```
// ========================================================================
//
// Developed by Bryan V. Egner, Darren E. Holland, and Julie V. Logan
// Modified by Darren Holland 2020-11-02
// ========================================================================
//
// This file is the main Geant4 .cc file for the RSM Geant4 Package. Compiling
this
// code for a design will run Geant4 in parallel for each measurement angle.
// ========================================================================
//
#include "B4DetectorConstruction.hh"
#include "B4aActionInitialization.hh"
#include "G4SystemOfUnits.hh"
#ifdef G4MULTITHREADED
#include "G4MTRunManager.hh"
#else
#include "G4RunManager.hh"
#endif
#include "G4UImanager.hh"
#include "G4UIcommand.hh"
#include "FTFP_BERT.hh"    // Photon PHysics -->  Recommended for HEP, Bertinia
Cascade
#include "Randomize.hh"
#include "G4VisExecutive.hh"
#include "G4UIExecutive.hh"
#include <math.h>
#include <cmath>
#include <stdio.h>
#include <string>
#include <sstream>
#include <stdlib.h>
#include <time.h>
#include <vector>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <iomanip>
#include "Settings.hh"
#include "G4MPImanager.hh"
#include "G4MPIsession.hh"
#include <sys/stat.h>
#include <thread>
// ========================================================================
//
// Load any default command options
// ========================================================================
//
using namespace std;

namespace {
  void PrintUsage() {
    G4cerr << " Usage: " << G4endl;
    G4cerr << " myMesh [-m macro ] [-u UIsession] [-t nThreads]" << G4endl;
    G4cerr << "   note: -t option is available only for multi-threaded mode."
           << G4endl;
  }
}

// ========================================================================
//
// Begin Main Program
// ========================================================================
//
```

```
int main(int argc,char** argv)
{
    // Choose the Random engine:
    G4Random::setTheEngine(new CLHEP::RanecuEngine);
    // At first, G4MPImanager/G4MPIsession should be created for running on
multiple nodes using MPI.
    G4MPImanager* g4MPI= new G4MPImanager(argc,argv);

    // MPI session (G4MPIsession) instead of G4UIterminal
    G4MPIsession* session= g4MPI-> GetMPIsession();

    // Load any macros:
    if ( argc > 7 ) {
      PrintUsage();
      return 1;
    }
    G4String macro;

    // Set default number of threads to use:
    #ifdef G4MULTITHREADED
      G4int nThreads = 1;
    #endif
    for ( G4int i=1; i<argc; i=i+2 ) {
        if      ( G4String(argv[i]) == "-m" ) macro = argv[i+1];
    #ifdef G4MULTITHREADED
        else if ( G4String(argv[i]) == "-t" ) {
          nThreads = G4UIcommand::ConvertToInt(argv[i+1]);
        }
    #endif
        else {
          PrintUsage();
          return 1;
        }
    }

    // Construct the MPI or default run manager (if Geant install didn't include
multithreading):
    #ifdef G4MULTITHREADED
      auto runManager = new G4MTRunManager;
      if ( nThreads > 0 ) {
        runManager->SetNumberOfThreads(nThreads);
      }
    #else
      auto runManager = new G4RunManager;
    #endif

    // Set mandatory initialization classes for run manager:
    // Register detector:
    auto detConstruction = new B4DetectorConstruction();
    runManager->SetUserInitialization(detConstruction);

    // Register the physics list:
    auto physicsList = new FTFP_BERT(0);  // Photon Physics List
    runManager->SetUserInitialization(physicsList);

    // Register the user action initialization - SetUserAction of
    // PrimaryGeneratorAction, RunAction, EventAction, SteppingAction
    // are found in the Build function of this class
    auto actionInitialization = new B4aActionInitialization();
    runManager->SetUserInitialization(actionInitialization);

    // Get the pointer to the User Interface manager:
    auto UImanager = G4UImanager::GetUIpointer();
```

```
    // Initialize run manager
    runManager-> Initialize();

    // Reduce printed output by setting process verbosities to 0
    UImanager->ApplyCommand("/mpi/verbose 0");
    UImanager->ApplyCommand("/run/verbose 0");
    UImanager->ApplyCommand("/event/verbose 0");
    UImanager->ApplyCommand("/process/verbose 0");
    UImanager->ApplyCommand("/process/em/verbose 0");
    UImanager->ApplyCommand("/vis/verbose 0");
    UImanager->ApplyCommand("/tracking/verbose 0");

    // Initialize a random seed (using the system time):
    thread_local thread::id threadid = std::this_thread::get_id();
    thread_local stringstream tstr;
    tstr << threadid;
    thread_local G4long tlong = stol(tstr.str());
    G4long seed = time(NULL) + tlong;
    G4Random::setTheSeed(seed);

    // ===== SOURCE DISTANCE in cm  ---> Settings::<> means pulled from
Settings.cc file
    //z down toward x/y plane
    double radius = Settings::SourceDist;

    //
============================================================================== //
    // Begin source rototation and runs
    //
============================================================================== //
    // Increment Phi by deltaphi around the z axis (the way the mask rotates)
    for (double phi=Settings::StartPhi; phi <= Settings::EndPhi; phi =
phi+Settings::deltaphi) {
        // Create string instances (for commands)
        ostringstream commandOS;
        ostringstream commandOSx;
        ostringstream commandOSxy;
        ostringstream numParticles;
        ostringstream energyParticles;
        G4String command = "";

        // Increment the theta position:
        for (double
theta=Settings::deltatheta*(0.5-floor(Settings::SourceDiv/2)/Settings::SourceDiv
); theta<360; theta = theta+Settings::deltatheta / Settings::SourceDiv) {

          // Set the source location:
          double xCoord =
radius*cos(theta*3.14159265358979/180.0)*sin(phi*3.14159265358979/180.0);
          double yCoord =
radius*sin(theta*3.14159265358979/180.0)*sin(phi*3.14159265358979/180.0);
          double zCoord = radius*cos(phi*3.14159265358979/180.0);
          commandOS << "/gps/pos/centre " << xCoord << " " << yCoord << " " <<
zCoord << " " << " cm";
          UImanager->ApplyCommand(G4String(commandOS.str()));

          // Set the source direction:
          // build the command of the rotation matrix (calculated as the vectors
normal to both x-axis and (xCoord,yCoord,zCoord) and both y-axis and
(xCoord,yCoord,zCoord))
          // calculated by finding any vector whose dot product with
(xCoord,yCoord,zCoord)=0 and then crossing the two, use the two obtained as the
two vectors you define
          commandOSx << "/gps/ang/rot1 "  << -yCoord << " " << xCoord << " " <<
```

```
0; //dot produce = 0 aka orthogonal to source vector
          commandOSxy << "/gps/ang/rot2 " << xCoord*zCoord << " " <<
zCoord*yCoord << " " << (-yCoord*yCoord - xCoord*xCoord); //cross product to
find third direction vector
          // orthogonal to source direction and dotted vector  Note: Sign
flipped to negative direction
          UImanager->ApplyCommand(G4String(commandOSx.str()));
          UImanager->ApplyCommand(G4String(commandOSxy.str()));

          // ===== Set the viewpoint theta to be from the source:
          ostringstream commandOSViewVector;

          // Reset strings for next source position
            commandOS.str("");
          commandOSx.str("");
          commandOSxy.str("");
          numParticles.str("");
          energyParticles.str("");

          //
========================================================================== //
          // Set source information
          //
========================================================================== //
          // Single-Energy Source:
          if ( Settings::SourceEnergyType == "none" ) {
                energyParticles << "/gps/energy " << Settings::energiesMeV << "
MeV";
                UImanager->ApplyCommand(G4String(energyParticles.str()));
          }
          else {
              // Multi-Energy Source:
              energyParticles << "/mpi/execute " << Settings::SourceEnergyType;
              UImanager->ApplyCommand(G4String(energyParticles.str()));
          }
          // Set particle type (neutron, gamma)
          string command = "/gps/particle " + Settings::PartType;
          UImanager->ApplyCommand(command);
          // Set to isotropic source
          command = "/gps/ang/type iso";
          UImanager->ApplyCommand(command);
          // Source cone for variance reduction:
          string ctheta = to_string(180 - Settings::coneangle);
          string mintheta = "/gps/ang/mintheta " + ctheta + " deg";
          UImanager->ApplyCommand(mintheta);
          command = "/gps/ang/maxtheta 180 deg";
          UImanager->ApplyCommand(command);

          //
========================================================================== //
          // Start MC runs!
          //
========================================================================== //
          // Set number of particles to run:
          numParticles << "/mpi/beamOn " << Settings::nParts2Run;
          // Run particles
          UImanager->ApplyCommand(G4String(numParticles.str()));
        }
    }
    //
========================================================================== //
    // Job termination:
    // Free the store: user actions, physics_list and detector_description are
    // owned and deleted by the run manager, so they should not be deleted
```

```
    // in the main() program !
    //
============================================================================ //
    delete g4MPI;
    delete runManager;
    //
============================================================================ //
    // Save the output filenames (source theta and phi positions) so wrapper
code
    // can combine the thread output data into one file
    //
============================================================================ //
    G4int runNum = 0;
    for (double phi=Settings::StartPhi; phi <= Settings::EndPhi; phi =
phi+Settings::deltaphi) {
        for (double
theta=Settings::deltatheta*(0.5-floor(Settings::SourceDiv/2)/Settings::SourceDiv
); theta<360; theta = theta+Settings::deltatheta / Settings::SourceDiv) {
            // Create output string for theta
            std::ostringstream streamObj;
            streamObj << std::fixed;
            streamObj << std::setprecision(3);
            streamObj << theta;
            string s_theta;
            string s_phi;
            // Create theta string
            if (theta+0.0005 < 10) s_theta="00" + streamObj.str();
                else if (theta+0.0005 < 100) s_theta="0" + streamObj.str();
            else s_theta=streamObj.str();

            // Create output string for phi
            std::ostringstream streamObj2;
            streamObj2 << std::fixed;
            streamObj2 << std::setprecision(3);
            streamObj2 << phi;
            // Create phi string
            if (phi+0.0005 < 10) s_phi="00" + streamObj2.str();
            else if (phi+0.0005 < 100) s_phi="0" + streamObj2.str();
            else s_phi=streamObj2.str();

            // Create filename string
            string fname_out_new = Settings::fname_out +"th" + s_theta + "ph" +
s_phi + ".o";

            // Connect run number to source position
            string runNum_str = G4UIcommand::ConvertToString(runNum);
            string fname_thetaphi="./" + runNum_str + "/SourcePos.txt";

            // Write filename to SourcePos.txt so wrapper code has info
            ofstream ofile;
            ofile.open (fname_thetaphi, ios::out);
            ofile << fname_out_new;
            ofile.close();
            runNum = runNum+1;
            }
    }

    // Create .comp file to track thread completion
    string fname_finished=Settings::fname_out + ".comp";
    ofstream ofile;
    ofile.open (fname_finished, ios::out | ios::app);
    // Append to file that the thread has finished (when all threads finish
    // the number of lines of text in this file should be equal to the number of
threads)
```

```
    ofile << "Thread is finished\n";
    ofile.close();
}
```