```
// CLHEP
#include "CLHEP/Units/PhysicalConstants.h"

// from Gaudi
#include "GaudiKernel/DeclareFactoryEntries.h"
#include "GaudiKernel/IRndmGen.h"
#include "GaudiKernel/IRndmGenSvc.h"
#include "GaudiKernel/RndmGenerators.h"
#include "GaudiKernel/Chrono.h"

// from MinervaKernel
#include "Kernel/MinervaException.h"

// from MCEvent
#include "Event/MCHit.h"

// from Det/MinervaDet
#include "MinervaDet/DeDetector.h"
#include "MinervaDet/DePlane.h"

// from Det/ODDet
#include "ODDet/DeOuterDetector.h"
#include "ODDet/DeODFrame.h"

#include "Pulse.h"
#include "OpticalModel.h"

using namespace Minerva;

// Declaration of the Tool Factory
DECLARE_TOOL_FACTORY(OpticalModel);

//=============================================================================
// OpticalModel constructor, initializes variables, declare properties
//=============================================================================

OpticalModel::OpticalModel(const std::string& type, const std::string& name, const
IInterface* parent )
  : GaudiTool(type, name, parent){

  // declare interface
  declareInterface<IOpticalModel>(this);

  // properties
  declareProperty("IDDetectorLocation", m_idDetectorLocation =
"/dd/Structure/Minerva/Detector/InnerDetector" );
  declareProperty("ODDetectorLocation", m_odDetectorLocation =
"/dd/Structure/Minerva/Detector/OuterDetector" );
  declareProperty("MTDetectorLocation", m_mtDetectorLocation =
"/dd/Structure/Minerva/Detector/MTest" );

  declareProperty("FudgeFactor",        m_fudgeFactor        = 1.4e-3 );
  declareProperty("NPulsesPerHit",      m_nPulsesPerHit      = 2 );
  declareProperty("OverallLightOutput", m_overallLightOutput = 1.e4/CLHEP::MeV );

  declareProperty("BirksConstant",        m_birksConstant         =
0.133*CLHEP::m/CLHEP::GeV );
  declareProperty("ScintFluorDecayTime",  m_scintFluorDecayTime   = 2.0*CLHEP::ns );
  declareProperty("FiberFluorDecayTime",  m_fiberFluorDecayTime   = 7.5*CLHEP::ns );
  declareProperty("ReflectionCoefficient",m_ReflectionCoefficient = 0.8 );

  declareProperty("GreenFiberRefractionIndex",       m_greenFiberRefractionIndex
```

```
= 1.6 );
  declareProperty("GreenFiberAttenuationAmplitude1", m_greenFiberAttenuationAmplitude1
= 0.67 );
  declareProperty("GreenFiberAttenuationLength1",    m_greenFiberAttenuationLength1
= 3.*CLHEP::m );
  declareProperty("GreenFiberAttenuationAmplitude2", m_greenFiberAttenuationAmplitude2
= 0.0 );
  declareProperty("GreenFiberAttenuationLength2",    m_greenFiberAttenuationLength2
= 3.*CLHEP::m );

  declareProperty("ClearFiberLength",                m_clearFiberLength
= 1.*CLHEP::m );
  declareProperty("ClearFiberRefractionIndex",       m_clearFiberRefractionIndex
= 1.6 );
  declareProperty("ClearFiberAttenuationAmplitude1", m_clearFiberAttenuationAmplitude1
= 1.0 );
  declareProperty("ClearFiberAttenuationLength1",    m_clearFiberAttenuationLength1
= 8.0*CLHEP::m );
  declareProperty("ClearFiberAttenuationAmplitude2", m_clearFiberAttenuationAmplitude2
= 0.0 );
  declareProperty("ClearFiberAttenuationLength2",    m_clearFiberAttenuationLength2
= 1.35*CLHEP::m );

}

//=============================================================================
// Destructor
//=============================================================================

OpticalModel::~OpticalModel() {
}

//=============================================================================
// initialize()
//=============================================================================

StatusCode OpticalModel::initialize() {

  StatusCode sc = GaudiTool::initialize();
  if ( sc.isFailure() ) return sc;

  // Try to get a handle to the MTest detector
  try{
    m_deMTestDetector = getDet<DeDetector>(m_mtDetectorLocation);
    if( m_deMTestDetector ){
      info() << "Found DeMTestDetector " << endmsg;
      m_deInnerDetector = m_deMTestDetector;
    }
  }
  catch(GaudiException& e){

    info() << "Failed to retrieve DeMTestDetector, try DeInnerDetector" << endmsg;

    // don't return a status code yet.
    // No MTest detector probably means this is the regular simulation.
    // Get a handle to the regular minerva inner and outer detectors.
    try{
      m_deInnerDetector = getDet<DeDetector>(m_idDetectorLocation);
    }
    catch(GaudiException& e){
      fatal() << "Failed to retrieve DeInnerDetector" << endmsg;
      return StatusCode::FAILURE;
```

```
    }

    // Get a handle to the (outer) detector
    try{
      m_deOuterDetector = getDet<DeOuterDetector>(m_odDetectorLocation);
    }
    catch(GaudiException& e){
      fatal() << "Failed to retrieve DeOuterDetector" << endmsg;
      return StatusCode::FAILURE;
    }
  }


  // initialize random service
  sc = service( "RndmGenSvc", m_randSvc, true );
  if( sc.isFailure() ) {
    throw MinervaException("Can't initialize RndmGenSvc");
  }
  return StatusCode::SUCCESS;

}


//==========================================================================
// processMCHit() - process one hit at a time
//==========================================================================

void OpticalModel::processMCHit(const MCHitPtr hit, Pulses& pulses) const {

  debug() << "OpticalModel::processMCHit" << endmsg;
  debug() << "Hit: " << *hit << endmsg;


  // These two lines find out if hit is in a strip or a bar
  // Returns zero for the irrelevant one.
  StripID stripid = findStripID( hit );
  BarID   barid = findBarID( hit );

  double pePerPulse = birksPe(hit->energy(), hit->pathLength())/(2.*m_nPulsesPerHit);


  if( stripid ){
    for( int i = 0; i < m_nPulsesPerHit; ++i ){
      PulsePtr directPulse = createDirectIDPulse(stripid, hit, pePerPulse);
      pulses.push_back(directPulse);
      PulsePtr reflectedPulse = createReflectedIDPulse(stripid, hit, pePerPulse);
      pulses.push_back(reflectedPulse);
    }
  }
  else if( barid ){
    for( int i = 0; i < m_nPulsesPerHit; ++i ){
      PulsePtr directPulse = createDirectODPulse(barid, hit, pePerPulse);
      pulses.push_back(directPulse);
      PulsePtr reflectedPulse = createReflectedODPulse(barid, hit, pePerPulse);
      pulses.push_back(reflectedPulse);
    }
  }
  else
    throw MinervaException("MCHit is not inside the detector");

}
```

```
//==========================================================================
// findStripID() - find StripID that the MCHit belongs to
//==========================================================================

StripID OpticalModel::findStripID(const MCHitPtr hit) const {

  debug() << "OpticalModel::findStripID(const MCHitPtr)" << endmsg;

  // determine hit center halfway between entry and exit
  Gaudi::XYZPoint hitCenter = hit->entry() + hit->displacement()/2.;
  debug() << "Hit at: " << hitCenter << endmsg;

  // skip hits that are not inside ID
  if ( ! m_deInnerDetector->isInside(hitCenter) ) {
    return 0;
  }

  StripID stripid = m_deInnerDetector->getStripID(hitCenter);

  return stripid;

}

//==========================================================================
// findBarID() - find BarID that the MCHit belongs to
//==========================================================================

BarID OpticalModel::findBarID(const MCHitPtr hit) const {

  debug() << "OpticalModel::findFrameID(const MCHitPtr)" << endmsg;

  // determine hit center halfway between entry and exit
  Gaudi::XYZPoint hitCenter = hit->entry() + hit->displacement()/2.;
  debug() << "Hit at: " << hitCenter << endmsg;

  // skip hits that are not inside OD
  // Protect from MTest here which has no OD.
  if ( m_deOuterDetector){
    if ( ! m_deOuterDetector->isInside(hitCenter) ) {
      return 0;
    }
    BarID barid = m_deOuterDetector->getBarID(hitCenter);

    return barid;
  }

  // a non-zero return is in the if statement above.
  return 0;

}


//==========================================================================
// createIDDirectPulse() - create a direct pulse in the ID of a given #pe,
//   calculate propagation time, attenuation
//==========================================================================

PulsePtr OpticalModel::createDirectIDPulse(StripID id, const MCHitPtr hit, double pe)
const {

  // Distance to connector
```

```
  Gaudi::XYZPoint hitCenter = hit->entry() + hit->displacement()/2.;
  const DePlane* plane = m_deInnerDetector->getDePlane(id);

  // time of the hit
  double time = hit->time();
  debug() << "direct pulse: hit time = " << time << endmsg;
  // add scint fluor decay time
  time += throwScintFluorDecayTime();
  debug() << "direct pulse: hit+scint time = " << time << endmsg;
  // add fiber fiber decay time
  time += throwFiberFluorDecayTime();
  debug() << "direct pulse: hit+scint+fluor time = " << time << endmsg;
  //     : need to add calculation of the non straight line distance
  double distanceGreen = plane->distanceToConnector(hitCenter, id);
  debug() << "direct pulse: distance green = " << distanceGreen << endmsg;
  // add green fiber travel time
  time += travelTimeGreenFiber(distanceGreen);
  debug() << "direct pulse: hit+scint+fluor+green time = " << time << endmsg;
  double atten = 1.;
  // attenuation in green
  atten *= attenuationGreenFiber(distanceGreen);
  debug() << "direct pulse: green attenuation = " << atten << endmsg;
  // add clear fiber travel time
  time += travelTimeClearFiber(m_clearFiberLength);
  debug() << "direct pulse: hit+scint+fluor+green+clear time = " << time << endmsg;
  // attenuation in clear
  atten *= attenuationClearFiber(m_clearFiberLength);
  debug() << "direct pulse: green*clear attenuation = " << atten << endmsg;
  // pe
  pe *= atten*m_fudgeFactor;
  debug() << "direct pulse: pe at pmt = " << pe << endmsg;
  // return Pulse
  return PulsePtr(new Pulse(id.id(), hit, time, pe));

}


//==========================================================================
// createIDDirectPulse() - create a direct pulse in the OD of a given #pe,
//   calculate propagation time, attenuation
//==========================================================================

PulsePtr OpticalModel::createDirectODPulse(BarID id, const MCHitPtr hit, double pe)
const {

  // Distance to connector
  Gaudi::XYZPoint hitCenter = hit->entry() + hit->displacement()/2.;
  const DeODFrame* frame = m_deOuterDetector->getDeODFrame(hitCenter);

  // time of the hit
  double time = hit->time();
  debug() << "direct pulse: hit time = " << time << endmsg;
  // add scint fluor decay time
  time += throwScintFluorDecayTime();
  debug() << "direct pulse: hit+scint time = " << time << endmsg;
  // add fiber fiber decay time
  time += throwFiberFluorDecayTime();
  debug() << "direct pulse: hit+scint+fluor time = " << time << endmsg;
  //     : need to add calculation of the non straight line distance
  double distanceGreen = frame->distanceToConnector(hitCenter, id);
  debug() << "direct pulse: distance green = " << distanceGreen << endmsg;
  // add green fiber travel time
```

```
  time += travelTimeGreenFiber(distanceGreen);
  debug() << "direct pulse: hit+scint+fluor+green time = " << time << endmsg;
  double atten = 1.;
  // attenuation in green
  atten *= attenuationGreenFiber(distanceGreen);
  debug() << "direct pulse: green attenuation = " << atten << endmsg;
  // add clear fiber travel time
  time += travelTimeClearFiber(m_clearFiberLength);
  debug() << "direct pulse: hit+scint+fluor+green+clear time = " << time << endmsg;
  // attenuation in clear
  atten *= attenuationClearFiber(m_clearFiberLength);
  debug() << "direct pulse: green*clear attenuation = " << atten << endmsg;
  // pe
  pe *= atten*m_fudgeFactor;
  debug() << "direct pulse: pe at pmt = " << pe << endmsg;
  // return Pulse
  return PulsePtr(new Pulse(id.id(), hit, time, pe));

}


//==========================================================================
// createReflectedIDPulse() - create a reflected pulse in the ID  of a
//   given #pe, calculate propagation time, attenuation
//==========================================================================

PulsePtr OpticalModel::createReflectedIDPulse(StripID id, const MCHitPtr hit, double
pe) const {

  // Distance to mirror + full strip length
  Gaudi::XYZPoint hitCenter = hit->entry() + hit->displacement()/2.;
  const DePlane* plane = m_deInnerDetector->getDePlane(id);

  // time of the hit
  double time = hit->time();
  debug() << "reflected pulse: hit time = " << time << endmsg;
  // add scint fluor decay time
  time += throwScintFluorDecayTime();
  debug() << "reflected pulse: hit+scint time = " << time << endmsg;
  // add fiber fiber decay time
  time += throwFiberFluorDecayTime();
  debug() << "reflected pulse: hit+scint+fluor time = " << time << endmsg;
  // need to add calculation of the non straight line distance
  double distanceGreen =  plane->distanceToMirror(hitCenter, id) +
    plane->getStripHalfLength(id)*2.;
  debug() << "reflected pulse: distance = " << distanceGreen << "; ( to mirror = "
          << plane->distanceToMirror(hitCenter, id) << "; Lstrip = "
          << plane->getStripHalfLength(id)*2. << " )" << endmsg;
  // add green fiber travel time
  time += travelTimeGreenFiber(distanceGreen);
  debug() << "reflected pulse: hit+scint+fluor+green time = " << time << endmsg;
  // attenuation in green
  double atten = attenuationGreenFiber(distanceGreen);
  debug() << "reflected pulse: green attenuation = " << atten << endmsg;
  // add clear fiber travel time
  time += travelTimeClearFiber(m_clearFiberLength);
  debug() << "reflected pulse: hit+scint+fluor+green+clear time = " << time << endmsg;
  // attenuation in clear
  atten *= attenuationClearFiber(m_clearFiberLength);
  debug() << "reflected pulse: green*clear attenuation = " << atten << endmsg;
  // pe
  pe *= atten*m_ReflectionCoefficient*m_fudgeFactor;
```

```
  debug() << "reflected pulse: pe at pmt = " << pe << endmsg;
  // return Pulse
  return PulsePtr(new Pulse(id.id(), hit, time, pe));

}


//=============================================================================
// createReflectedIDPulse() - create a reflected pulse in the ID  of a
//   given #pe, calculate propagation time, attenuation
//=============================================================================

PulsePtr OpticalModel::createReflectedODPulse(BarID id, const MCHitPtr hit, double pe)
const {

  // Distance to mirror + full strip length
  Gaudi::XYZPoint hitCenter = hit->entry() + hit->displacement()/2.;
  const DeODFrame* frame = m_deOuterDetector->getDeODFrame(hitCenter);

  // time of the hit
  double time = hit->time();
  debug() << "reflected pulse: hit time = " << time << endmsg;
  // add scint fluor decay time
  time += throwScintFluorDecayTime();
  debug() << "reflected pulse: hit+scint time = " << time << endmsg;
  // add fiber fiber decay time
  time += throwFiberFluorDecayTime();
  debug() << "reflected pulse: hit+scint+fluor time = " << time << endmsg;
  // need to add calculation of the non straight line distance
  double distanceGreen =  frame->distanceToMirror(hitCenter, id) +
    frame->getBarHalfLength(id)*2.;
  debug() << "reflected pulse: distance = " << distanceGreen << "; ( to mirror = "
          << frame->distanceToMirror(hitCenter, id) << "; Lstrip = "
          << frame->getBarHalfLength(id)*2. << " )" << endmsg;
  // add green fiber travel time
  time += travelTimeGreenFiber(distanceGreen);
  debug() << "reflected pulse: hit+scint+fluor+green time = " << time << endmsg;
  // attenuation in green
  double atten = attenuationGreenFiber(distanceGreen);
  debug() << "reflected pulse: green attenuation = " << atten << endmsg;
  // add clear fiber travel time
  time += travelTimeClearFiber(m_clearFiberLength);
  debug() << "reflected pulse: hit+scint+fluor+green+clear time = " << time << endmsg;
  // attenuation in clear
  atten *= attenuationClearFiber(m_clearFiberLength);
  debug() << "reflected pulse: green*clear attenuation = " << atten << endmsg;
  // pe
  pe *= atten*m_ReflectionCoefficient*m_fudgeFactor;
  debug() << "reflected pulse: pe at pmt = " << pe << endmsg;
  // return Pulse
  return PulsePtr(new Pulse(id.id(), hit, time, pe));

}


//=============================================================================
// calculate #pe's according to Birk's law
//=============================================================================

double OpticalModel::birksPe(double dE, double dx) const {

  static const double kTinyPath = 1.e-3*CLHEP::mm;
```

```
  double dEdx = 0.;
  if ( dx > kTinyPath ) {
    dEdx = dE/dx;
  } else {
    // in case path length is 0 - pretend that it's 1 mm
    dEdx = dE/CLHEP::mm;
  }
  double meanPE = (m_overallLightOutput*dE/(1. + m_birksConstant*dEdx));
  Rndm::Numbers poisson(m_randSvc , Rndm::Poisson(meanPE) );
  return poisson();

}

//=========================================================================
// throw scintillator fluor decay time
//=========================================================================

double OpticalModel::throwScintFluorDecayTime() const {

  static Rndm::Numbers scintFluorT(m_randSvc ,
Rndm::Exponential(m_scintFluorDecayTime) );
  return scintFluorT();

}

//=========================================================================
// throw fiber fluor decay time
//=========================================================================

double OpticalModel::throwFiberFluorDecayTime() const {

  static Rndm::Numbers fiberFluorT(m_randSvc ,
Rndm::Exponential(m_fiberFluorDecayTime) );
  return fiberFluorT();

}

//=========================================================================
// travel time in the green fiber
//=========================================================================

double  OpticalModel::travelTimeGreenFiber(double distance) const {

  return distance*m_greenFiberRefractionIndex/c_light;

}

//=========================================================================
// travel time in the clear fiber
//=========================================================================

double  OpticalModel::travelTimeClearFiber(double distance) const {

  return distance*m_clearFiberRefractionIndex/c_light;

}

//=========================================================================
// attenuation in the green fiber
//=========================================================================

double  OpticalModel::attenuationGreenFiber(double distance) const {
```

```
  debug() << "distance = " << distance << endmsg;
  debug() << "att length1 = " << m_greenFiberAttenuationLength1 << endmsg;
  double att =
m_greenFiberAttenuationAmplitude1*exp(-distance/m_greenFiberAttenuationLength1) +
    m_greenFiberAttenuationAmplitude2*exp(-distance/m_greenFiberAttenuationLength2);
  debug() << "att = " << att << endmsg;
  return att;

}

//=========================================================================
// attenuation in the clear fiber
//=========================================================================

double  OpticalModel::attenuationClearFiber(double distance) const {

  debug() << "distance = " << distance << endmsg;
  debug() << "att length1 = " << m_clearFiberAttenuationLength1 << endmsg;
  double att =
m_clearFiberAttenuationAmplitude1*exp(-distance/m_clearFiberAttenuationLength1) +
    m_clearFiberAttenuationAmplitude2*exp(-distance/m_clearFiberAttenuationLength2);
  debug() << "att = " << att << endmsg;
  return att;

}


//=========================================================================
//=========================================================================
//=========================================================================
```