Revealing the invisible : an artistic exploration of particle showers

Clipet **Baptiste**, Delevoye **Anatole**, Godbille **Violette**, Maurice **Louis**,
Papillon **Maxime**, Reerink **Arthur**, Torris **Valentine**, Trepant **Hugo**.

Lycée de la Croix Blanche – Bondues, FRANCE

March 13, 2026

What is it about ?

Science, like art, can sometimes seem beyond the reach of the uninitiated; indeed, artists, like scientists, question reality, try to explain their environment, and even unveil the invisible. If artists see sensations and ideas in the invisible, for scientists it is more akin to physics.

The links between these two fields are strong : we can cite the deep desire to make the world around us more understandable, the choice of mediums, the control of conditions, and even the observation of effects produced. For both of them, experimentation occupies a central place.

Serendipity also plays an essential role in these two universes. In science as in art, some discoveries emerge from an unexpected result, an anomaly, or an unforeseen observation. In the artistic field, these accidents can become material for creation and open new perspectives for understanding reality.

Our proposal falls within this context...

Experiment proposal : make art with particles

The purpose of our experiment is to create images that can then constitute works of art. To do this, we want to irradiate a target stack of thin foils by a charged-particle beam. The target will be made up of layers of metal (1mm) and radiochromic films (EBT3/EBT4 type, 0,28mm) which would serve as a detector that make the particle

shower visible. Indeed, the advantage of these films is that they keep a permanent trace of irradiation in the form of optical density variations.

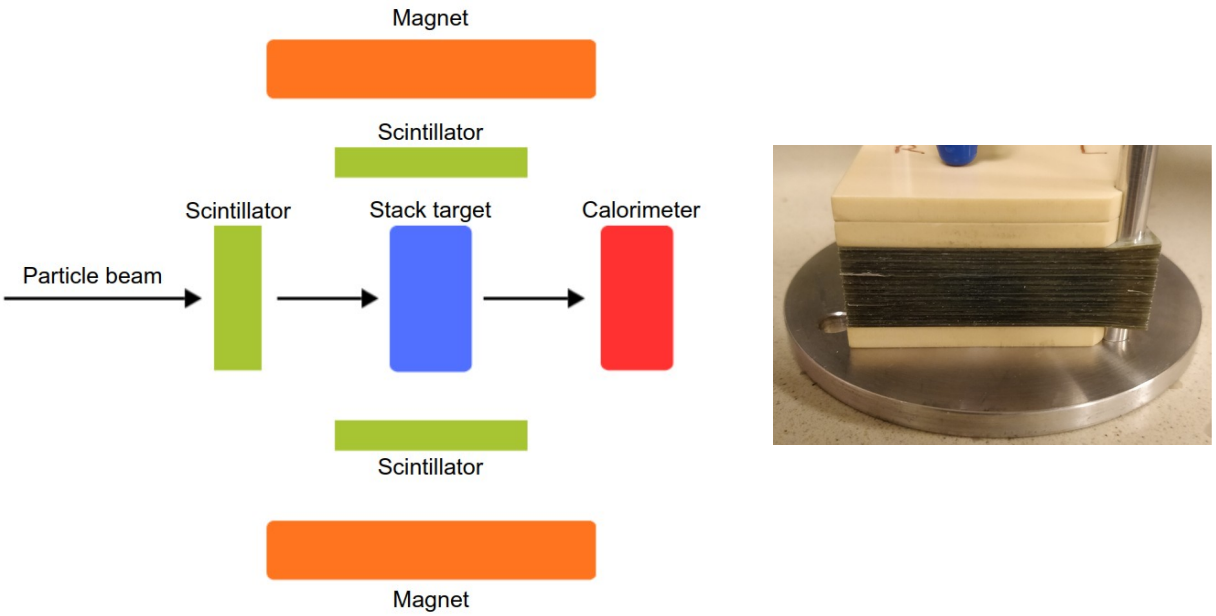

Fig n°1 : Experimental setup diagram and a stack target example.

When the particles interact with the first layer of the stack, a shower of particles is created which itself causes new showers, and so on... We will be able to detect all these showers thanks to the radiochromic films between the metal layers (fig n°2). Radiochromic films locally record the energy deposited, thus producing a series of imprints corresponding to different depths in matter. The whole constitutes a kind of experimental tomography of the shower, a series of images in time and depth.

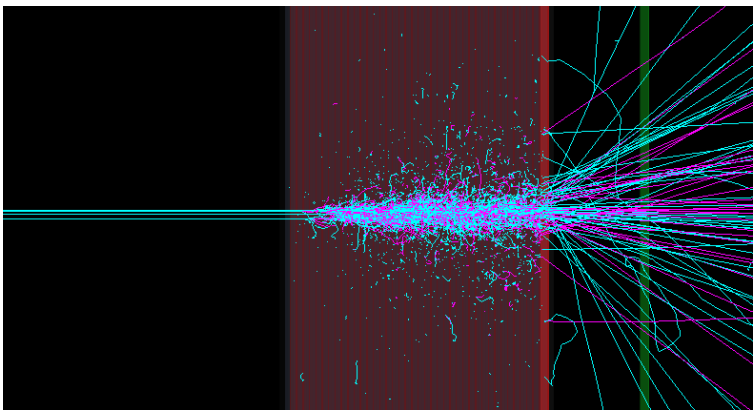

Fig n°2 : Example of a particle shower into the stack target in our Geant4 [1] simulation

We will repeat this experiment several times by changing the parameters : different metals, different particle beams and different energies. Each of these experiments will yield different results, allowing us to construct an artistic series. Each experiment (and therefore each element of the artistic series) will bear the signature of the interaction of the particles with the metal. We will thus give them a name of the form "***E*** GeV ***particle*** beam on ***metal***".

We would like to be able to immerse the target in a magnetic field of approximately 1 Tesla in order to potentially widen the shower. Finally, it may be possible to space the layers for even better results.

Once the experiment is complete, we will digitize [2] the radiochromic films in high resolution and digitally process the images. We will colorize the images for a more striking visual impact and adjust the brightness, contrast, etc., for a better result. The artistic composition will ultimately be developed in collaboration with a visual artist for what is envisioned as a monumental exhibition (see outreach proposal).

Preparing our art-science project : a theoretical approach

The radiochromic films

Radiochromic films basically act as dosimeters; in this experiment, they will take on the role of particle detectors. The interaction between the radiochromic film and the ionizing radiation changes the structure of the film and thus its color [3]. This feature allows us to visualize the path taken by the particles in the stack. The invisible thus becomes visible.

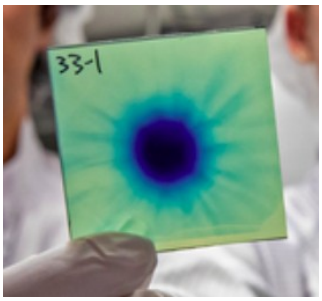

Fig n°3 : Radiochromic film hit by a proton beam.

As part of this project, the images obtained can be analyzed to estimate the effective spatial resolution of this detection system. This resolution corresponds to the smallest spatial structure that the film device and the scanner are able to distinguish. It depends on the intrinsic properties of the film, the thickness of the sensitive layer, the energy deposition by the particles, and the characteristics of the scanning system.

We can define, in a simplified way, the spatial resolution as the minimum distance Δx allowing for the distinction of two neighboring energy deposits. If the image is digitized with a resolution of R pixels per inch (dpi), the physical size of a pixel is approximately : $\Delta x = \frac{25.4}{R} mm$. For example, for a scan at 600 dpi, the size of one pixel corresponds to approximately 0.042 mm. This scale sets an instrumental limit for observing recorded spatial structures.

Particle showers naturally produce energy distributions with spatial gradients and fine structures. The statistical analysis of these patterns, in particular through the study of

intensity profiles or spatial spectra, can provide information on the system's ability to reproduce these details. The project thus offers the possibility of studying how the real structure of energy deposits is transformed by the detector response and by the digitization process.

This experimental dimension adds scientific value to the artistic project : the images produced are not only visual interpretations of physical phenomena, but also experimental objects allowing us to question the resolution and representation limits of a widely used detector in physics and dosimetry.

Particle – matter interactions

When a beam of high-energy particles passes through a material, the incoming particles do not simply pass through it rectilinearly before stopping. They interact with the atoms in the material and split to produce new particles. These successive interactions give rise to a flow of secondary particles, called particle shower. In a simplified model, we can imagine that for each interaction a particle produces two secondary particles. After n successive interactions, the number of particles is then approximately : $N(n) = 2^n$. This rapid growth explains why a single incident particle can produce a shower containing a large number of particles.

However, total energy is conserved. If the initial energy is E_0, the average energy per particle after n interactions becomes : $E_n = \frac{E_0}{2^n}$. When the energy becomes too low, interactions gradually cease and the imprint of the shower on the films goes out. There is therefore a point where the number of particles is maximal, called the shower maximum (see fig n°4).

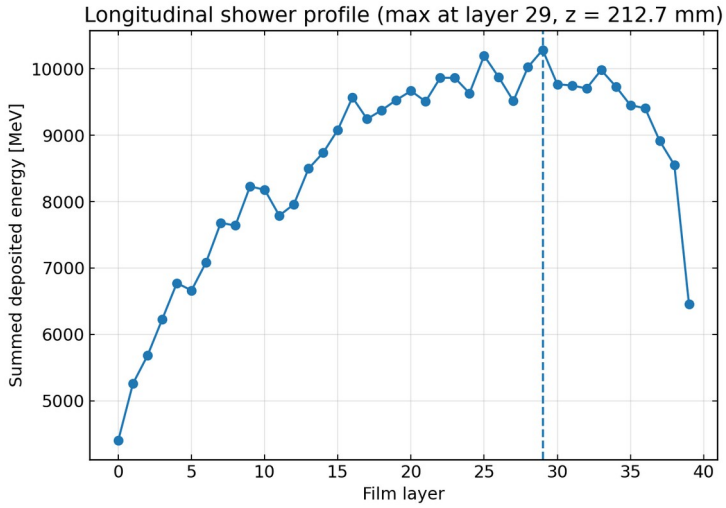

Fig n°4 : Longitudinal shower profile. Geant4 simulation with 15000 runs of 6 GeV protons and Pions (50%, 50%) on a Tungsten stack target (1 mm). Maximum shower is at film n°29.

Theoretical calculations are very complex, so we took the time to create a simulation project using Geant4 software. We wrote the project in such a way that it is possible to modify the experimental parameters : the beam particles, their energy, the target material, the thickness, the magnetic field, etc. The project's source code is included in the appendix.

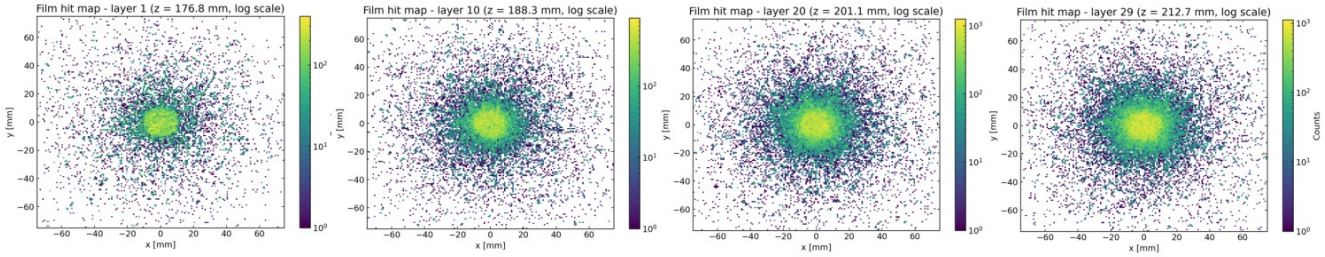

Fig n°5 : Evolution of the film hit map with the layer. Geant4 simulation with 15000 runs of 6 GeV protons and Pions (50%, 50%) on a Tungsten stack target (1 mm). Maximum shower is at film n°29.

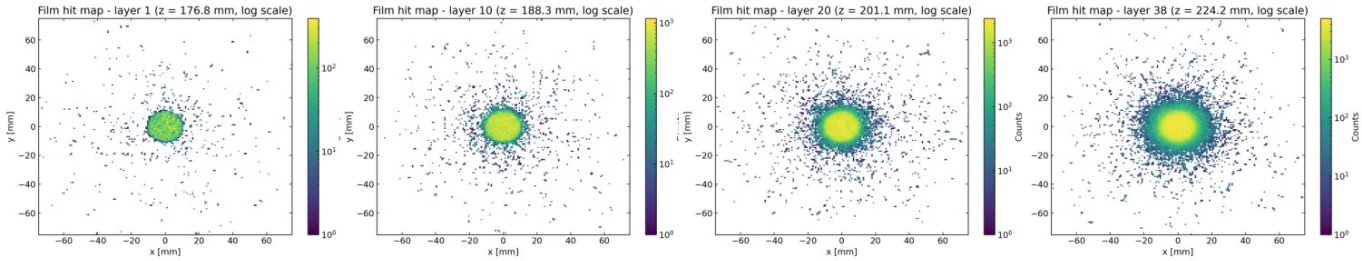

Fig n°6 : Evolution of the film hit map with the layer. Geant4 simulation with 15000 runs of 3 GeV electrons on a Iron stack target (1 mm). Maximum shower is at film n°39.

Thanks to the different simulation results, we can predict the statistical shape of the particle shower and thus optimize the parameters of the real experiment to obtain the best result on the films. We hope that this will allow us to create our work of art thanks to the BL4S competition.

Outreach proposal

We want to show, through our proposal, that scientific research can also be creative, visual, and surprising.

We propose to do a workshop that will be based on the images obtained from our experiment using radiochromic films exposed to a particle beam in the accelerator. Students will discover the patterns created by the interaction of particles with matter, playing with contrast. These images will be presented as a form of “drawing made by particles,” where each trace creates a picture.

We will emphasize an essential aspect of scientific research : it is impossible to predict exactly what patterns will appear. Like in a work of art, the final result depends on many parameters and always contains an element of unpredictability.

Students will then be invited to create their own drawings inspired by these scientific images, using stencils, and colors. The activity will be both artistic and playful, while illustrating how scientists observe and interpret patterns in order to understand invisible phenomena.

Here, science becomes more accessible and democratic: it is no longer only about complex calculations, which can seem difficult or unappealing to a non-specialist audience.

We also have planed to invite the visual artist who would help us with the composition to give a public lecture open to all to present the design stages of a work of art on the theme of science.

Références

[1] *Geant4 website*

https://geant4.web.cern.ch/

[2] Changes in scanning orientation effects of Gafchromic EBT-3 film irradiated with ultra-high dose rate proton beams, Hiroshi Yasuda, Hassna Bantan, Masumi Umezawa, Masashi Yamada, Katsunori Yogo & Toshiyuki Toshito

https://link.springer.com/article/10.1007/s42452-025-06484-6

[3] A review on radiochromic film dosimetry in radiation therapy, Arash Darafsheh, Hamid Ghaznavi

https://pmc.ncbi.nlm.nih.gov/articles/PMC12672139/

Appendix

TheUnveilers Geant4 project source code

****** CMakeLists.txt ******

```
cmake_minimum_required(VERSION 3.16)
project(TheUnveilers VERSION 2.0 LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)

find_package(Geant4 REQUIRED COMPONENTS ui_all vis_all)
include(${Geant4_USE_FILE})

include_directories(${PROJECT_SOURCE_DIR}/include)
file(GLOB THEUNVEILERS_SOURCES CONFIGURE_DEPENDS ${PROJECT_SOURCE_DIR}/src/*.cc)

add_executable(TheUnveilers ${THEUNVEILERS_SOURCES})
target_link_libraries(TheUnveilers PRIVATE ${Geant4_LIBRARIES})

file(COPY ${PROJECT_SOURCE_DIR}/macros DESTINATION ${PROJECT_BINARY_DIR})
file(COPY ${PROJECT_SOURCE_DIR}/analysis DESTINATION ${PROJECT_BINARY_DIR})
file(COPY ${PROJECT_SOURCE_DIR}/README.md DESTINATION ${PROJECT_BINARY_DIR})
```

****** src/ActionInitialization.cc ******

```
#include "ActionInitialization.hh"
#include "DetectorConstruction.hh"
#include "PrimaryGeneratorAction.hh"
#include "RunAction.hh"
#include "EventAction.hh"
#include "SteppingAction.hh"

ActionInitialization::ActionInitialization(DetectorConstruction* det)
: fDet(det)
{}

void ActionInitialization::Build() const
{
   SetUserAction(new PrimaryGeneratorAction());
   SetUserAction(new RunAction(fDet));

   auto* eventAction = new EventAction(fDet);
   SetUserAction(eventAction);
   SetUserAction(new SteppingAction(eventAction));
}
```

****** src/DetectorConstruction.cc ******

```
#include "DetectorConstruction.hh"
#include "DetectorMessenger.hh"
#include "SimpleEMField.hh"

#include "G4Box.hh"
#include "G4ChordFinder.hh"
#include "G4ClassicalRK4.hh"
#include "G4Colour.hh"
#include "G4EqMagElectricField.hh"
#include "G4FieldManager.hh"
#include "G4LogicalVolume.hh"
#include "G4MagIntegratorDriver.hh"
#include "G4MagIntegratorStepper.hh"
#include "G4Material.hh"
#include "G4NistManager.hh"
#include "G4PVPlacement.hh"
#include "G4RunManager.hh"
#include "G4SystemOfUnits.hh"
#include "G4TransportationManager.hh"
#include "G4VisAttributes.hh"
#include "G4UImanager.hh"

#include <algorithm>
#include <cctype>
#include <cmath>
#include <sstream>

namespace {
G4String Trim(const G4String& s)
{
   const auto begin = s.find_first_not_of(" \t\n\r");
   if (begin == std::string::npos) return "";
   const auto end = s.find_last_not_of(" \t\n\r");
   return s.substr(begin, end - begin + 1);
}

G4String Upper(G4String s)
{
   std::transform(s.begin(), s.end(), s.begin(), [](unsigned char c) { return std::toupper(c); });
   return s;
}
}
```

```
DetectorConstruction::DetectorConstruction()
{
   auto* nist = G4NistManager::Instance();
   nist->FindOrBuildMaterial("G4_AIR");

   fMaterialAliases = {
      {"W", "G4_W"},
      {"TUNGSTEN", "G4_W"},
      {"PB", "G4_Pb"},
      {"LEAD", "G4_Pb"},
      {"FE", "G4_Fe"},
      {"IRON", "G4_Fe"},
      {"CU", "G4_Cu"},
      {"COPPER", "G4_Cu"},
      {"AL", "G4_Al"},
      {"ALUMINUM", "G4_Al"},
      {"ALUMINIUM", "G4_Al"},
      {"SI", "G4_Si"},
      {"SILICON", "G4_Si"},
      {"C", "G4_C"},
      {"CARBON", "G4_C"},
      {"GRAPHITE", "G4_GRAPHITE"},
      {"PMMA", "G4_PLEXIGLASS"},
      {"PLEXI", "G4_PLEXIGLASS"},
      {"MYLAR", "G4_MYLAR"},
      {"KAPTON", "G4_KAPTON"},
      {"WATER", "G4_WATER"},
      {"PS", "G4_POLYSTYRENE"},
      {"POLYSTYRENE", "G4_POLYSTYRENE"},
      {"SCINT", "G4_PLASTIC_SC_VINYLTOLUENE"},
      {"SCINTILLATOR", "G4_PLASTIC_SC_VINYLTOLUENE"},
      {"STEEL", "G4_STAINLESS-STEEL"},
      {"STAINLESS", "G4_STAINLESS-STEEL"},
      {"BRASS", "G4_BRASS"},
      {"RCF_ACTIVE", "RCF_Active"},
      {"RCF_EBT3", "RCF_EBT3"}
   };

   DefineMaterials();
   fMessenger = new DetectorMessenger(this);
}

DetectorConstruction::~DetectorConstruction()
{
   delete fMessenger;
   delete fChordFinder;
   delete fIntegrationDriver;
   delete fStepper;
   delete fEquation;
   delete fEMField;
}

void DetectorConstruction::DefineMaterials()
{
   auto* nist = G4NistManager::Instance();
   fWorldMat = nist->FindOrBuildMaterial("G4_AIR");

   if (!G4Material::GetMaterial("RCF_Active", false)) {
      auto* H = nist->FindOrBuildElement("H");
      auto* C = nist->FindOrBuildElement("C");
      auto* O = nist->FindOrBuildElement("O");
      auto* rcfActive = new G4Material("RCF_Active", 1.20 * g / cm3, 3);
      rcfActive->AddElement(H, 5);
      rcfActive->AddElement(C, 6);
      rcfActive->AddElement(O, 2);
   }

   if (!G4Material::GetMaterial("RCF_EBT3", false)) {
      auto* H = nist->FindOrBuildElement("H");
      auto* C = nist->FindOrBuildElement("C");
      auto* O = nist->FindOrBuildElement("O");
      auto* rcfEbt3 = new G4Material("RCF_EBT3", 1.35 * g / cm3, 3);
      rcfEbt3->AddElement(H, 8);
      rcfEbt3->AddElement(C, 10);
      rcfEbt3->AddElement(O, 4);
   }

   fDefaultAbsMat = FindMaterialFromToken(fAbsMatName);
   fFilmMat = FindMaterialFromToken(fFilmMatName);
}

G4String DetectorConstruction::ResolveMaterialName(const G4String& token) const
{
   const auto clean = Trim(token);
   const auto upper = Upper(clean);
   const auto it = fMaterialAliases.find(upper);
   if (it != fMaterialAliases.end()) return it->second;
   return clean;
}

G4Material* DetectorConstruction::FindMaterialFromToken(const G4String& token) const
{
   const auto resolved = ResolveMaterialName(token);
   auto* mat = G4Material::GetMaterial(resolved, false);
   if (!mat) {
```

```
        mat = G4NistManager::Instance()->FindOrBuildMaterial(resolved, false);
    }
    return mat ? mat : G4NistManager::Instance()->FindOrBuildMaterial("G4_W");
}

std::vector<G4String> DetectorConstruction::ParseCSV(const G4String& text) const
{
    std::vector<G4String> out;
    std::stringstream ss(text);
    std::string item;
    while (std::getline(ss, item, ',')) {
        const auto t = Trim(item);
        if (!t.empty()) out.push_back(t);
    }
    return out;
}

void DetectorConstruction::RequestGeometryUpdate()
{
    auto* runManager = G4RunManager::GetRunManager();
    if (runManager) {
        runManager->GeometryHasBeenModified();
        runManager->ReinitializeGeometry(true);
    }
    RefreshVisualization();
}

void DetectorConstruction::RefreshVisualization()
{
    auto* ui = G4UImanager::GetUIpointer();
    if (!ui) return;

    ui->ApplyCommand("/vis/drawVolume");
    ui->ApplyCommand("/vis/scene/notifyHandlers");
    ui->ApplyCommand("/vis/viewer/rebuild");
    ui->ApplyCommand("/vis/viewer/refresh");
}

void DetectorConstruction::SetNumberOfLayers(G4int n)
{
    if (n > 0) {
        fNLayers = n;
        RequestGeometryUpdate();
    }
}

void DetectorConstruction::SetAbsorberMaterial(const G4String& name)
{
    fAbsMatName = name;
    fDefaultAbsMat = FindMaterialFromToken(name);
    RequestGeometryUpdate();
}

void DetectorConstruction::SetFilmMaterial(const G4String& name)
{
    fFilmMatName = name;
    fFilmMat = FindMaterialFromToken(name);
    RequestGeometryUpdate();
}

void DetectorConstruction::SetLayerPattern(const G4String& csv)
{
    fLayerPatternTokens = ParseCSV(csv);
    RequestGeometryUpdate();
}

void DetectorConstruction::SetAbsorberThickness(G4double value)
{
    if (value > 0.) {
        fAbsThickness = value;
        RequestGeometryUpdate();
    }
}

void DetectorConstruction::SetFilmThickness(G4double value)
{
    if (value > 0.) {
        fFilmThickness = value;
        RequestGeometryUpdate();
    }
}

void DetectorConstruction::SetGapThickness(G4double value)
{
    if (value >= 0.) {
        fGapThickness = value;
        RequestGeometryUpdate();
    }
}

void DetectorConstruction::SetStackXY(G4double value)
{
    if (value > 0.) {
        fStackXY = value;
        RequestGeometryUpdate();
```

```
    }
}

void DetectorConstruction::SetFieldX(G4double value) { fBx = value; UpdateField(); }
void DetectorConstruction::SetFieldY(G4double value) { fBy = value; UpdateField(); }
void DetectorConstruction::SetFieldZ(G4double value) { fBz = value; UpdateField(); }
void DetectorConstruction::SetElectricFieldX(G4double value) { fEx = value; UpdateField(); }
void DetectorConstruction::SetElectricFieldY(G4double value) { fEy = value; UpdateField(); }
void DetectorConstruction::SetElectricFieldZ(G4double value) { fEz = value; UpdateField(); }

G4VPhysicalVolume* DetectorConstruction::Construct()
{
    DefineMaterials();

    fFilmLogicals.clear();
    fAbsLogicals.clear();

    fPitch = fAbsThickness + fFilmThickness + fGapThickness;
    fStackLength = fNLayers * fPitch;
    fLeakPosZ = fStackPosZ + 0.5 * fStackLength + 2.0 * cm;

    auto* solidWorld = new G4Box("World", fWorldXY / 2., fWorldXY / 2., fWorldZ / 2.);
    auto* logicWorld = new G4LogicalVolume(solidWorld, fWorldMat, "World");
    auto* physWorld = new G4PVPlacement(nullptr, {}, logicWorld, "World", nullptr, false, 0, true);

    auto* solidStack = new G4Box("Stack", fStackXY / 2., fStackXY / 2., fStackLength / 2. + 1. * mm);
    auto* logicStack = new G4LogicalVolume(solidStack, fWorldMat, "Stack");
    new G4PVPlacement(nullptr,
                      G4ThreeVector(0, 0, fStackPosZ),
                      logicStack,
                      "Stack",
                      logicWorld,
                      false,
                      0,
                      true);

    auto* solidLeak = new G4Box("LeakPlane", fStackXY / 2., fStackXY / 2., fLeakThickness / 2.);
    fLeakLogical = new G4LogicalVolume(solidLeak, fWorldMat, "LeakPlane");
    new G4PVPlacement(nullptr,
                      G4ThreeVector(0, 0, fLeakPosZ),
                      fLeakLogical,
                      "LeakPlane",
                      logicWorld,
                      false,
                      0,
                      true);

    const G4double z0 = -0.5 * fStackLength;
    for (G4int i = 0; i < fNLayers; ++i) {
        G4Material* layerAbsMat = fDefaultAbsMat;
        if (!fLayerPatternTokens.empty()) {
            const auto& token = fLayerPatternTokens[static_cast<std::size_t>(i) % fLayerPatternTokens.size()];
            layerAbsMat = FindMaterialFromToken(token);
        }

        auto* solidAbs = new G4Box("Abs", fStackXY / 2., fStackXY / 2., fAbsThickness / 2.);
        auto* logicAbs = new G4LogicalVolume(solidAbs, layerAbsMat, "Abs");
        fAbsLogicals.push_back(logicAbs);

        auto* solidFilm = new G4Box("Film", fStackXY / 2., fStackXY / 2., fFilmThickness / 2.);
        auto* logicFilm = new G4LogicalVolume(solidFilm, fFilmMat, "Film");
        fFilmLogicals.push_back(logicFilm);

        const G4double zBase = z0 + i * fPitch;
        new G4PVPlacement(nullptr,
                          G4ThreeVector(0, 0, zBase + 0.5 * fAbsThickness),
                          logicAbs,
                          "Abs",
                          logicStack,
                          false,
                          i,
                          true);
        new G4PVPlacement(nullptr,
                          G4ThreeVector(0, 0, zBase + fAbsThickness + 0.5 * fFilmThickness),
                          logicFilm,
                          "Film",
                          logicStack,
                          false,
                          i,
                          true);
    }

    auto* worldVis = new G4VisAttributes();
    worldVis->SetVisibility(false);
    logicWorld->SetVisAttributes(worldVis);

    auto* stackVis = new G4VisAttributes(G4Colour(0.85, 0.85, 0.85, 0.04));
    stackVis->SetVisibility(true);
    logicStack->SetVisAttributes(stackVis);

    for (auto* absLV : fAbsLogicals) {
        auto* vis = new G4VisAttributes(G4Colour(0.35, 0.38, 0.48, 0.45));
        vis->SetForceSolid(true);
        absLV->SetVisAttributes(vis);
    }
```

```
    for (auto* filmLV : fFilmLogicals) {
        auto* vis = new G4VisAttributes(G4Colour(0.95, 0.12, 0.12, 0.45));
        vis->SetForceSolid(true);
        filmLV->SetVisAttributes(vis);
    }

    auto* leakVis = new G4VisAttributes(G4Colour(0.10, 0.90, 0.15, 0.30));
    leakVis->SetForceSolid(true);
    fLeakLogical->SetVisAttributes(leakVis);

    return physWorld;
}

void DetectorConstruction::ConstructSDandField()
{
    fFieldManager = G4TransportationManager::GetTransportationManager()->GetFieldManager();

    if (!fEMField) {
        fEMField = new SimpleEMField();
        fEquation = new G4EqMagElectricField(fEMField);
        fStepper = new G4ClassicalRK4(fEquation, 8);
        fIntegrationDriver = new G4MagInt_Driver(0.01 * mm, fStepper, fStepper->GetNumberOfVariables());
        fChordFinder = new G4ChordFinder(fIntegrationDriver);
    }

    UpdateField();
}

void DetectorConstruction::UpdateField()
{
    if (!fEMField || !fFieldManager) return;

    fEMField->SetFieldValue(G4ThreeVector(fBx, fBy, fBz), G4ThreeVector(fEx, fEy, fEz));

    if (fEMField->HasAnyField()) {
        fFieldManager->SetDetectorField(fEMField);
        fFieldManager->SetChordFinder(fChordFinder);
    } else {
        fFieldManager->SetDetectorField(nullptr);
        fFieldManager->SetChordFinder(nullptr);
    }
}

****** src/DetectorMessenger.cc ******

#include "DetectorMessenger.hh"
#include "DetectorConstruction.hh"

#include "G4ApplicationState.hh"
#include "G4UIdirectory.hh"
#include "G4UIcmdWithADoubleAndUnit.hh"
#include "G4UIcmdWithAString.hh"
#include "G4UIcmdWithAnInteger.hh"

DetectorMessenger::DetectorMessenger(DetectorConstruction* det)
: fDetector(det)
{
    fDetDir = new G4UIdirectory("/unveilers/det/");
    fDetDir->SetGuidance("Detector geometry control");

    fFieldDir = new G4UIdirectory("/unveilers/field/");
    fFieldDir->SetGuidance("Uniform electromagnetic field control");

    fLayersCmd = new G4UIcmdWithAnInteger("/unveilers/det/setLayers", this);
    fLayersCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fAbsMatCmd = new G4UIcmdWithAString("/unveilers/det/setAbsorberMaterial", this);
    fAbsMatCmd->SetGuidance("Examples: W, Pb, Fe, Cu, Al, Si, C, PMMA, WATER, STEEL");
    fAbsMatCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fFilmMatCmd = new G4UIcmdWithAString("/unveilers/det/setFilmMaterial", this);
    fFilmMatCmd->SetGuidance("Examples: RCF_EBT3, RCF_ACTIVE, G4_MYLAR, G4_KAPTON");
    fFilmMatCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fPatternCmd = new G4UIcmdWithAString("/unveilers/det/setLayerPattern", this);
    fPatternCmd->SetGuidance("Comma-separated absorber pattern, e.g. W,W,Pb,W,Cu");
    fPatternCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fAbsThickCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/det/setAbsThickness", this);
    fAbsThickCmd->SetUnitCategory("Length");
    fAbsThickCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fFilmThickCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/det/setFilmThickness", this);
    fFilmThickCmd->SetUnitCategory("Length");
    fFilmThickCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fGapThickCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/det/setGapThickness", this);
    fGapThickCmd->SetUnitCategory("Length");
    fGapThickCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fStackXYCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/det/setStackXY", this);
    fStackXYCmd->SetUnitCategory("Length");
    fStackXYCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
```

```
   fFieldXCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/field/setBx", this);
   fFieldXCmd->SetUnitCategory("Magnetic flux density");
   fFieldXCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

   fFieldYCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/field/setBy", this);
   fFieldYCmd->SetUnitCategory("Magnetic flux density");
   fFieldYCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

   fFieldZCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/field/setBz", this);
   fFieldZCmd->SetUnitCategory("Magnetic flux density");
   fFieldZCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

   fElectricXCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/field/setEx", this);
   fElectricXCmd->SetUnitCategory("Electric field");
   fElectricXCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

   fElectricYCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/field/setEy", this);
   fElectricYCmd->SetUnitCategory("Electric field");
   fElectricYCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

   fElectricZCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/field/setEz", this);
   fElectricZCmd->SetUnitCategory("Electric field");
   fElectricZCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
}

DetectorMessenger::~DetectorMessenger()
{
   delete fLayersCmd;
   delete fAbsMatCmd;
   delete fFilmMatCmd;
   delete fPatternCmd;
   delete fAbsThickCmd;
   delete fFilmThickCmd;
   delete fGapThickCmd;
   delete fStackXYCmd;
   delete fFieldXCmd;
   delete fFieldYCmd;
   delete fFieldZCmd;
   delete fElectricXCmd;
   delete fElectricYCmd;
   delete fElectricZCmd;
   delete fDetDir;
   delete fFieldDir;
}

void DetectorMessenger::SetNewValue(G4UIcommand* cmd, G4String value)
{
   if (cmd == fLayersCmd) {
      fDetector->SetNumberOfLayers(fLayersCmd->GetNewIntValue(value));
   } else if (cmd == fAbsMatCmd) {
      fDetector->SetAbsorberMaterial(value);
   } else if (cmd == fFilmMatCmd) {
      fDetector->SetFilmMaterial(value);
   } else if (cmd == fPatternCmd) {
      fDetector->SetLayerPattern(value);
   } else if (cmd == fAbsThickCmd) {
      fDetector->SetAbsorberThickness(fAbsThickCmd->GetNewDoubleValue(value));
   } else if (cmd == fFilmThickCmd) {
      fDetector->SetFilmThickness(fFilmThickCmd->GetNewDoubleValue(value));
   } else if (cmd == fGapThickCmd) {
      fDetector->SetGapThickness(fGapThickCmd->GetNewDoubleValue(value));
   } else if (cmd == fStackXYCmd) {
      fDetector->SetStackXY(fStackXYCmd->GetNewDoubleValue(value));
   } else if (cmd == fFieldXCmd) {
      fDetector->SetFieldX(fFieldXCmd->GetNewDoubleValue(value));
   } else if (cmd == fFieldYCmd) {
      fDetector->SetFieldY(fFieldYCmd->GetNewDoubleValue(value));
   } else if (cmd == fFieldZCmd) {
      fDetector->SetFieldZ(fFieldZCmd->GetNewDoubleValue(value));
   } else if (cmd == fElectricXCmd) {
      fDetector->SetElectricFieldX(fElectricXCmd->GetNewDoubleValue(value));
   } else if (cmd == fElectricYCmd) {
      fDetector->SetElectricFieldY(fElectricYCmd->GetNewDoubleValue(value));
   } else if (cmd == fElectricZCmd) {
      fDetector->SetElectricFieldZ(fElectricZCmd->GetNewDoubleValue(value));
   }
}

****** src/EventAction.cc ******

#include "EventAction.hh"
#include "DetectorConstruction.hh"

#include "G4AnalysisManager.hh"
#include "G4Event.hh"
#include "G4PrimaryParticle.hh"
#include "G4PrimaryVertex.hh"
#include "G4SystemOfUnits.hh"

EventAction::EventAction(DetectorConstruction* det)
: fDet(det)
{}

void EventAction::BeginOfEventAction(const G4Event*)
{
```

```
    fLayerEdep.assign(fDet->GetNumberOfFilms(), 0.0);
    fTotalFilmEdep = 0.0;
    fLeakEkin = 0.0;
}

void EventAction::AddFilmEdep(G4int layer, G4double e)
{
    if (layer >= 0 && layer < static_cast<G4int>(fLayerEdep.size())) {
        fLayerEdep[layer] += e;
        fTotalFilmEdep += e;
    }
}

void EventAction::AddLeakEkin(G4double e)
{
    fLeakEkin += e;
}

void EventAction::EndOfEventAction(const G4Event* evt)
{
    auto* man = G4AnalysisManager::Instance();

    G4int primaryPDG = 0;
    G4double primaryE = 0.0;
    if (evt->GetNumberOfPrimaryVertex() > 0) {
        auto* vtx = evt->GetPrimaryVertex(0);
        if (vtx && vtx->GetNumberOfParticle() > 0) {
            auto* p = vtx->GetPrimary(0);
            if (p) {
                primaryPDG = p->GetPDGcode();
                primaryE = p->GetKineticEnergy();
            }
        }
    }

    man->FillH1(2, fTotalFilmEdep / MeV);
    man->FillH1(3, fLeakEkin / MeV);

    man->FillNtupleIColumn(1, 0, evt->GetEventID());
    man->FillNtupleIColumn(1, 1, primaryPDG);
    man->FillNtupleDColumn(1, 2, primaryE / MeV);
    man->FillNtupleDColumn(1, 3, fTotalFilmEdep / MeV);
    man->FillNtupleDColumn(1, 4, fLeakEkin / MeV);
    man->AddNtupleRow(1);

    for (G4int i = 0; i < static_cast<G4int>(fLayerEdep.size()); ++i) {
        const G4double zCenter = fDet->GetStackStartZ() + i * fDet->GetPitch() +
                                 fDet->GetAbsorberThickness() + 0.5 * fDet->GetFilmThickness();
        man->FillH2(3, i, fLayerEdep[i] / MeV);

        man->FillNtupleIColumn(2, 0, evt->GetEventID());
        man->FillNtupleIColumn(2, 1, primaryPDG);
        man->FillNtupleIColumn(2, 2, i);
        man->FillNtupleDColumn(2, 3, zCenter / mm);
        man->FillNtupleDColumn(2, 4, fLayerEdep[i] / MeV);
        man->FillNtupleDColumn(2, 5, fLeakEkin / MeV);
        man->AddNtupleRow(2);
    }
}

****** src/main.cc ******

#include "G4RunManagerFactory.hh"
#include "G4UImanager.hh"
#include "G4UIExecutive.hh"
#include "G4VisExecutive.hh"

#include "FTFP_BERT.hh"
#include "G4EmStandardPhysics_option4.hh"

#include "DetectorConstruction.hh"
#include "ActionInitialization.hh"

int main(int argc, char** argv)
{
    auto* runManager = G4RunManagerFactory::CreateRunManager(G4RunManagerType::SerialOnly);

    auto* detector = new DetectorConstruction();
    runManager->SetUserInitialization(detector);

    auto* physics = new FTFP_BERT;
    physics->ReplacePhysics(new G4EmStandardPhysics_option4());
    runManager->SetUserInitialization(physics);

    runManager->SetUserInitialization(new ActionInitialization(detector));

    auto* visManager = new G4VisExecutive;
    visManager->Initialize();

    auto* uiManager = G4UImanager::GetUIpointer();
    auto* session = new G4UIExecutive(argc, argv);

    if (argc == 1) {
        uiManager->ApplyCommand("/control/execute macros/vis.mac");
    } else {
```

```
      G4String command = "/control/execute ";
      uiManager->ApplyCommand(command + G4String(argv[1]));
   }

   session->SessionStart();

   delete session;
   delete visManager;
   delete runManager;
   return 0;
}
```

****** src/PrimaryGeneretorAction.cc ******

```
#include "PrimaryGeneratorAction.hh"
#include "PrimaryGeneratorMessenger.hh"

#include "G4Event.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4RandomDirection.hh"
#include "G4SystemOfUnits.hh"
#include "Randomize.hh"

#include <algorithm>
#include <cmath>
#include <cctype>
#include <sstream>
#include <stdexcept>

namespace {
G4String Trim(const G4String& s)
{
   auto begin = s.find_first_not_of(" \t\n\r");
   if (begin == std::string::npos) return "";
   auto end = s.find_last_not_of(" \t\n\r");
   return s.substr(begin, end - begin + 1);
}
}

PrimaryGeneratorAction::PrimaryGeneratorAction()
{
   fGun = new G4ParticleGun(1);
   fMessenger = new PrimaryGeneratorMessenger(this);
   UpdateParticleDefinitions();
}

PrimaryGeneratorAction::~PrimaryGeneratorAction()
{
   delete fMessenger;
   delete fGun;
}

std::vector<G4String> PrimaryGeneratorAction::ParseCSV(const G4String& text) const
{
   std::vector<G4String> out;
   std::stringstream ss(text);
   std::string item;
   while (std::getline(ss, item, ',')) {
      const auto t = Trim(item);
      if (!t.empty()) out.push_back(t);
   }
   return out;
}

void PrimaryGeneratorAction::SetParticles(const G4String& csv)
{
   auto particles = ParseCSV(csv);
   if (!particles.empty()) {
      fParticleNames = particles;
      if (fWeights.size() != fParticleNames.size()) {
         fWeights.assign(fParticleNames.size(), 1.0);
      }
      UpdateParticleDefinitions();
   }
}

void PrimaryGeneratorAction::SetWeights(const G4String& csv)
{
   auto tokens = ParseCSV(csv);
   if (tokens.empty()) return;

   std::vector<G4double> w;
   for (const auto& tok : tokens) {
      w.push_back(std::max(0.0, std::stod(tok)));
   }
   if (w.size() == fParticleNames.size()) {
      fWeights = w;
   }
}

void PrimaryGeneratorAction::SetDirection(const G4ThreeVector& d)
{
   if (d.mag2() > 0.) {
      fDirection = d.unit();
```

```
    }
}

void PrimaryGeneratorAction::UpdateParticleDefinitions()
{
    auto* table = G4ParticleTable::GetParticleTable();
    fParticles.clear();
    for (const auto& name : fParticleNames) {
        auto* p = table->FindParticle(name);
        if (!p) {
            throw std::runtime_error("Unknown particle in /unveilers/beam/setParticles: " + std::string(name));
        }
        fParticles.push_back(p);
    }
}

G4ParticleDefinition* PrimaryGeneratorAction::SelectParticle() const
{
    if (fParticles.empty()) return G4ParticleTable::GetParticleTable()->FindParticle("pi+");

    G4double sum = 0.0;
    for (auto w : fWeights) sum += std::max(0.0, w);
    if (sum <= 0.) return fParticles.front();

    const G4double x = G4UniformRand() * sum;
    G4double acc = 0.0;
    for (std::size_t i = 0; i < fParticles.size(); ++i) {
        acc += std::max(0.0, fWeights[i]);
        if (x <= acc) return fParticles[i];
    }
    return fParticles.back();
}

void PrimaryGeneratorAction::GeneratePrimaries(G4Event* event)
{
    auto* particle = SelectParticle();
    fGun->SetParticleDefinition(particle);

    const G4double rho = fBeamRadius * std::sqrt(G4UniformRand());
    const G4double phi = CLHEP::twopi * G4UniformRand();
    const G4double x = rho * std::cos(phi);
    const G4double y = rho * std::sin(phi);
    fGun->SetParticlePosition(fPosition + G4ThreeVector(x, y, 0.));

    const G4double tx = G4RandGauss::shoot(0., fSigmaX);
    const G4double ty = G4RandGauss::shoot(0., fSigmaY);
    G4ThreeVector dir = fDirection + G4ThreeVector(tx, ty, 0.);
    if (dir.mag2() == 0.) dir = G4ThreeVector(0., 0., 1.);
    fGun->SetParticleMomentumDirection(dir.unit());

    G4double energy = fEnergy;
    if (fEnergySigma > 0.) {
        do {
            energy = G4RandGauss::shoot(fEnergy, fEnergySigma);
        } while (energy <= 0.);
    }
    fGun->SetParticleEnergy(energy);

    fGun->GeneratePrimaryVertex(event);
}

****** src/PrimaryGeneretorMessenger.cc ******

#include "PrimaryGeneratorMessenger.hh"
#include "PrimaryGeneratorAction.hh"

#include "G4ApplicationState.hh"
#include "G4UIcmdWith3Vector.hh"
#include "G4UIcmdWith3VectorAndUnit.hh"
#include "G4UIcmdWithADoubleAndUnit.hh"
#include "G4UIcmdWithAString.hh"
#include "G4UIdirectory.hh"

PrimaryGeneratorMessenger::PrimaryGeneratorMessenger(PrimaryGeneratorAction* gun)
: fGun(gun)
{
    fDir = new G4UIdirectory("/unveilers/beam/");
    fDir->SetGuidance("Primary beam control");

    fParticlesCmd = new G4UIcmdWithAString("/unveilers/beam/setParticles", this);
    fParticlesCmd->SetGuidance("Comma-separated particles, e.g. e-,e+,proton,pi+");
    fParticlesCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fWeightsCmd = new G4UIcmdWithAString("/unveilers/beam/setWeights", this);
    fWeightsCmd->SetGuidance("Comma-separated weights, same length as setParticles");
    fWeightsCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fEnergyCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/beam/setEnergy", this);
    fEnergyCmd->SetUnitCategory("Energy");
    fEnergyCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fEnergySigmaCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/beam/setEnergySigma", this);
    fEnergySigmaCmd->SetUnitCategory("Energy");
    fEnergySigmaCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
```

```
    fRadiusCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/beam/setBeamRadius", this);
    fRadiusCmd->SetUnitCategory("Length");
    fRadiusCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fSigmaXCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/beam/setSigmaX", this);
    fSigmaXCmd->SetUnitCategory("Angle");
    fSigmaXCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fSigmaYCmd = new G4UIcmdWithADoubleAndUnit("/unveilers/beam/setSigmaY", this);
    fSigmaYCmd->SetUnitCategory("Angle");
    fSigmaYCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fPositionCmd = new G4UIcmdWith3VectorAndUnit("/unveilers/beam/setPosition", this);
    fPositionCmd->SetUnitCategory("Length");
    fPositionCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

    fDirectionCmd = new G4UIcmdWith3Vector("/unveilers/beam/setDirection", this);
    fDirectionCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
}

PrimaryGeneratorMessenger::~PrimaryGeneratorMessenger()
{
    delete fParticlesCmd;
    delete fWeightsCmd;
    delete fEnergyCmd;
    delete fEnergySigmaCmd;
    delete fRadiusCmd;
    delete fSigmaXCmd;
    delete fSigmaYCmd;
    delete fPositionCmd;
    delete fDirectionCmd;
    delete fDir;
}

void PrimaryGeneratorMessenger::SetNewValue(G4UIcommand* cmd, G4String value)
{
    if (cmd == fParticlesCmd) {
        fGun->SetParticles(value);
    } else if (cmd == fWeightsCmd) {
        fGun->SetWeights(value);
    } else if (cmd == fEnergyCmd) {
        fGun->SetEnergy(fEnergyCmd->GetNewDoubleValue(value));
    } else if (cmd == fEnergySigmaCmd) {
        fGun->SetEnergySigma(fEnergySigmaCmd->GetNewDoubleValue(value));
    } else if (cmd == fRadiusCmd) {
        fGun->SetBeamRadius(fRadiusCmd->GetNewDoubleValue(value));
    } else if (cmd == fSigmaXCmd) {
        fGun->SetSigmaX(fSigmaXCmd->GetNewDoubleValue(value));
    } else if (cmd == fSigmaYCmd) {
        fGun->SetSigmaY(fSigmaYCmd->GetNewDoubleValue(value));
    } else if (cmd == fPositionCmd) {
        fGun->SetPosition(fPositionCmd->GetNew3VectorValue(value));
    } else if (cmd == fDirectionCmd) {
        fGun->SetDirection(fDirectionCmd->GetNew3VectorValue(value));
    }
}

****** src/RunAction.cc ******

#include "RunAction.hh"
#include "DetectorConstruction.hh"

#include "G4AnalysisManager.hh"
#include "G4Run.hh"

#include <algorithm>

RunAction::RunAction(DetectorConstruction* det)
: fDet(det)
{
    auto* man = G4AnalysisManager::Instance();
    man->SetVerboseLevel(1);
    man->SetDefaultFileType("root");
    man->SetNtupleMerging(false);

    const auto nLayers = std::max(1, fDet->GetNumberOfFilms());

    man->CreateH1("hLayer", "Film layer index;layer;entries", nLayers, -0.5, nLayers - 0.5);
    man->CreateH1("hEdepStep", "Deposited energy per hit in films;Edep [MeV];entries", 400, 0., 40.);
    man->CreateH2("hLongitudinalStep", "Longitudinal profile from hit steps;layer;Edep [MeV]", nLayers, -0.5, nLayers - 0.5, 300, 0., 30.);
    man->CreateH2("hXY", "Transverse map in films;x [mm];y [mm]", 300, -150., 150., 300, -150., 150.);
    man->CreateH2("hRLayer", "Radial spread vs depth;layer;r [mm]", nLayers, -0.5, nLayers - 0.5, 250, 0., 150.);
    man->CreateH1("hEventFilmEdep", "Total Edep per event in all films;Edep [MeV];events", 400, 0., 400.);
    man->CreateH1("hLeakEkin", "Escaping kinetic energy entering leak plane;E_{kin} [MeV];events", 400, 0., 5000.);
    man->CreateH2("hLongitudinalEvent", "Per-event longitudinal sums;layer;Edep [MeV]", nLayers, -0.5, nLayers - 0.5, 400, 0., 400.);
    man->CreateH2("hLeakXY", "Leakage transverse map;x [mm];y [mm]", 300, -150., 150., 300, -150., 150.);

    man->CreateNtuple("filmHits", "Film hit-level information");
    man->CreateNtupleIColumn("eventID");
    man->CreateNtupleIColumn("layer");
    man->CreateNtupleDColumn("x_mm");
    man->CreateNtupleDColumn("y_mm");
    man->CreateNtupleDColumn("z_mm");
    man->CreateNtupleDColumn("r_mm");
    man->CreateNtupleDColumn("edep_MeV");
```

```
    man->CreateNtupleIColumn("pdg");
    man->CreateNtupleIColumn("trackID");
    man->CreateNtupleIColumn("parentID");
    man->FinishNtuple(0);

    man->CreateNtuple("eventSummary", "Per-event summary");
    man->CreateNtupleIColumn("eventID");
    man->CreateNtupleIColumn("primaryPDG");
    man->CreateNtupleDColumn("primaryE_MeV");
    man->CreateNtupleDColumn("totalFilmEdep_MeV");
    man->CreateNtupleDColumn("leakEkin_MeV");
    man->FinishNtuple(1);

    man->CreateNtuple("eventLayers", "Per-event summed energy in each film layer");
    man->CreateNtupleIColumn("eventID");
    man->CreateNtupleIColumn("primaryPDG");
    man->CreateNtupleIColumn("layer");
    man->CreateNtupleDColumn("z_mm");
    man->CreateNtupleDColumn("layerEdep_MeV");
    man->CreateNtupleDColumn("leakEkin_MeV");
    man->FinishNtuple(2);

    man->CreateNtuple("leakHits", "Leakage plane information");
    man->CreateNtupleIColumn("eventID");
    man->CreateNtupleIColumn("pdg");
    man->CreateNtupleDColumn("x_mm");
    man->CreateNtupleDColumn("y_mm");
    man->CreateNtupleDColumn("z_mm");
    man->CreateNtupleDColumn("r_mm");
    man->CreateNtupleDColumn("ekin_MeV");
    man->FinishNtuple(3);
}

RunAction::~RunAction()
{
    delete G4AnalysisManager::Instance();
}

void RunAction::BeginOfRunAction(const G4Run*)
{
    G4AnalysisManager::Instance()->OpenFile("TheUnveilers");
}

void RunAction::EndOfRunAction(const G4Run*)
{
    auto* man = G4AnalysisManager::Instance();
    man->Write();
    man->CloseFile();
}

****** src/SteppingAction.cc ******

#include "SteppingAction.hh"
#include "EventAction.hh"

#include "G4AnalysisManager.hh"
#include "G4Event.hh"
#include "G4RunManager.hh"
#include "G4Step.hh"
#include "G4SystemOfUnits.hh"
#include "G4Track.hh"
#include "G4TouchableHandle.hh"
#include "G4VPhysicalVolume.hh"

#include <cmath>

SteppingAction::SteppingAction(EventAction* eventAction)
: fEventAction(eventAction)
{}

void SteppingAction::UserSteppingAction(const G4Step* step)
{
    auto* preVol = step->GetPreStepPoint()->GetPhysicalVolume();
    auto* postVol = step->GetPostStepPoint()->GetPhysicalVolume();
    if (!preVol) return;

    auto* man = G4AnalysisManager::Instance();
    auto* evt = G4RunManager::GetRunManager()->GetCurrentEvent();
    const G4int eventID = evt ? evt->GetEventID() : -1;

    const G4double edep = step->GetTotalEnergyDeposit();
    const auto prePos = step->GetPreStepPoint()->GetPosition();
    const G4double r = std::sqrt(prePos.x() * prePos.x() + prePos.y() * prePos.y());
    const auto* track = step->GetTrack();
    const G4int pdg = track->GetDefinition()->GetPDGEncoding();
    const G4int trackID = track->GetTrackID();
    const G4int parentID = track->GetParentID();

    const auto preName = preVol->GetName();

    if (preName == "Film" && edep > 0.) {
        const G4int layer = step->GetPreStepPoint()->GetTouchable()->GetCopyNumber();

        man->FillH1(0, layer);
        man->FillH1(1, edep / MeV);
```

```
        man->FillH2(0, layer, edep / MeV);
        man->FillH2(1, prePos.x() / mm, prePos.y() / mm);
        man->FillH2(2, layer, r / mm);

        man->FillNtupleIColumn(0, 0, eventID);
        man->FillNtupleIColumn(0, 1, layer);
        man->FillNtupleDColumn(0, 2, prePos.x() / mm);
        man->FillNtupleDColumn(0, 3, prePos.y() / mm);
        man->FillNtupleDColumn(0, 4, prePos.z() / mm);
        man->FillNtupleDColumn(0, 5, r / mm);
        man->FillNtupleDColumn(0, 6, edep / MeV);
        man->FillNtupleIColumn(0, 7, pdg);
        man->FillNtupleIColumn(0, 8, trackID);
        man->FillNtupleIColumn(0, 9, parentID);
        man->AddNtupleRow(0);

        if (fEventAction) {
            fEventAction->AddFilmEdep(layer, edep);
        }
    }

    const bool enteringLeakPlane = (postVol && postVol->GetName() == "LeakPlane" && preName != "LeakPlane");
    if (enteringLeakPlane) {
        const auto pos = step->GetPostStepPoint()->GetPosition();
        const G4double rr = std::sqrt(pos.x() * pos.x() + pos.y() * pos.y());
        const G4double ekin = step->GetPostStepPoint()->GetKineticEnergy();

        man->FillH2(4, pos.x() / mm, pos.y() / mm);
        man->FillNtupleIColumn(3, 0, eventID);
        man->FillNtupleIColumn(3, 1, pdg);
        man->FillNtupleDColumn(3, 2, pos.x() / mm);
        man->FillNtupleDColumn(3, 3, pos.y() / mm);
        man->FillNtupleDColumn(3, 4, pos.z() / mm);
        man->FillNtupleDColumn(3, 5, rr / mm);
        man->FillNtupleDColumn(3, 6, ekin / MeV);
        man->AddNtupleRow(3);

        if (fEventAction) {
            fEventAction->AddLeakEkin(ekin);
        }
    }
}


****** include/ActionInitialization.hh ******

#ifndef ActionInitialization_h
#define ActionInitialization_h

#include "G4VUserActionInitialization.hh"

class DetectorConstruction;

class ActionInitialization : public G4VUserActionInitialization
{
public:
    explicit ActionInitialization(DetectorConstruction* det);
    ~ActionInitialization() override = default;

    void Build() const override;

private:
    DetectorConstruction* fDet = nullptr;
};

#endif

****** include/DetectorConstruction.hh ******

#ifndef DetectorConstruction_h
#define DetectorConstruction_h

#include "G4VUserDetectorConstruction.hh"
#include "G4SystemOfUnits.hh"
#include "globals.hh"

#include <map>
#include <vector>

class DetectorMessenger;
class SimpleEMField;
class G4LogicalVolume;
class G4VPhysicalVolume;
class G4Material;
class G4FieldManager;
class G4EqMagElectricField;
class G4MagIntegratorStepper;
class G4ChordFinder;
class G4MagInt_Driver;

class DetectorConstruction : public G4VUserDetectorConstruction
{
public:
    DetectorConstruction();
    ~DetectorConstruction() override;
```

```
    G4VPhysicalVolume* Construct() override;
    void ConstructSDandField() override;

    void SetNumberOfLayers(G4int n);
    void SetAbsorberMaterial(const G4String& name);
    void SetFilmMaterial(const G4String& name);
    void SetLayerPattern(const G4String& csv);
    void SetAbsorberThickness(G4double value);
    void SetFilmThickness(G4double value);
    void SetGapThickness(G4double value);
    void SetStackXY(G4double value);

    void SetFieldX(G4double value);
    void SetFieldY(G4double value);
    void SetFieldZ(G4double value);
    void SetElectricFieldX(G4double value);
    void SetElectricFieldY(G4double value);
    void SetElectricFieldZ(G4double value);

    G4int GetNumberOfFilms() const { return fNLayers; }
    G4double GetPitch() const { return fPitch; }
    G4double GetStackStartZ() const { return fStackPosZ - 0.5 * fStackLength; }
    G4double GetLeakZ() const { return fLeakPosZ; }
    G4double GetFilmThickness() const { return fFilmThickness; }
    G4double GetAbsorberThickness() const { return fAbsThickness; }
    G4double GetGapThickness() const { return fGapThickness; }

    G4String ResolveMaterialName(const G4String& token) const;

private:
    void DefineMaterials();
    void RequestGeometryUpdate();
    void RefreshVisualization();
    void UpdateField();
    std::vector<G4String> ParseCSV(const G4String& text) const;
    G4Material* FindMaterialFromToken(const G4String& token) const;

private:
    DetectorMessenger* fMessenger = nullptr;

    G4Material* fWorldMat = nullptr;
    G4Material* fDefaultAbsMat = nullptr;
    G4Material* fFilmMat = nullptr;

    std::map<G4String, G4String> fMaterialAliases;
    std::vector<G4String> fLayerPatternTokens;
    std::vector<G4LogicalVolume*> fFilmLogicals;
    std::vector<G4LogicalVolume*> fAbsLogicals;
    G4LogicalVolume* fLeakLogical = nullptr;

    G4String fAbsMatName = "W";
    G4String fFilmMatName = "RCF_EBT3";

    G4int fNLayers = 24;

    G4double fWorldXY = 1.4 * m;
    G4double fWorldZ = 3.0 * m;
    G4double fStackXY = 30. * cm;
    G4double fAbsThickness = 3.5 * mm;
    G4double fFilmThickness = 0.28 * mm;
    G4double fGapThickness = 1.0 * mm;
    G4double fLeakThickness = 0.2 * mm;
    G4double fPitch = 0.0;
    G4double fStackLength = 0.0;
    G4double fStackPosZ = 20. * cm;
    G4double fLeakPosZ = 0.0;

    G4double fBx = 0.0;
    G4double fBy = 0.0;
    G4double fBz = 0.0;
    G4double fEx = 0.0;
    G4double fEy = 0.0;
    G4double fEz = 0.0;

    SimpleEMField* fEMField = nullptr;
    G4FieldManager* fFieldManager = nullptr;
    G4EqMagElectricField* fEquation = nullptr;
    G4MagIntegratorStepper* fStepper = nullptr;
    G4MagInt_Driver* fIntegrationDriver = nullptr;
    G4ChordFinder* fChordFinder = nullptr;
};

#endif

****** include/DetectorMessenger.hh ******

#ifndef DetectorMessenger_h
#define DetectorMessenger_h

#include "G4UImessenger.hh"

class DetectorConstruction;
class G4UIdirectory;
class G4UIcmdWithAString;
```

```
class G4UIcmdWithAnInteger;
class G4UIcmdWithADoubleAndUnit;
class G4UIcommand;

class DetectorMessenger : public G4UImessenger
{
public:
   explicit DetectorMessenger(DetectorConstruction* det);
   ~DetectorMessenger() override;

   void SetNewValue(G4UIcommand* cmd, G4String value) override;

private:
   DetectorConstruction* fDetector = nullptr;

   G4UIdirectory* fDetDir = nullptr;
   G4UIdirectory* fFieldDir = nullptr;

   G4UIcmdWithAnInteger* fLayersCmd = nullptr;
   G4UIcmdWithAString* fAbsMatCmd = nullptr;
   G4UIcmdWithAString* fFilmMatCmd = nullptr;
   G4UIcmdWithAString* fPatternCmd = nullptr;
   G4UIcmdWithADoubleAndUnit* fAbsThickCmd = nullptr;
   G4UIcmdWithADoubleAndUnit* fFilmThickCmd = nullptr;
   G4UIcmdWithADoubleAndUnit* fGapThickCmd = nullptr;
   G4UIcmdWithADoubleAndUnit* fStackXYCmd = nullptr;

   G4UIcmdWithADoubleAndUnit* fFieldXCmd = nullptr;
   G4UIcmdWithADoubleAndUnit* fFieldYCmd = nullptr;
   G4UIcmdWithADoubleAndUnit* fFieldZCmd = nullptr;
   G4UIcmdWithADoubleAndUnit* fElectricXCmd = nullptr;
   G4UIcmdWithADoubleAndUnit* fElectricYCmd = nullptr;
   G4UIcmdWithADoubleAndUnit* fElectricZCmd = nullptr;
};

#endif

****** include/EventAction.hh ******

#ifndef EventAction_h
#define EventAction_h

#include "G4UserEventAction.hh"
#include "globals.hh"
#include <vector>

class DetectorConstruction;
class G4Event;

class EventAction : public G4UserEventAction
{
public:
   explicit EventAction(DetectorConstruction* det);
   ~EventAction() override = default;

   void BeginOfEventAction(const G4Event*) override;
   void EndOfEventAction(const G4Event*) override;

   void AddFilmEdep(G4int layer, G4double e);
   void AddLeakEkin(G4double e);

private:
   DetectorConstruction* fDet = nullptr;
   std::vector<G4double> fLayerEdep;
   G4double fTotalFilmEdep = 0.0;
   G4double fLeakEkin = 0.0;
};

#endif

****** include/PrimaryGeneratorAction.hh ******

#ifndef PrimaryGeneratorAction_h
#define PrimaryGeneratorAction_h

#include "G4VUserPrimaryGeneratorAction.hh"
#include "G4ThreeVector.hh"
#include "G4SystemOfUnits.hh"
#include "globals.hh"

#include <vector>

class G4Event;
class G4ParticleGun;
class G4ParticleDefinition;
class PrimaryGeneratorMessenger;

class PrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
public:
   PrimaryGeneratorAction();
   ~PrimaryGeneratorAction() override;

   void GeneratePrimaries(G4Event*) override;
```

```
    void SetParticles(const G4String& csv);
    void SetWeights(const G4String& csv);
    void SetEnergy(G4double e) { if (e > 0.) fEnergy = e; }
    void SetEnergySigma(G4double s) { if (s >= 0.) fEnergySigma = s; }
    void SetBeamRadius(G4double r) { if (r >= 0.) fBeamRadius = r; }
    void SetSigmaX(G4double v) { if (v >= 0.) fSigmaX = v; }
    void SetSigmaY(G4double v) { if (v >= 0.) fSigmaY = v; }
    void SetPosition(const G4ThreeVector& p) { fPosition = p; }
    void SetDirection(const G4ThreeVector& d);

private:
    std::vector<G4String> ParseCSV(const G4String& text) const;
    void UpdateParticleDefinitions();
    G4ParticleDefinition* SelectParticle() const;

private:
    G4ParticleGun* fGun = nullptr;
    PrimaryGeneratorMessenger* fMessenger = nullptr;

    std::vector<G4String> fParticleNames = {"pi+"};
    std::vector<G4double> fWeights = {1.0};
    std::vector<G4ParticleDefinition*> fParticles;

    G4double fEnergy = 5.0 * GeV;
    G4double fEnergySigma = 0.0;
    G4double fBeamRadius = 1.0 * mm;
    G4double fSigmaX = 1.0 * mrad;
    G4double fSigmaY = 1.0 * mrad;
    G4ThreeVector fPosition = G4ThreeVector(0., 0., -40. * cm);
    G4ThreeVector fDirection = G4ThreeVector(0., 0., 1.);
};

#endif

****** include/PrimaryGeneratorMessenger.hh ******

#ifndef PrimaryGeneratorMessenger_h
#define PrimaryGeneratorMessenger_h

#include "G4UImessenger.hh"

class PrimaryGeneratorAction;
class G4UIdirectory;
class G4UIcmdWithAString;
class G4UIcmdWithADoubleAndUnit;
class G4UIcommand;
class G4UIcmdWith3VectorAndUnit;
class G4UIcmdWith3Vector;

class PrimaryGeneratorMessenger : public G4UImessenger
{
public:
    explicit PrimaryGeneratorMessenger(PrimaryGeneratorAction* gun);
    ~PrimaryGeneratorMessenger() override;

    void SetNewValue(G4UIcommand* cmd, G4String value) override;

private:
    PrimaryGeneratorAction* fGun = nullptr;
    G4UIdirectory* fDir = nullptr;

    G4UIcmdWithAString* fParticlesCmd = nullptr;
    G4UIcmdWithAString* fWeightsCmd = nullptr;
    G4UIcmdWithADoubleAndUnit* fEnergyCmd = nullptr;
    G4UIcmdWithADoubleAndUnit* fEnergySigmaCmd = nullptr;
    G4UIcmdWithADoubleAndUnit* fRadiusCmd = nullptr;
    G4UIcmdWithADoubleAndUnit* fSigmaXCmd = nullptr;
    G4UIcmdWithADoubleAndUnit* fSigmaYCmd = nullptr;
    G4UIcmdWith3VectorAndUnit* fPositionCmd = nullptr;
    G4UIcmdWith3Vector* fDirectionCmd = nullptr;
};

#endif

****** include/RunAction.hh ******

#ifndef RunAction_h
#define RunAction_h

#include "G4UserRunAction.hh"

class DetectorConstruction;
class G4Run;

class RunAction : public G4UserRunAction
{
public:
    explicit RunAction(DetectorConstruction* det);
    ~RunAction() override;

    void BeginOfRunAction(const G4Run*) override;
    void EndOfRunAction(const G4Run*) override;

private:
    DetectorConstruction* fDet = nullptr;
```

```
};

#endif

****** include/SimpleEMField.hh ******

#ifndef SimpleEMField_h
#define SimpleEMField_h

#include "G4ElectroMagneticField.hh"
#include "G4ThreeVector.hh"
#include "globals.hh"

class SimpleEMField : public G4ElectroMagneticField
{
public:
   SimpleEMField() = default;
   ~SimpleEMField() override = default;

   void SetFieldValue(const G4ThreeVector& magneticField, const G4ThreeVector& electricField)
   {
      fMagneticField = magneticField;
      fElectricField = electricField;
   }

   void GetFieldValue(const G4double[4], G4double* fieldBandE) const override
   {
      fieldBandE[0] = fMagneticField.x();
      fieldBandE[1] = fMagneticField.y();
      fieldBandE[2] = fMagneticField.z();
      fieldBandE[3] = fElectricField.x();
      fieldBandE[4] = fElectricField.y();
      fieldBandE[5] = fElectricField.z();
   }

   G4bool DoesFieldChangeEnergy() const override
   {
      return (fElectricField.mag2() > 0.);
   }

   G4bool HasAnyField() const
   {
      return (fMagneticField.mag2() > 0. || fElectricField.mag2() > 0.);
   }

private:
   G4ThreeVector fMagneticField;
   G4ThreeVector fElectricField;
};

#endif

****** include/SteppingAction.hh ******

#ifndef SteppingAction_h
#define SteppingAction_h

#include "G4UserSteppingAction.hh"

class EventAction;
class G4Step;

class SteppingAction : public G4UserSteppingAction
{
public:
   explicit SteppingAction(EventAction* eventAction);
   ~SteppingAction() override = default;

   void UserSteppingAction(const G4Step*) override;

private:
   EventAction* fEventAction = nullptr;
};

#endif

****** macros/vis.mac ******

/control/verbose 1
/run/verbose 1
/tracking/verbose 0

/unveilers/det/setLayers 40
/unveilers/det/setAbsThickness 1 mm
/unveilers/det/setFilmThickness 0.28 mm
/unveilers/det/setGapThickness 0.0 mm
/unveilers/det/setAbsorberMaterial W
/unveilers/det/setFilmMaterial RCF_EBT3
/unveilers/det/setStackXY 15 cm

/unveilers/field/setBx 0 tesla
/unveilers/field/setBy 0 tesla
/unveilers/field/setBz 0 tesla
/unveilers/field/setEx 0 megavolt/m
/unveilers/field/setEy 0 megavolt/m
```

```
/unveilers/field/setEz 0 megavolt/m

/unveilers/beam/setParticles pi+, proton
/unveilers/beam/setWeights 0.5, 0.5
/unveilers/beam/setEnergy 6 GeV
/unveilers/beam/setEnergySigma 0,15 GeV
/unveilers/beam/setBeamRadius 10 mm
/unveilers/beam/setSigmaX 1 mrad
/unveilers/beam/setSigmaY 1 mrad
/unveilers/beam/setPosition 0 0 -40 cm
/unveilers/beam/setDirection 0 0 1

/run/initialize

/vis/open OGL 1400x950-0+0
/vis/viewer/set/style surface
/vis/drawVolume
/vis/viewer/set/autoRefresh true
/vis/scene/add/axes 0 0 0 10 cm
/vis/scene/add/text2D 0.02 0.85 20 ! ! The Unveilers of the Invisible
/vis/scene/add/trajectories smooth
/vis/modeling/trajectories/create/drawByParticleID
/vis/modeling/trajectories/drawByParticleID-0/default/setDrawStepPts false
/vis/modeling/trajectories/drawByParticleID-0/set gamma yellow
/vis/modeling/trajectories/drawByParticleID-0/set e- cyan
/vis/modeling/trajectories/drawByParticleID-0/set e+ magenta
/vis/modeling/trajectories/drawByParticleID-0/set proton red
/vis/modeling/trajectories/drawByParticleID-0/set pi+ green
/vis/modeling/trajectories/drawByParticleID-0/set pi- blue
/vis/modeling/trajectories/drawByParticleID-0/set mu+ orange
/vis/modeling/trajectories/drawByParticleID-0/set mu- brown
/vis/viewer/set/viewpointThetaPhi 80 15 deg
/vis/viewer/zoom 1.5
/vis/scene/endOfEventAction accumulate
/tracking/storeTrajectory 1

/run/beamOn 15000
```

****** analysis/plot_shower.py ******

```
import os
import sys
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
import uproot

ROOT_FILE = sys.argv[1] if len(sys.argv) > 1 else "TheUnveilers.root"
OUTDIR = sys.argv[2] if len(sys.argv) > 2 else "plots"
os.makedirs(OUTDIR, exist_ok=True)

plt.rcParams.update({
    "figure.figsize": (8, 5.5),
    "font.size": 12,
    "axes.titlesize": 15,
    "axes.labelsize": 12,
    "legend.fontsize": 11,
    "xtick.direction": "in",
    "ytick.direction": "in",
    "xtick.top": True,
    "ytick.right": True,
    "axes.linewidth": 1.0,
    "grid.alpha": 0.25,
})

PDG_LABELS = {
    11: "e-",
    -11: "e+",
    13: "mu-",
    -13: "mu+",
    22: "gamma",
    211: "pi+",
    -211: "pi-",
    2212: "proton",
}


def pdg_name(pdg):
    return PDG_LABELS.get(int(pdg), str(int(pdg)))


with uproot.open(ROOT_FILE) as f:
    film_tree = f["filmHits"]
    summary_tree = f["eventSummary"]
    layers_tree = f["eventLayers"]
    leak_tree = f["leakHits"]

    film = {k: film_tree[k].array(library="np") for k in film_tree.keys()}
    summary = {k: summary_tree[k].array(library="np") for k in summary_tree.keys()}
    layers = {k: layers_tree[k].array(library="np") for k in layers_tree.keys()}
    leak = {k: leak_tree[k].array(library="np") for k in leak_tree.keys()}
```

```
all_layers = np.unique(layers["layer"]).astype(int)
if len(all_layers) == 0:
    raise RuntimeError("No eventLayers entries found in ROOT file.")

n_layers = int(all_layers.max()) + 1

z_by_layer = np.full(n_layers, np.nan)
for layer_idx in range(n_layers):
    mask = layers["layer"] == layer_idx
    if np.any(mask):
        z_by_layer[layer_idx] = np.mean(layers["z_mm"][mask])

longitudinal = np.zeros(n_layers)
for layer_idx in range(n_layers):
    mask = layers["layer"] == layer_idx
    longitudinal[layer_idx] = layers["layerEdep_MeV"][mask].sum()

imax = int(np.argmax(longitudinal))
zmax = z_by_layer[imax]

# ---------------------------------------------------------------------
# Longitudinal shower profile
# ---------------------------------------------------------------------
plt.figure()
plt.plot(np.arange(n_layers), longitudinal, marker="o")
plt.axvline(imax, linestyle="--")
plt.xlabel("Film layer")
plt.ylabel("Summed deposited energy [MeV]")
plt.title(f"Longitudinal shower profile (max at layer {imax}, z = {zmax:.1f} mm)")
plt.grid(True, alpha=0.3)
plt.savefig(os.path.join(OUTDIR, "longitudinal_profile.png"), dpi=180, bbox_inches="tight")
plt.close()

plt.figure()
plt.plot(z_by_layer, longitudinal, marker="o")
plt.axvline(zmax, linestyle="--")
plt.xlabel("Depth z [mm]")
plt.ylabel("Summed deposited energy [MeV]")
plt.title("Longitudinal profile vs depth")
plt.grid(True, alpha=0.3)
plt.savefig(os.path.join(OUTDIR, "longitudinal_vs_depth.png"), dpi=180, bbox_inches="tight")
plt.close()

# ---------------------------------------------------------------------
# Global XY map over all films
# ---------------------------------------------------------------------
x_all = film["x_mm"]
y_all = film["y_mm"]
r_all = np.sqrt(x_all**2 + y_all**2)

xy_range = [[np.min(x_all), np.max(x_all)], [np.min(y_all), np.max(y_all)]]

# 1) Carte standard linéaire
plt.figure()
h = plt.hist2d(x_all, y_all, bins=180, range=xy_range)
plt.xlabel("x [mm]")
plt.ylabel("y [mm]")
plt.title("Film hit map (all films)")
plt.colorbar(label="Counts")
plt.savefig(os.path.join(OUTDIR, "film_xy_map.png"), dpi=180, bbox_inches="tight")
plt.close()

# 2) Carte logarithmique : bien meilleure visibilité de la gerbe périphérique
plt.figure()
counts, xedges, yedges, im = plt.hist2d(
    x_all,
    y_all,
    bins=180,
    range=xy_range,
    norm=LogNorm(vmin=1)
)
plt.xlabel("x [mm]")
plt.ylabel("y [mm]")
plt.title("Film hit map (all films, log scale)")
plt.colorbar(label="Counts")
plt.savefig(os.path.join(OUTDIR, "film_xy_map_log.png"), dpi=180, bbox_inches="tight")
plt.close()

# 3) Carte hors faisceau : on masque le disque central r < 10 mm
beam_radius_mm = 10.0
mask_outside_beam = r_all >= beam_radius_mm

plt.figure()
plt.hist2d(
    x_all[mask_outside_beam],
    y_all[mask_outside_beam],
    bins=180,
    range=xy_range,
    norm=LogNorm(vmin=1)
)
circle = plt.Circle((0, 0), beam_radius_mm, color="white", fill=False, linestyle="--", linewidth=1.5)
plt.gca().add_patch(circle)
plt.xlabel("x [mm]")
plt.ylabel("y [mm]")
```

```
plt.title("Film hit map outside primary beam (r ≥ 10 mm)")
plt.colorbar(label="Counts")
plt.savefig(os.path.join(OUTDIR, "film_xy_outside_beam_log.png"), dpi=180, bbox_inches="tight")
plt.close()

# ---------------------------------------------------------------------
# XY maps for each individual film
# ---------------------------------------------------------------------
film_layers_dir = os.path.join(OUTDIR, "film_layers")
os.makedirs(film_layers_dir, exist_ok=True)

beam_radius_mm = 10.0

for layer_idx in range(n_layers):
    mask = film["layer"] == layer_idx
    if not np.any(mask):
        continue

    zc = z_by_layer[layer_idx]
    x = film["x_mm"][mask]
    y = film["y_mm"][mask]
    r = np.sqrt(x**2 + y**2)

    xy_range_layer = [[-75, 75], [-75, 75]]

    # 1) Carte standard
    plt.figure()
    plt.hist2d(x, y, bins=180, range=xy_range_layer)
    plt.xlabel("x [mm]")
    plt.ylabel("y [mm]")
    plt.title(f"Film hit map - layer {layer_idx} (z = {zc:.1f} mm)")
    plt.colorbar(label="Counts")
    plt.savefig(
        os.path.join(film_layers_dir, f"film_xy_layer_{layer_idx:03d}.png"),
        dpi=180,
        bbox_inches="tight"
    )
    plt.close()

    # 2) Carte logarithmique : meilleure visibilité de la gerbe périphérique
    plt.figure()
    plt.hist2d(
        x,
        y,
        bins=180,
        range=xy_range_layer,
        norm=LogNorm(vmin=1)
    )
    plt.xlabel("x [mm]")
    plt.ylabel("y [mm]")
    plt.title(f"Film hit map - layer {layer_idx} (z = {zc:.1f} mm, log scale)")
    plt.colorbar(label="Counts")
    plt.savefig(
        os.path.join(film_layers_dir, f"film_xy_layer_{layer_idx:03d}_log.png"),
        dpi=180,
        bbox_inches="tight"
    )
    plt.close()

    # 3) Carte hors faisceau : on masque le disque central r < 10 mm
    mask_outside_beam = r >= beam_radius_mm
    if np.any(mask_outside_beam):
        plt.figure()
        plt.hist2d(
            x[mask_outside_beam],
            y[mask_outside_beam],
            bins=180,
            range=xy_range_layer,
            norm=LogNorm(vmin=1)
        )
        circle = plt.Circle(
            (0, 0),
            beam_radius_mm,
            color="white",
            fill=False,
            linestyle="--",
            linewidth=1.5
        )
        plt.gca().add_patch(circle)
        plt.xlabel("x [mm]")
        plt.ylabel("y [mm]")
        plt.title(f"Shower around beam - layer {layer_idx} (z = {zc:.1f} mm)")
        plt.colorbar(label="Counts")
        plt.savefig(
            os.path.join(film_layers_dir, f"film_xy_layer_{layer_idx:03d}_outside_beam_log.png"),
            dpi=180,
            bbox_inches="tight"
        )
        plt.close()

# ---------------------------------------------------------------------
# Radial shower development
# ---------------------------------------------------------------------
mean_r = np.full(n_layers, np.nan)
rms_r = np.full(n_layers, np.nan)
```

```
r90 = np.full(n_layers, np.nan)
r95 = np.full(n_layers, np.nan)

for i in range(n_layers):
    mask = film["layer"] == i
    ri = film["r_mm"][mask]
    wi = film["edep_MeV"][mask]

    if len(ri) == 0:
        continue

    w = np.maximum(wi, 1e-12)

    mean_r[i] = np.average(ri, weights=w)
    rms_r[i] = np.sqrt(np.average((ri - mean_r[i])**2, weights=w))

    order = np.argsort(ri)
    rs = ri[order]
    ws = w[order]
    c = np.cumsum(ws)

    idx90 = np.searchsorted(c, 0.90 * c[-1], side="left")
    idx95 = np.searchsorted(c, 0.95 * c[-1], side="left")

    idx90 = min(idx90, len(rs) - 1)
    idx95 = min(idx95, len(rs) - 1)

    r90[i] = rs[idx90]
    r95[i] = rs[idx95]

plt.figure()
plt.plot(np.arange(n_layers), mean_r, marker="o", label="Mean radius")
plt.plot(np.arange(n_layers), rms_r, marker="^", label="RMS radius")
plt.plot(np.arange(n_layers), r90, marker="s", label=r"$R_{90}$")
plt.plot(np.arange(n_layers), r95, marker="d", label=r"$R_{95}$")
plt.xlabel("Film layer")
plt.ylabel("Radius [mm]")
plt.title("Shower transverse extent by film layer")
plt.legend()
plt.grid(True, alpha=0.3)
plt.savefig(os.path.join(OUTDIR, "radial_development.png"), dpi=180, bbox_inches="tight")
plt.close()

plt.figure()
plt.plot(z_by_layer, mean_r, marker="o", label="Mean radius")
plt.plot(z_by_layer, rms_r, marker="^", label="RMS radius")
plt.plot(z_by_layer, r90, marker="s", label=r"$R_{90}$")
plt.plot(z_by_layer, r95, marker="d", label=r"$R_{95}$")
plt.xlabel("Depth z [mm]")
plt.ylabel("Radius [mm]")
plt.title("Shower transverse extent vs depth")
plt.legend()
plt.grid(True, alpha=0.3)
plt.savefig(os.path.join(OUTDIR, "shower_extent_vs_depth.png"), dpi=220, bbox_inches="tight")
plt.close()

plt.figure()
plt.plot(z_by_layer, r90, marker="s")
plt.xlabel("Depth z [mm]")
plt.ylabel(r"$R_{90}$ [mm]")
plt.title("Shower extent vs depth")
plt.grid(True, alpha=0.3)
plt.savefig(os.path.join(OUTDIR, "shower_extent_r90_vs_depth.png"), dpi=220, bbox_inches="tight")
plt.close()

# ----------------------------------------------------------------------
# Leakage maps and distributions
# ----------------------------------------------------------------------
if len(leak["x_mm"]) > 0:
    plt.figure()
    plt.hist2d(leak["x_mm"], leak["y_mm"], bins=180)
    plt.xlabel("x [mm]")
    plt.ylabel("y [mm]")
    plt.title("Leakage plane map")
    plt.colorbar(label="Counts")
    plt.savefig(os.path.join(OUTDIR, "leakage_xy_map.png"), dpi=180, bbox_inches="tight")
    plt.close()

plt.figure()
plt.hist(summary["leakEkin_MeV"], bins=80)
plt.xlabel("Escaping kinetic energy [MeV]")
plt.ylabel("Events")
plt.title("Leakage energy distribution")
plt.savefig(os.path.join(OUTDIR, "leakage_energy.png"), dpi=180, bbox_inches="tight")
plt.close()

# ----------------------------------------------------------------------
# By primary particle
# ----------------------------------------------------------------------
primary_pdgs = np.unique(summary["primaryPDG"])
if len(primary_pdgs) > 1:
    plt.figure()
    for pdg in primary_pdgs:
        mask = summary["primaryPDG"] == pdg
        plt.hist(summary["totalFilmEdep_MeV"][mask], bins=50, histtype="step", label=pdg_name(pdg))
```

```
    plt.xlabel("Total deposited energy in films [MeV]")
    plt.ylabel("Events")
    plt.title("Event energy deposit by primary particle")
    plt.legend()
    plt.savefig(os.path.join(OUTDIR, "event_edep_by_primary.png"), dpi=180, bbox_inches="tight")
    plt.close()

    plt.figure()
    for pdg in primary_pdgs:
        profile = np.zeros(n_layers)
        maskp = layers["primaryPDG"] == pdg
        for layer_idx in range(n_layers):
            mask = maskp & (layers["layer"] == layer_idx)
            profile[layer_idx] = layers["layerEdep_MeV"][mask].sum()
        plt.plot(np.arange(n_layers), profile, marker="o", label=pdg_name(pdg))
    plt.xlabel("Film layer")
    plt.ylabel("Summed deposited energy [MeV]")
    plt.title("Longitudinal profile by primary particle")
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.savefig(os.path.join(OUTDIR, "longitudinal_by_primary.png"), dpi=180, bbox_inches="tight")
    plt.close()

# ---------------------------------------------------------------------
# Text summary
# ---------------------------------------------------------------------
with open(os.path.join(OUTDIR, "summary.txt"), "w", encoding="utf-8") as fh:
    fh.write("TheUnveilers analysis summary\n")
    fh.write(f"ROOT file: {ROOT_FILE}\n")
    fh.write(f"Number of events: {len(summary['eventID'])}\n")
    fh.write(f"Number of layers: {n_layers}\n")
    fh.write(f"Shower maximum layer: {imax}\n")
    fh.write(f"Shower maximum depth z [mm]: {zmax:.3f}\n")
    fh.write(f"Total longitudinal deposited energy [MeV]: {longitudinal.sum():.3f}\n")

    valid_r90 = np.isfinite(r90)
    if np.any(valid_r90):
        imax_r90 = np.nanargmax(r90)
        fh.write(f"Maximum R90 [mm]: {r90[imax_r90]:.3f} at layer {imax_r90}, z = {z_by_layer[imax_r90]:.3f} mm\n")

    if len(primary_pdgs) > 0:
        fh.write("Primary particles present: " + ", ".join(pdg_name(p) for p in primary_pdgs) + "\n")

print(f"Plots written to: {OUTDIR}")
print(f"Shower max at layer {imax}, z = {zmax:.2f} mm")
```