LhARA Ionacoustic Meeting

10/03/2022

SmartPhantom (Geant4 Simulation)

- Produces ROOT file

Post Processing Scripts (in C++)

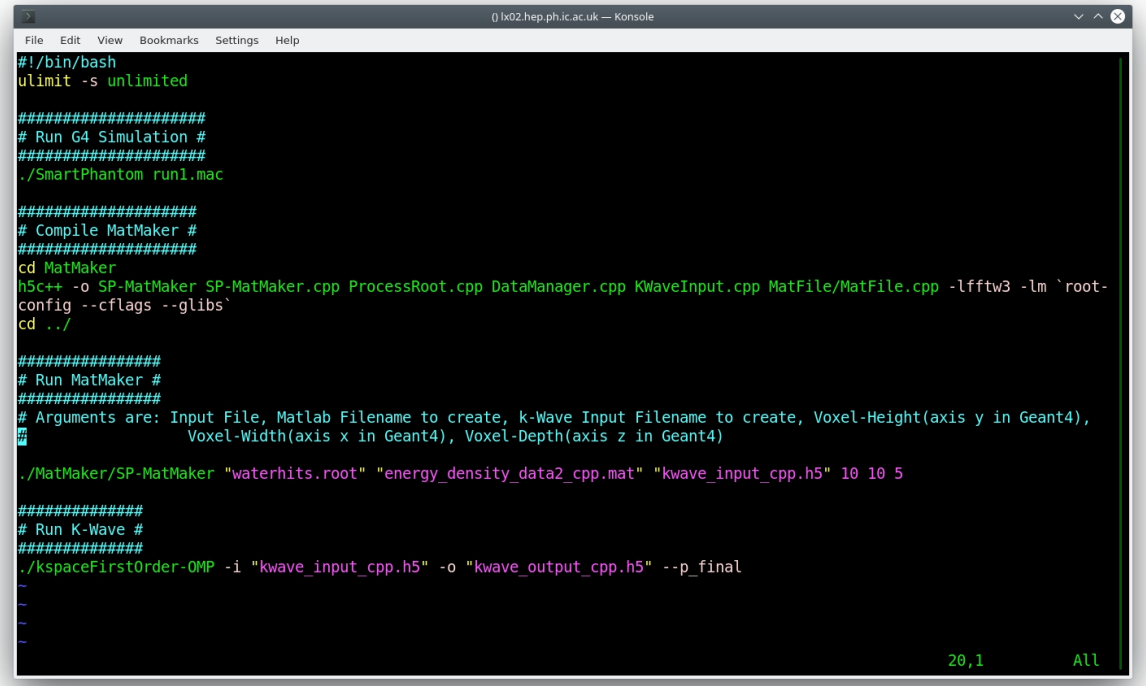
- Voxelize data
- Basic plot of energy in 3 axes
- Produce Matlab input file
- Produce HDF5 file for k-Wave(C++) input

k-Wave (C++ Binary)

- Produce HDF5 file

SP-MatMaker.sh

- Commands to run all those scripts are bundled into this bash script.
- Users can just comment or run the commands separately as necessary.
- Run with:
  - . SP-MatMaker.sh

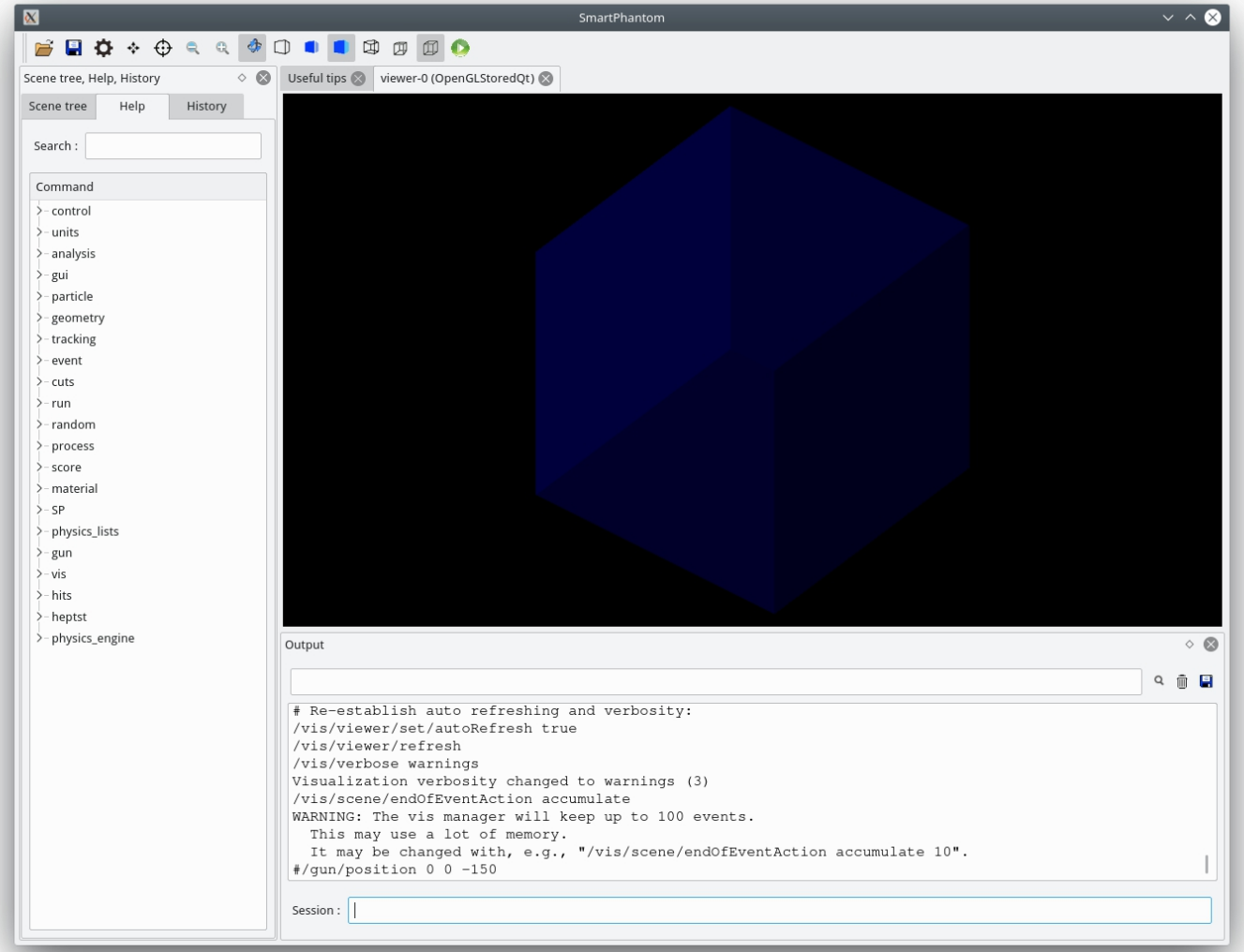


SmartPhantom (Geant4)

- Purpose:
  - Run physics interaction simulation through a water phantom
- Input:
  - Text file with particle coordinates for beam
  - (x [m], x’ [rad], y [m], y’ [rad], KE [MeV], t [ns])
  - Default: *test_file.dat*
- Output:
  - ROOT file containing information of phantom volume and detectors
  - Stores: energy deposited, time of hit, event number, position (x,y,z), steplength, delta time of hit, and name of particle causing the interaction
  - Default: *waterhits.root*
- Command to run separately:
  - *./SmartPhantom run1.mac*

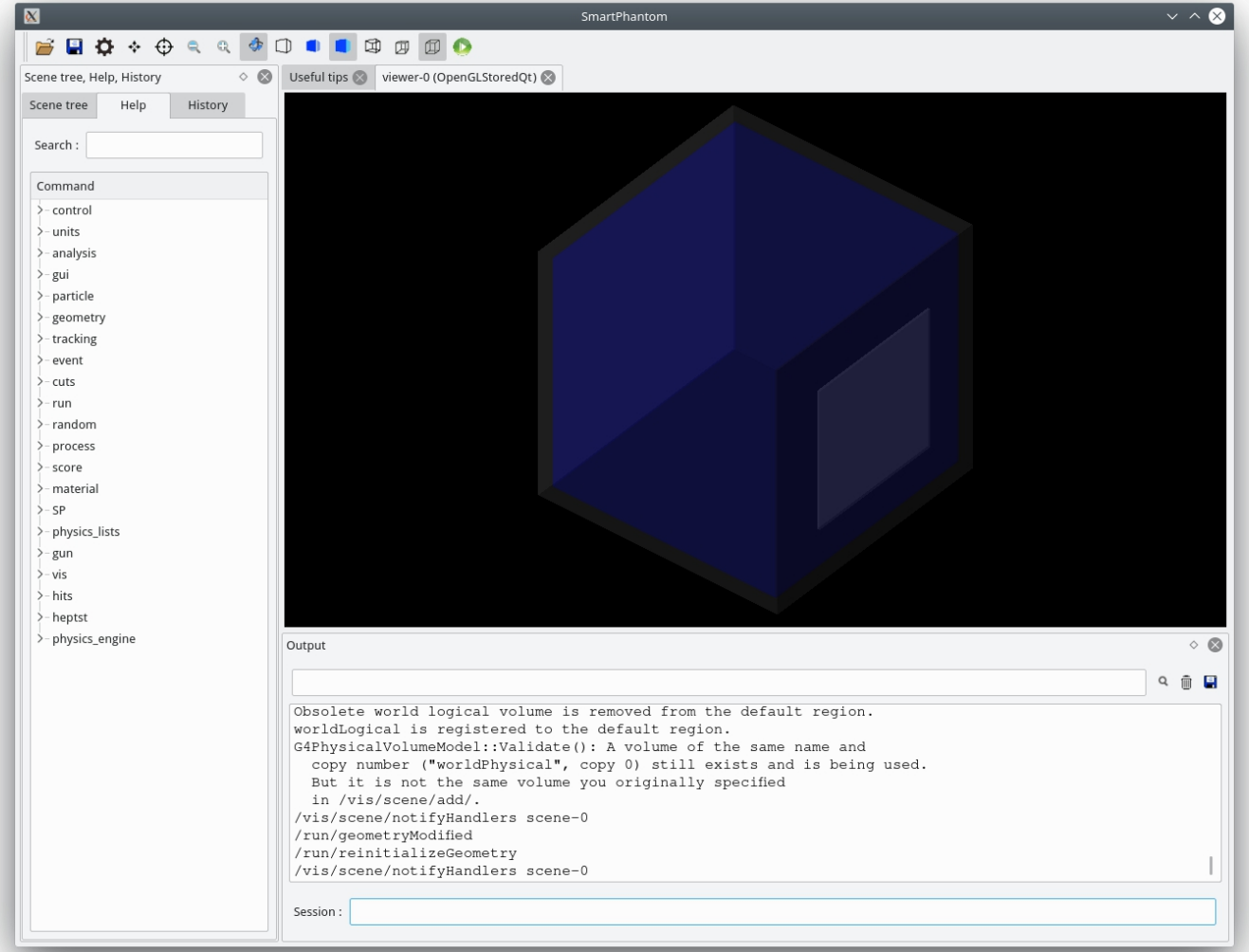
SmartPhantom (Geant4) -- Visualisation

- Can be run with visualisation
  - *./SmartPhantom*
- By default it is just a cube of water
- Various commands can be entered into the console
- Basic IO commands:
  - /SP/IO/inputFile “test_file.dat”
  - /SP/IO/outputFile “waterhits.root”



SmartPhantom (Geant4) -- Basic Commands

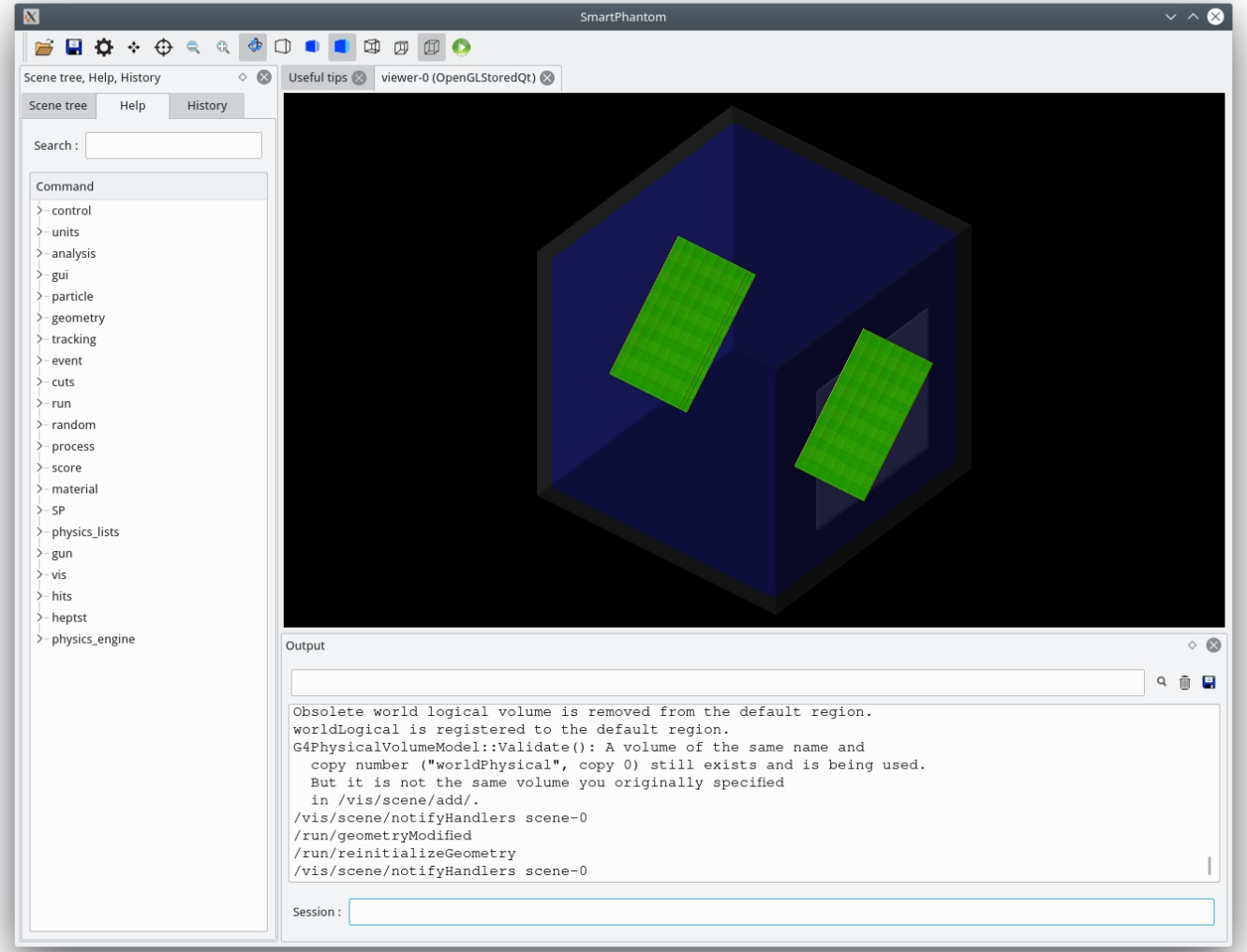
- To add phantom walls:
  - */SP/detector/enablePhantomWall true*



SmartPhantom (Geant4) -- Basic Commands

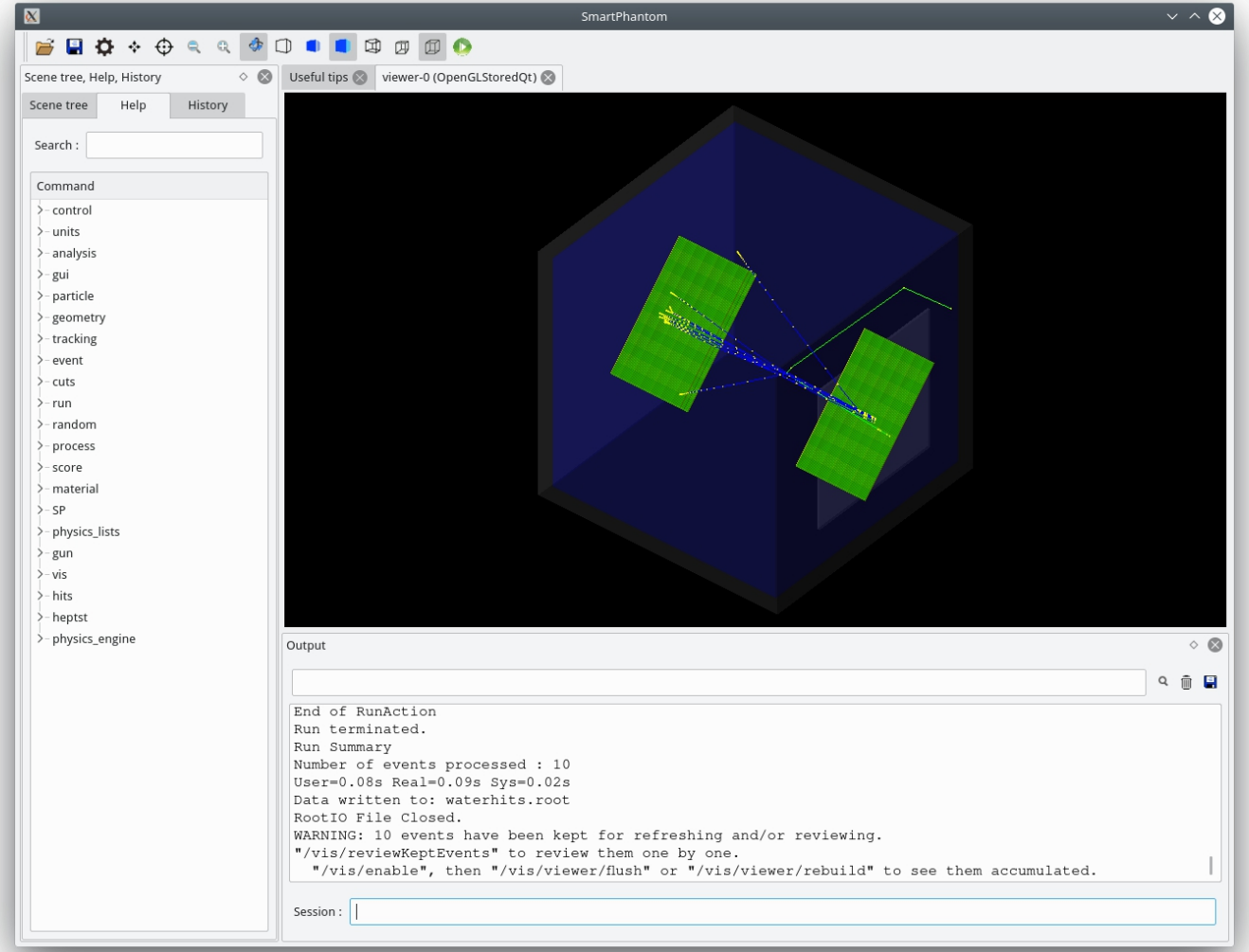
- To add detectors:
  - */SP/detector/enableScifi true*

- Adds four scintillating stations each with two planes (pictured in green)



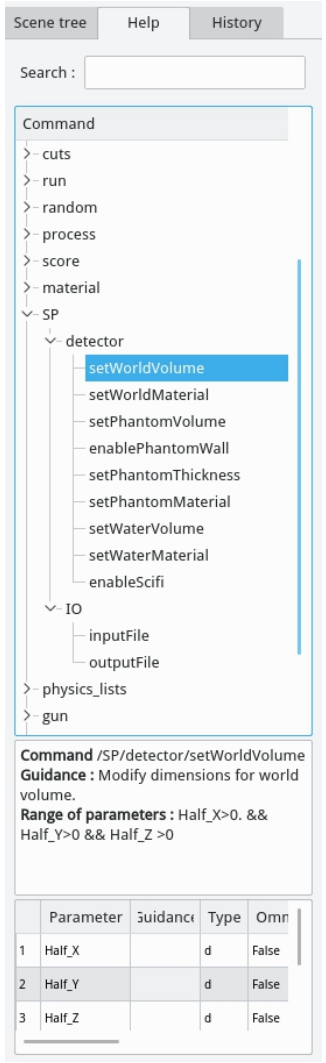
SmartPhantom (Geant4) -- Basic Commands

- To run a beam type:
  - */run/beamOn 10*
- Simulates 10 proton particles as defined by the first ten coordinates in the input file.
- Change the argument to the number you want



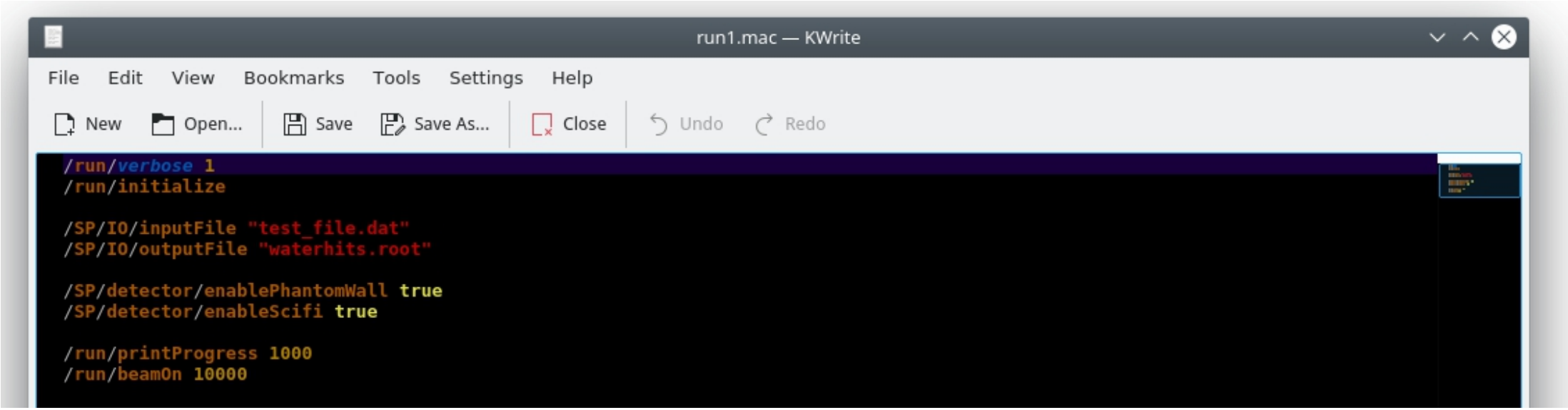
SmartPhantom (Geant4) -- Basic Commands

- Can use the left hand menu to see different commands.
- Commands under SP are ones I added
- There is a basic description of what the commands do and what arguments are expected.
- For more complex edits you will have to edit the source files and recompile (i.e. src/DetectorConstruction.cc)



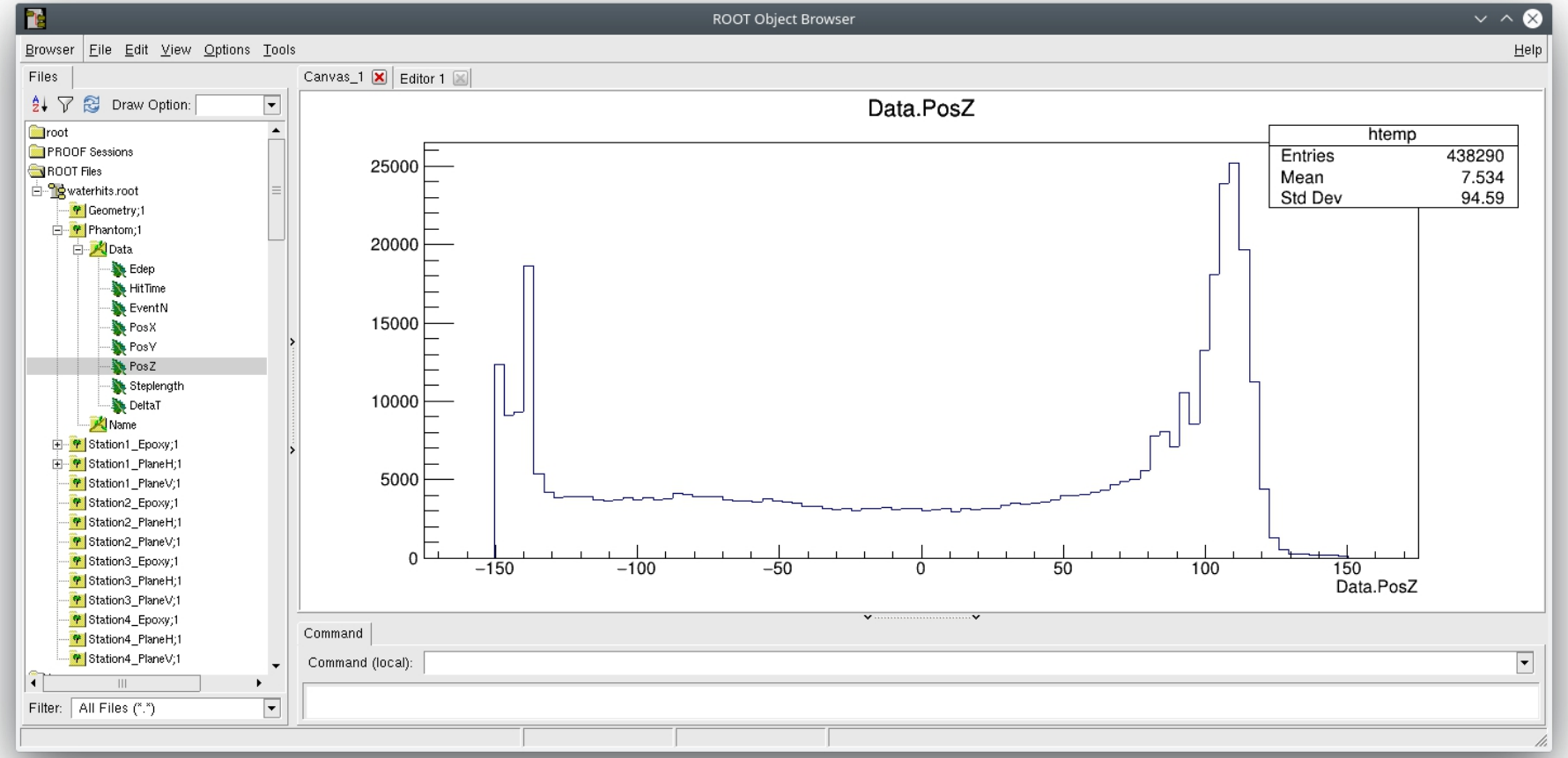
SmartPhantom (Geant4) -- Batch Mode

- For large number of particles easiest to run in batch mode
- Supply a macro file as the argument
  - *./SmartPhantom run1.mac*
- Supply the G4commands in the macro file



SmartPhantom (Geant4) -- ROOT File

- Produced ROOT file where different volumes are TTrees storing various simulation results shown as histograms

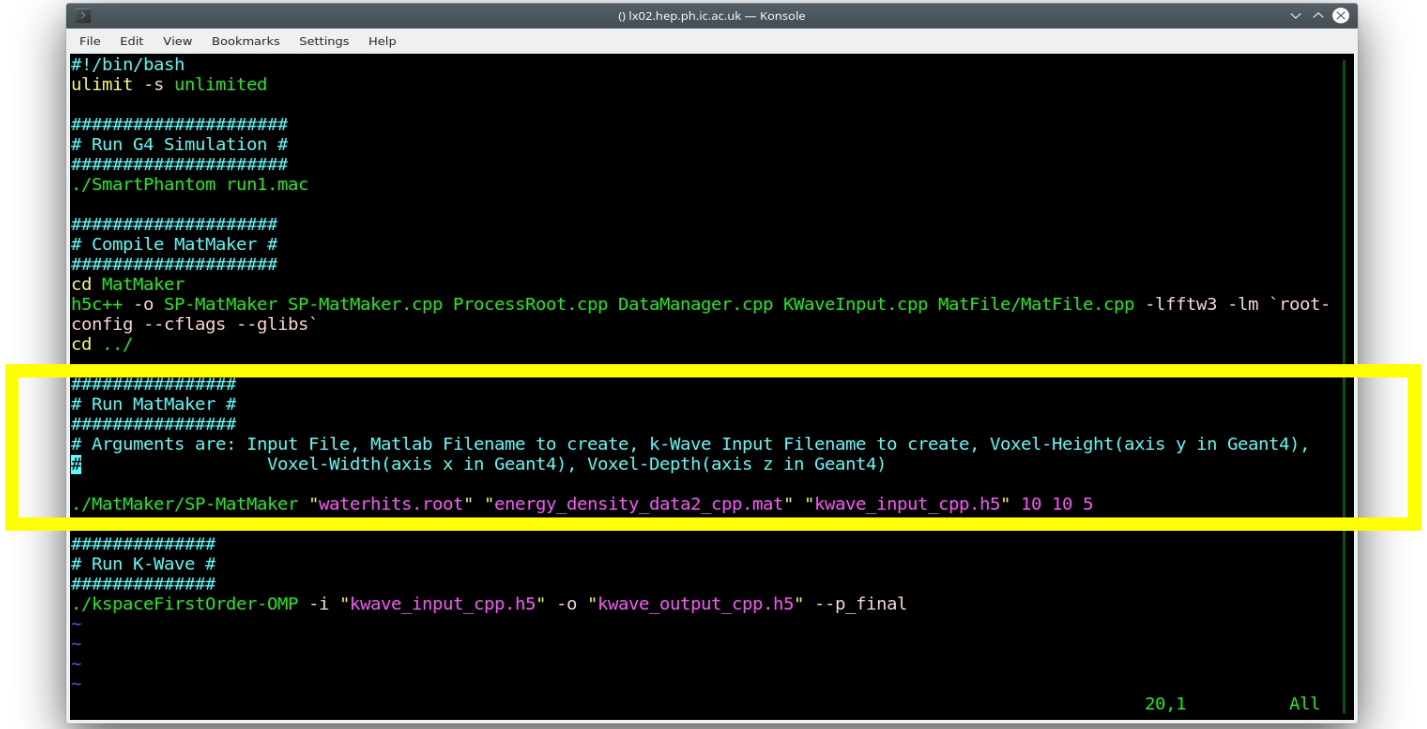


Post Processing Scripts

- Purpose:
  - Apply post processing to data (i.e. voxelize results)
  - Still very basic
- Input:
  - Reads the ROOT file from the Geant4 simulation + parameter settings
  - Default: *waterhits.root*
- Output:
  - Writes input file for k-Wave as a Matlab file or HDF5 file for the C++ binary
  - Default:
    - *energyplot-protons.png*
    - *energy_density_data2_cpp.mat*
    - *kwave_input_cpp.h5*
- Command to run separately (after compiling):
  - *./MatMaker/SP-MatMaker “waterhits.root” “energy_density_data2_cpp.mat” “kwave_input_cpp.h5” 10 10 5*

Post Processing Scripts -- Adjusting Parameters

- Easiest parameter to adjust is the voxel size
  - Done by editing *SP-MatMaker.sh* and adjusting last three parameters
  - Since phantom is 300 x 300 x 300 mm^3, divide by the given size to get number of voxels
  - Note transverse coordinates in Geant4 and k-Wave are swapped (x ↔ y)



Post Processing Scripts -- Adjusting Parameters

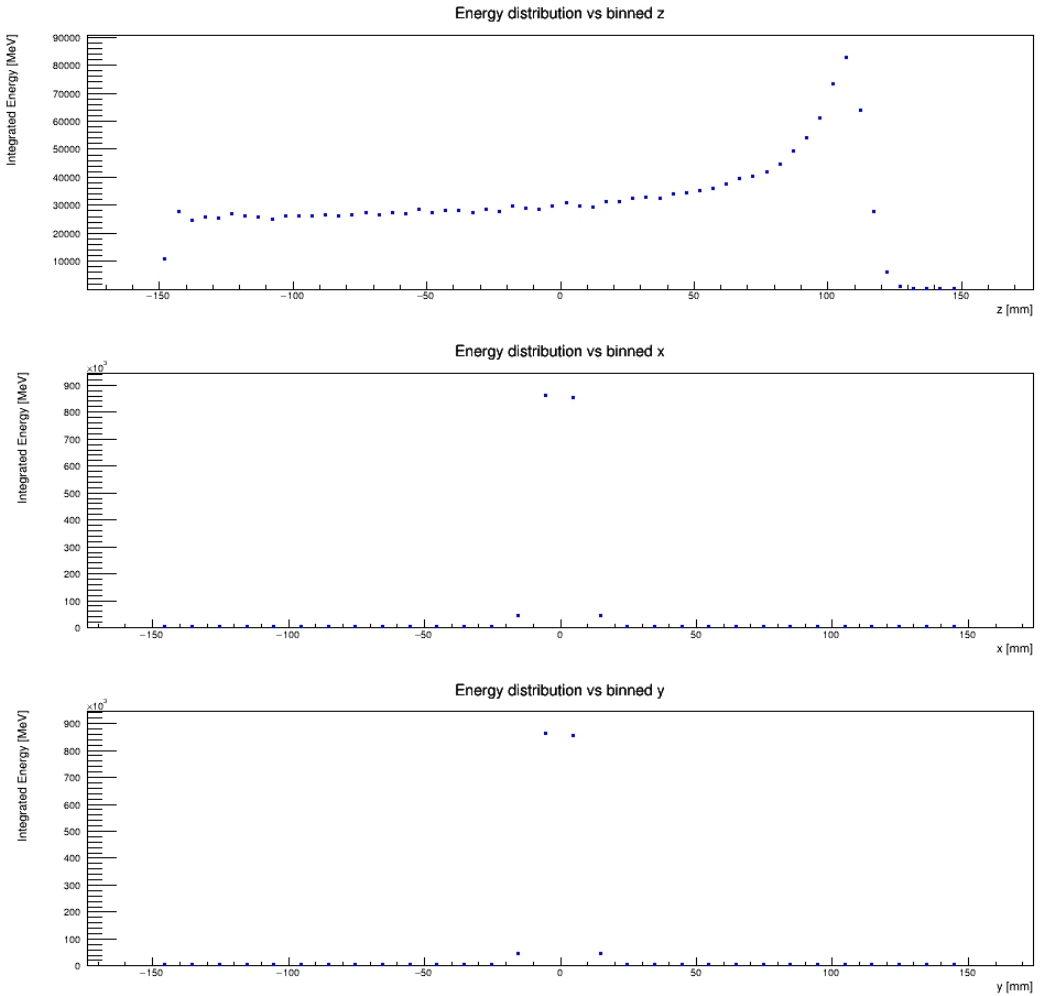
- Adjust the k-Wave input is done by editing: *MatMaker/SP-MatMaker.cpp*
  - Only basic parameters implemented (based on simple example Ben sent me)

```
() lx02.hep.ph.ic.ac.uk — Konsole
File  Edit  View  Bookmarks  Settings  Help
    ////////////////////////
    // Write K-Wave Input //
    ////////////////////////
    KWaveInput kwaveInput(kwaveName);

    // Define bin dimensions
    int binDim[3];
    binDim[0] = rootFile.GetBinHeight();
    binDim[1] = rootFile.GetBinWidth();
    binDim[2] = rootFile.GetBinDepth();

    kwaveInput.SetBinDim(binDim);

    double* energyData = rootFile.GetEnergySum();
    kwaveInput.SetEnergyData(energyData);

    // Apply smoothening by multiplying by window filter
    kwaveInput.SetGrueneisen(0.11);             // Grueneisen parameter for water
    kwaveInput.SetSmooth(true);                 // Whether or not to smoothen data
    kwaveInput.SetRestoreMagnitude(true);       // Whether to restore magnitude after smooth
    kwaveInput.SetRotWindow(true);              // Type of window function
                                                // (K-Wave uses rotational window by default)

    // Grid Size (should be consistent with Matlab convention)
    unsigned long Nx = binDim[0];               // Height
    unsigned long Ny = binDim[1];               // Width
    unsigned long Nz = binDim[2];               // Depth
    float dx = 1e-3;                            // grid point spacing in the x direction [m]
    float dy = 1e-3;                            // grid point spacing in the y direction [m]
    float dz = 1e-3;                            // grid point spacing in the z direction [m]

    kwaveInput.SetNx(Nx);
    kwaveInput.SetNy(Ny);
    kwaveInput.SetNz(Nz);
    kwaveInput.SetDx(dx);
    kwaveInput.SetDy(dy);
    kwaveInput.SetDz(dz);
```

```
    // Thickness of absorbing boundary (PML) around the simulation
    kwaveInput.SetPMLSizeX(10);                 // Not sure why despite pml_size = 1 in matlab
                                                // it still writes default (default = 10)
    kwaveInput.SetPMLSizeY(10);
    kwaveInput.SetPMLSizeZ(10);
    kwaveInput.SetPMLAlphaX(2);                 // Absorption within the perfectly matched
                                                // layer in Nepers per grid point (default = 2)
    kwaveInput.SetPMLAlphaY(2);
    kwaveInput.SetPMLAlphaZ(2);

    // Define the properties of the propagation medium
    float mediumSoundSpeed = 1500;              // [m/s]
    kwaveInput.SetMediumSoundSpeed(mediumSoundSpeed);
    kwaveInput.SetCRef(1500);                   // reference sound speed used within the
                                                // k-space operator (phase correction term)


    // Set the time limit and time steps
    float tEnd = (float)Nx*dx/mediumSoundSpeed;  // [s]
    kwaveInput.SetTEnd(tEnd);
    kwaveInput.SetCFL(0.3);

    // Sensor Mask
    // Following should give the same indices as with: sensor.mask(Nx/2, Ny/4:4:3*Ny/4, 3*Nz/4) = 1
    std::vector<unsigned long> sensorMask = kwaveInput.SensorMaskIndices(Nx, Nx/2, Nx/2, 0,
                                                                         Ny, Ny/4, 3*Ny/4, 4,
                                                                         Nz, 3*Nz/4, 3*Nz/4, 0);
    kwaveInput.SetSensorMask(sensorMask);

    kwaveInput.WriteFile();

    std::cout << "Finished Main" << std::endl;
}
```

Post Processing Scripts -- Output

- Running the script produces a plot to check binned energy against the three axes
  - *energyplot-protons.png*
- File for input to Matlab for k-Wave (to supply as initial pressure source)
  - *energy_density_data2_cpp.mat*
- File for input to run k-Wave (C++ binary)
  - *kwave_input_cpp.h5*

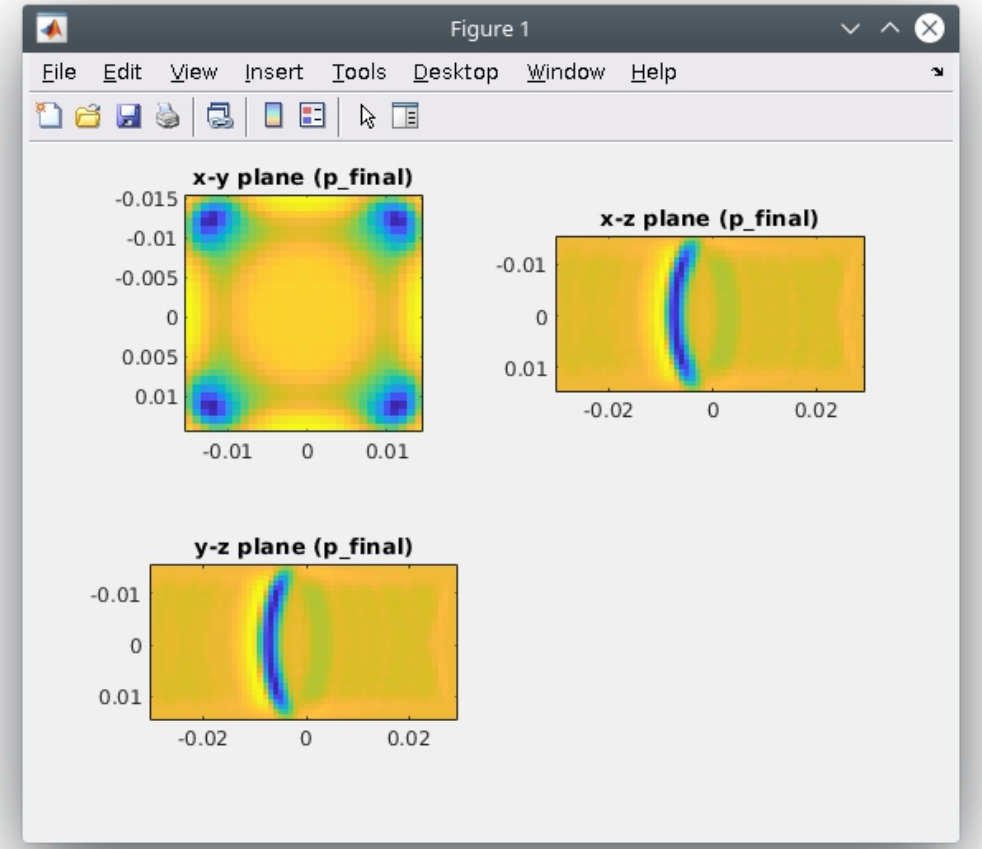


k-Wave (C++ binary) - From (www.k-wave.org)

- Purpose:
  - Model pressure wave
- Input:
  - Reads the HDF5 file from the Geant4 simulation + parameter settings (to be discussed)
  - Default: *kwave_input_cpp.h5*
- Output:
  - Runs k-Wave simulation where arguments specify variable saved
  - Default: *kwave_output_cpp.h5*
- Command to run separately:
  - *./kspaceFirstOrder-OMP -i “kwave_input_cpp.h5” -o “kwave_output_cpp.h5” --p_final*
  - Final argument allows user to select what parameters to save, in this case the final pressure field

k-Wave (C++ binary) - From (www.k-wave.org)

- Plotting p_final from the C++ output gives:



- Seems to match final pressure field when running in Matlab (colormap different)