Special Mesh Rendering

John Allison and Evgueni Tcherniaev

What is a “G4Mesh”?

- In many applications, especially medical applications, the material world is represented in part by a `G4PVParameterised` (including `G4VNestedParameterisation`).
- Each element has a different location (and size?) and may be programmed to have a different material and colour.
- On request
  `/vis/viewer/set/specialMeshRendering`
  the vis manager asks `G4PhysicalVolumeModel` to look out for candidates. If a volume’s descendents (up to 3 deep) is a `G4PVParameterised`, the volume is declared a “container” and wrapped in a special class, `G4Mesh`, and passed to the scene handler for ”special rendering”.
- `G4Mesh` anticipates 5 types (see inset) but only 3—rectangular(2 types) and tetrahedron—are programmed at present.
- The user may ask for specific meshes:
  `/vis/viewer/set/specialMeshVolumes <container-name>`
- There are two options:
  `/vis/viewer/set/specialMeshRenderingOption [dots|surfaces]`
- The above commands must come before
  `/vis/drawVolume`

```
class G4Mesh {
 public:
  enum MeshType {
   invalid
   , rectangle
   , nested3DRectangular
   , cylinder
   , sphere
   , tetrahedron
  };
```

Special mesh rendering

- When a mesh is asked for and is found, instead of descending the geometry tree, `G4PhysicalVolumeModel` passes the whole mesh to the vis driver.
  - The driver may (optionally) implement AddCompound(const G4Mesh&)
  - Otherwise the base class default is invoked, drawing just the “container”.
- The following drivers invoke “standard special mesh rendering”:
  - OpenGL, ToolsSG, Qt3D and OpenInventor

```
void G4OpenGLSceneHandler::AddCompound(const G4Mesh& mesh) {
  StandardSpecialMeshRendering(mesh);
}
```

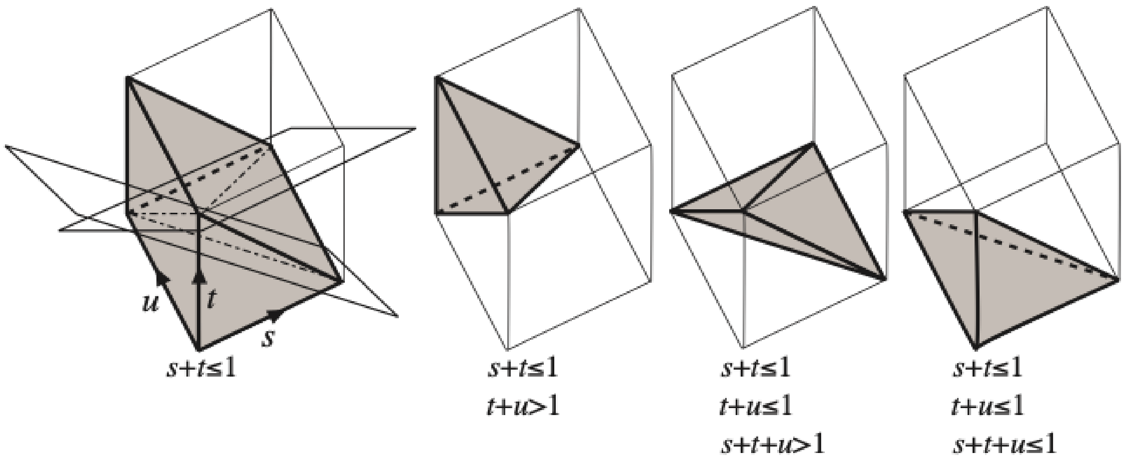
Standard special mesh rendering

- 4 programmed possibilities:
  - `Draw[3DRect|Tet]MeshAs[Dots|Surfaces]`
- Each involves a descent into the geometry sub-tree of the mesh, picking up the coordinates of the individual elements of the mesh, with their material and colour.
  - For "dots", each element gets a random point within it and a `std::multimap` is filled by material (with its colour).
    - Exploits fast rendering of points (`G4Polymarker::dots`) in OpenGL.
  - For "surfaces", each element is accumulated and used to construct a `G4Polyhedron` with `G4PolyhedronBoxMesh` or `G4PolyhedronTetMesh` as appropriate.
    - This is where the magic happens. Shared surfaces are removed by a very clever fast algorithm (Evgueni Tchernaiev) leaving a tractable `G4Polyhedron` of the outer faces.

```
void G4VSceneHandler::StandardSpecialMeshRendering(const G4Mesh& mesh)
// Standard way of special mesh rendering.
// MySceneHandler::AddCompound(const G4Mesh& mesh) may use this if
// appropriate or implement its own special mesh rendereing.
{
 G4bool implemented = false;
 switch (mesh.GetMeshType()) {
  case G4Mesh::rectangle: [[fallthrough]];
  case G4Mesh::nested3DRectangular:
   switch (fpViewer->GetViewParameters().GetSpecialMeshRenderingOption()) {
    case G4ViewParameters::meshAsDots:
     Draw3DRectMeshAsDots(mesh);  // Rectangular 3-deep mesh as dots
     implemented = true;
     break;
    case G4ViewParameters::meshAsSurfaces:
     Draw3DRectMeshAsSurfaces(mesh);  // Rectangular 3-deep mesh as surfaces
     implemented = true;
     break;
   }
   break;
  case G4Mesh::tetrahedron:
   switch (fpViewer->GetViewParameters().GetSpecialMeshRenderingOption()) {
    case G4ViewParameters::meshAsDots:
     DrawTetMeshAsDots(mesh);  // Tetrahedron mesh as dots
     implemented = true;
     break;
    case G4ViewParameters::meshAsSurfaces:
     DrawTetMeshAsSurfaces(mesh);  // Tetrahedron mesh as surfaces
     implemented = true;
     break;
   }
   break;
  case G4Mesh::cylinder: [[fallthrough]];
  case G4Mesh::sphere: [[fallthrough]];
  case G4Mesh::invalid: break;
 }
 if (!implemented) {
  G4VSceneHandler::AddCompound(mesh);  // Base class function - just print warning
 }
 return;
}
```

Generating Random Point in Tetrahedron

- Steps to generate a random point inside a tetrahedron defined by four vertices (v_0,v_1,v_2,v_3):
  - Generation of three random values: *s, t, u*. The point corresponding to these values is inside of a cube with the sides equal to 1
  - The cube can be subdivided in six equal tetrahedra. Appling a transformation that places the point into the tetrahedron at the origin
  - Calculation of the position in the original tetrahedron: $p = v_0(1 - s - t - u) + v_1 s + v_2 t + v_3 u$



- Detailed explanation of the algorithm:
  [C.Rocchini and P.Cignoni, *Generating Random Points in a Tetrahedron,* Journal of Graphics Tools, Volume 5, Issue 4 (2000)]

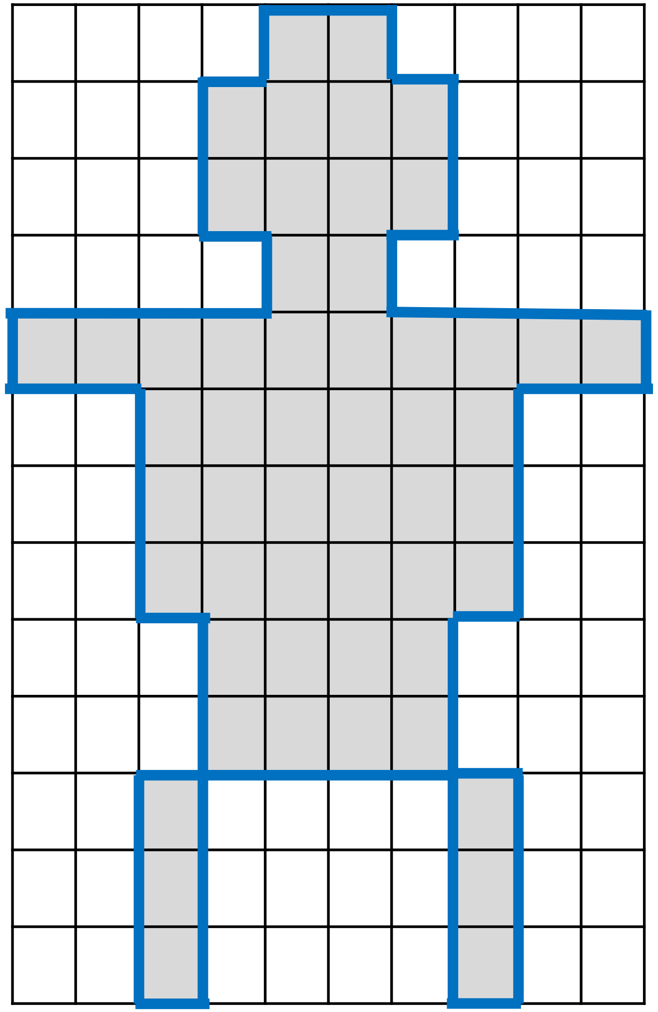
Constructing Polyhedron from rectangular mesh

- G4PolyhedronBoxMesh(sx, sy, sz, positions)

  sx, sy, sz – voxel dimensions

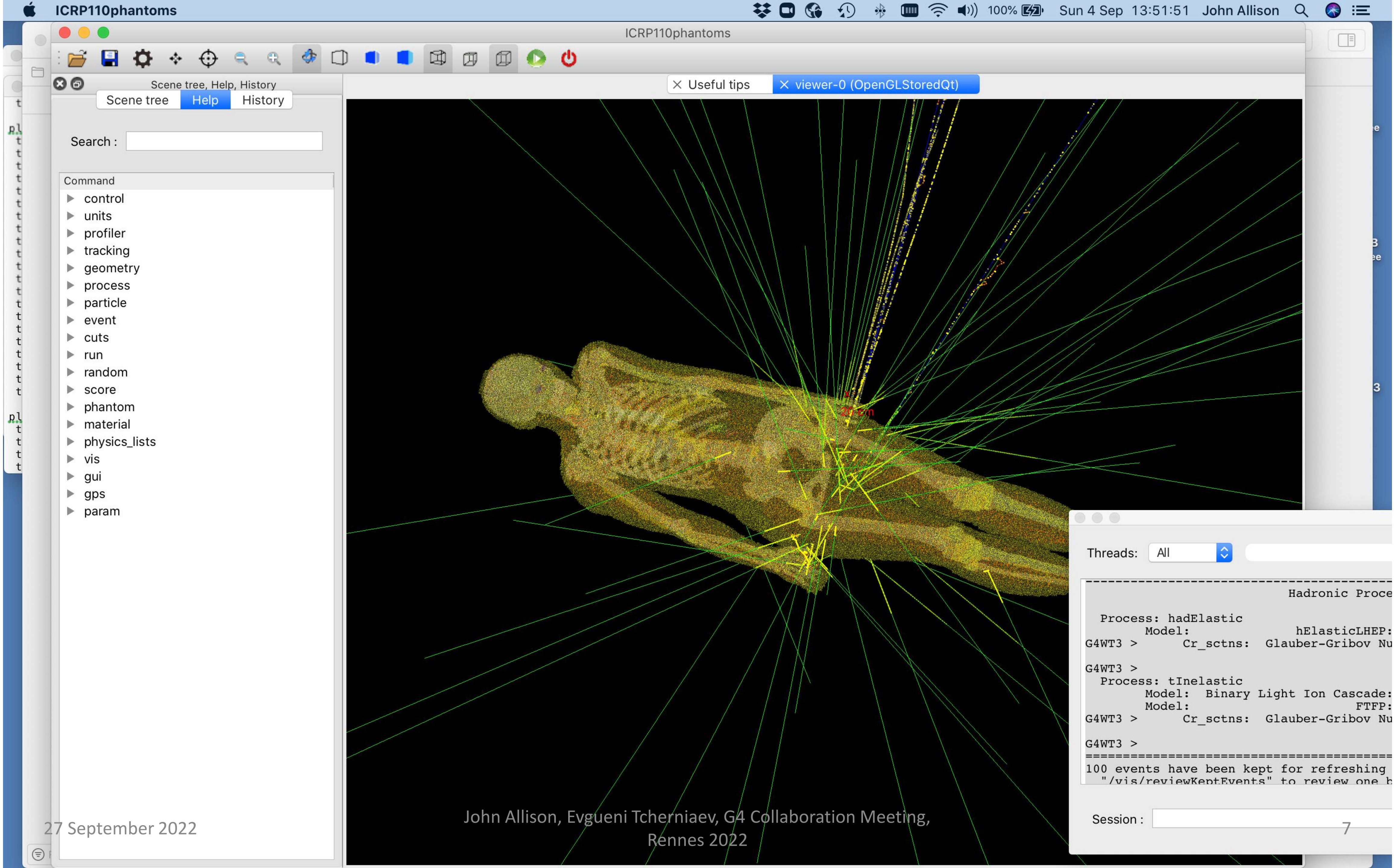
  positions – array of voxels centers

- The construction of a polyhedron is fast, it takes O(N) time, and is done in two steps:
  - Step one: Construction of a 3D grid surrounding the mesh, the grid cells corresponding to the mesh voxels are marked (grey cells on the sketch image)
  - Step two: Only cell faces that do not have marked neighboring cells are included into the resulting polyhedron (blue edges on the sketch)



ICRP110 Performance

- ICRP110 is an existing Geant4 advanced example:
  - From [ICRP, 2009. Adult Reference Computational Phantoms. ICRP Publication 110. Ann. ICRP 39 (2)].
  - Contains a `G4VNestedParameterisation` of 299x137x348 (14,255,124) boxes:
    - 3,885,291 of which are "visible" representing 52 organs
    - one dot per box are sorted into 52 `G4Polymarker` objects of different materials (and colour)
  - These 3,885,291 dots are rendered at about 5 fps on MacBook Pro (Retina, Mid 2012), 2.7 GHz Quad-Core Intel Core i7, 16 GB


ICRP110phantoms
Sun 4 Sep 13:51:51 John Allison
100%
Scene tree, Help, History
Scene tree
Help
History
Search :
Command
control
units
profiler
tracking
geometry
process
particle
event
cuts
run
random
score
phantom
material
physics_lists
vis
gui
gps
param
Useful tips
viewer-0 (OpenGLStoredQt)
Threads: All
Hadronic Proce
Process: hadElastic
Model: hElasticLHEP:
G4WT3 > Cr_sctns: Glauber-Gribov Nu
G4WT3 >
Process: tInelastic
Model: Binary Light Ion Cascade:
Model: FTFP:
G4WT3 > Cr_sctns: Glauber-Gribov Nu
G4WT3 >
100 events have been kept for refreshing
"/vis/reviewKeptEvents" to review one b
Session :

Constructing Polyhedron from tetrahedron mesh

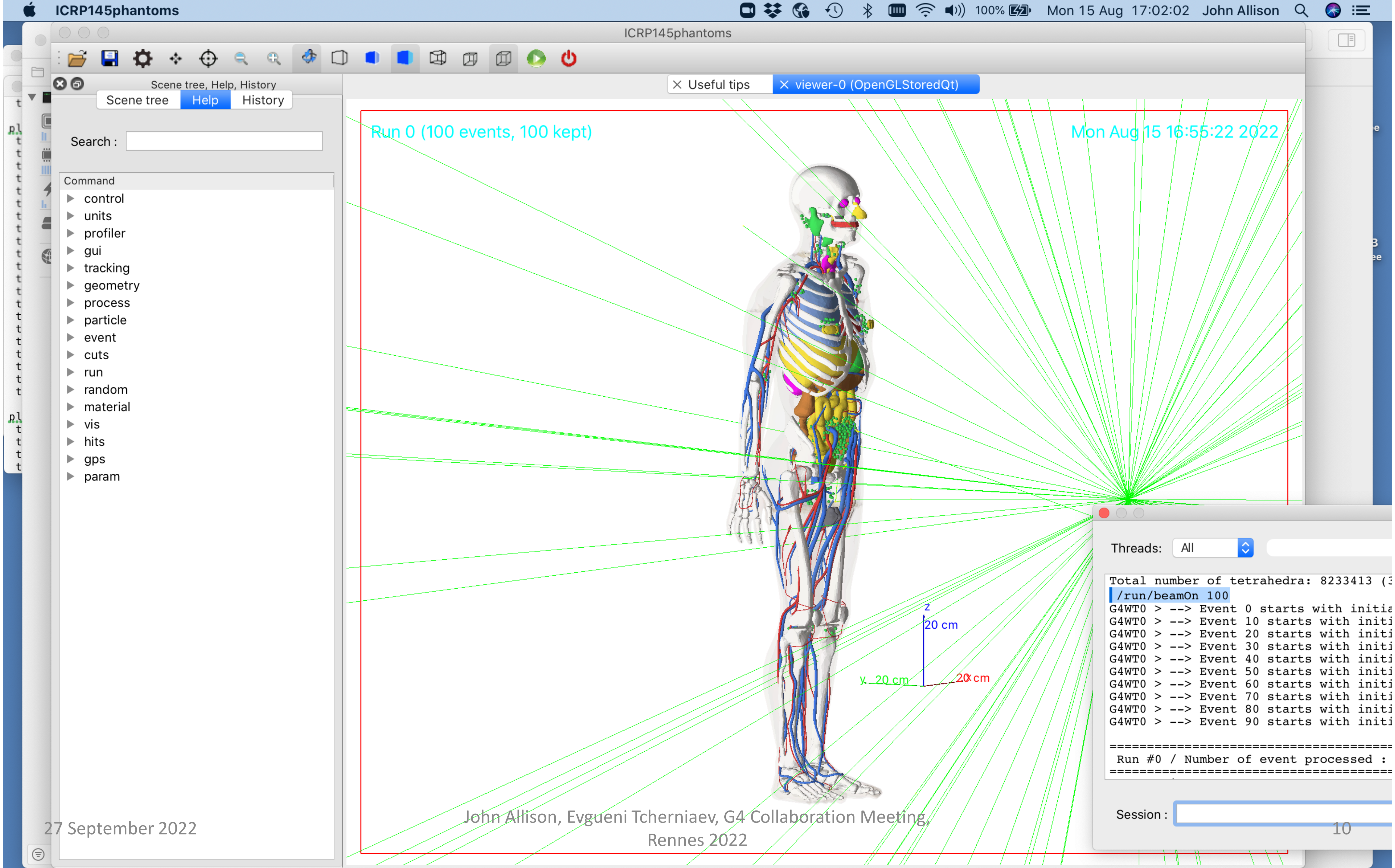
- G4PolyhedronTetMesh(vertices)

  vertices – tetrahedra, four vertices per tetrahedron
- Elimination of internal (shared) faces is done in two steps using a technique similar to “hash map”. Objects are sorted into lists, the index of the list for an object is calculated by applying a hash function to this object
  - Step one: Identification of coincident vertices. Below is a C++ code to generate a hash value for a vertex:

    ```
    auto index = std::hash(v.x());
    index ^= std::hash(v.y());
    index ^= std::hash(v.z());
    index %= n_of_lists;
    ```
  - Step two: Identification of internal/external faces and selecting the external ones. The faces are sorted into lists using smallest index among the three vertex indices that define the face

ICRP145 Performance

- ICRP145 is a new advanced example, presented in the Examples Session:
  - From ICRP, 2020. Adult mesh-type reference computational phantoms. ICRP Publication 145. Ann. ICRP 49(3)
    C.H. Kim, Y.S. Yeom, N. Petoussi-Henss, M. Zankl, W.E. Bolch, C. Lee, C. Choi, T.T. Nguyen, K. Eckerman, H.S. Kim, M.C. Han, R. Qiu, B.S. Chung, H. Han, B. Shin
  - Contains a one-level `G4PVParameterised` of 8,233,413 `G4Tets` (32,933,652 faces):
    - The 8,233,413 `G4Tets` represent 187 organs
    - By eliminating internal shared faces, tetrahedra are converted to 187 `G4Polyhedron` objects by material (and colour) with only 4,807,770 faces (14% of original number of faces)
    - Timings:
      - Geometry construction: 20 s
      - Physics tables: 90 s
      - Closing geometry: 20 s
      - Creating graphical database: 20 s
      - Rendering: 2 fps on MacBook Pro (Retina, Mid 2012), 2.7 GHz Quad-Core Intel Core i7, 16 GB


ICRP145phantoms
Mon 15 Aug 17:02:02
John Allison
Scene tree, Help, History
Scene tree
Help
History
Search :
Command
control
units
profiler
gui
tracking
geometry
process
particle
event
cuts
run
random
material
vis
hits
gps
param
Useful tips
viewer-0 (OpenGLStoredQt)
Run 0 (100 events, 100 kept)
Mon Aug 15 16:55:22 2022
20 cm
y 20 cm
20 cm
Threads:
All
Total number of tetrahedra: 8233413
/run/beamOn 100
G4WT0 > --> Event 0 starts with initia
G4WT0 > --> Event 10 starts with initi
G4WT0 > --> Event 20 starts with initi
G4WT0 > --> Event 30 starts with initi
G4WT0 > --> Event 40 starts with initi
G4WT0 > --> Event 50 starts with initi
G4WT0 > --> Event 60 starts with initi
G4WT0 > --> Event 70 starts with initi
G4WT0 > --> Event 80 starts with initi
G4WT0 > --> Event 90 starts with initi
Run #0 / Number of event processed :
Session :

That’s it

Thankyou

…except for documentation !!! 😧