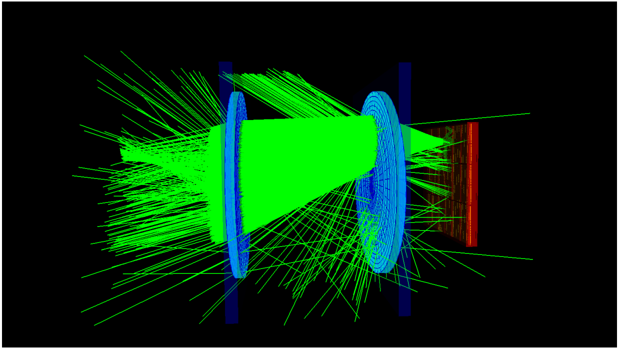
Simulation Results : Scintillation

Scintillation example : Liquid Argon



Simulation Results : MiniEuso

Mini-EUSO ray-traying : Reflections switched on (Fresnel reflection + Total internal reflection)



Mini-EUSO ray-traying : Reflections switched on (Fresnel reflection + Total internal reflection)