Status of the development of the Unified Solids library

Marek Gayer, CERN PH/SFT

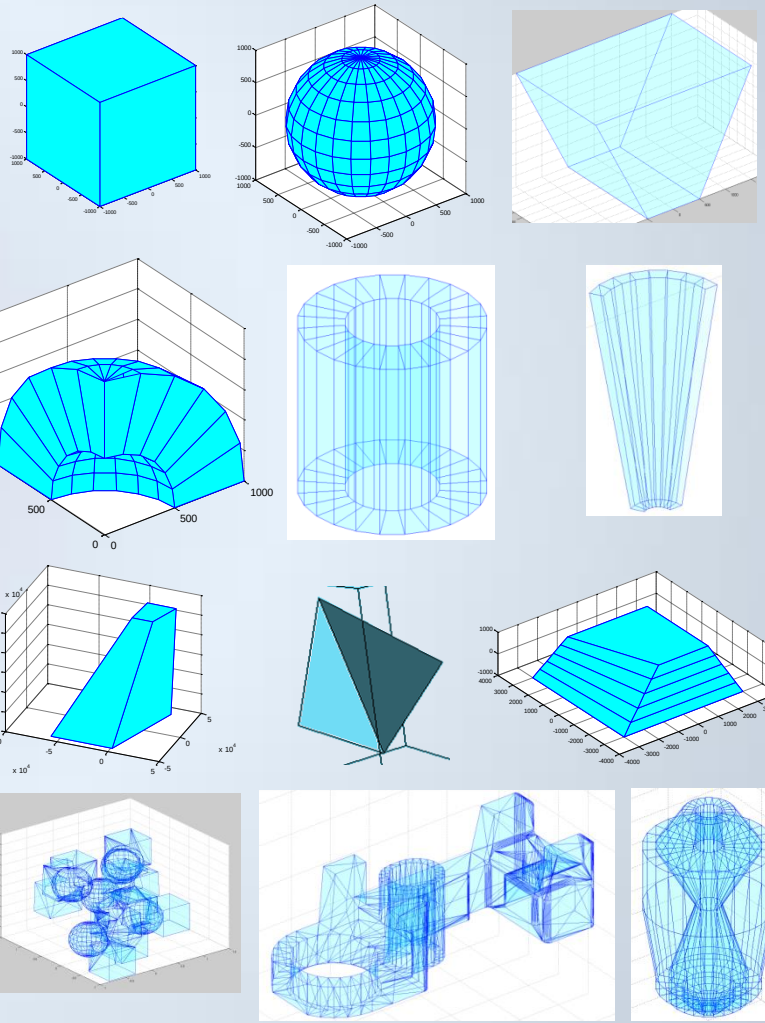
2nd AIDA Annual Meeting, Frascati 2013

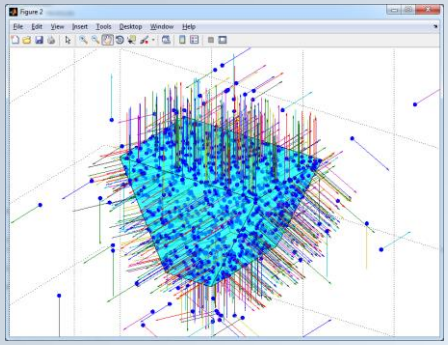
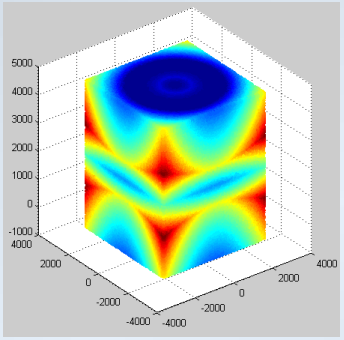
Motivations for a common solids library

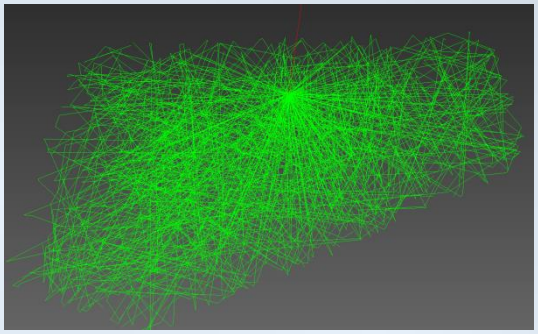
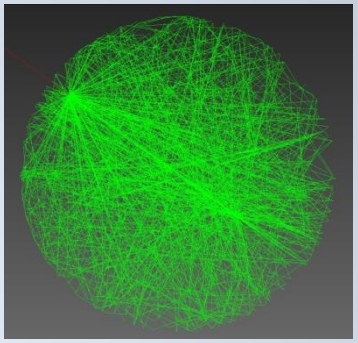
- Optimize and guarantee better long-term maintenance of ROOT and Gean4 solids libraries
  - A rough estimation indicates that about 70-80% of code investment for the geometry modeler concerns solids, to guarantee the required precision and efficiency in a huge variety of combinations
- Create a single high quality library to replace solid libraries in Geant4 and ROOT
  - Starting from what exists today in Geant4 and Root
  - Adopt a single type for each shape
  - Make fast Tessellated Solid and new Multi-Union
  - Reach complete conformance to GDML solids schema
- Create extensive testing suite

Solids implemented so far

- Box
- Orb
- Trapezoid
- Sphere (+ sphere section)
- Tube (+ cylindrical section)
- Cone (+ conical section)
- Generic trapezoid
- Tetrahedron
- Arbitrary Trapezoid (ongoing)
- **Multi-Union**
- **Tessellated Solid**
- **Polycone** (close to be finished)



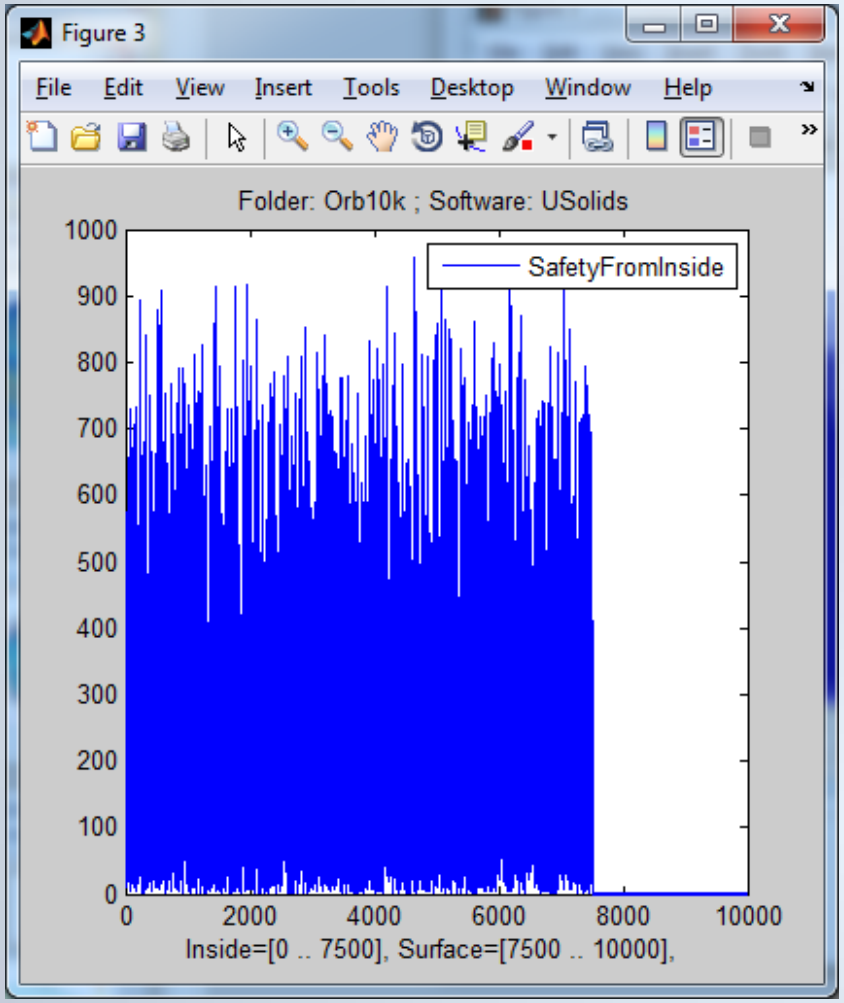
Testing Suite

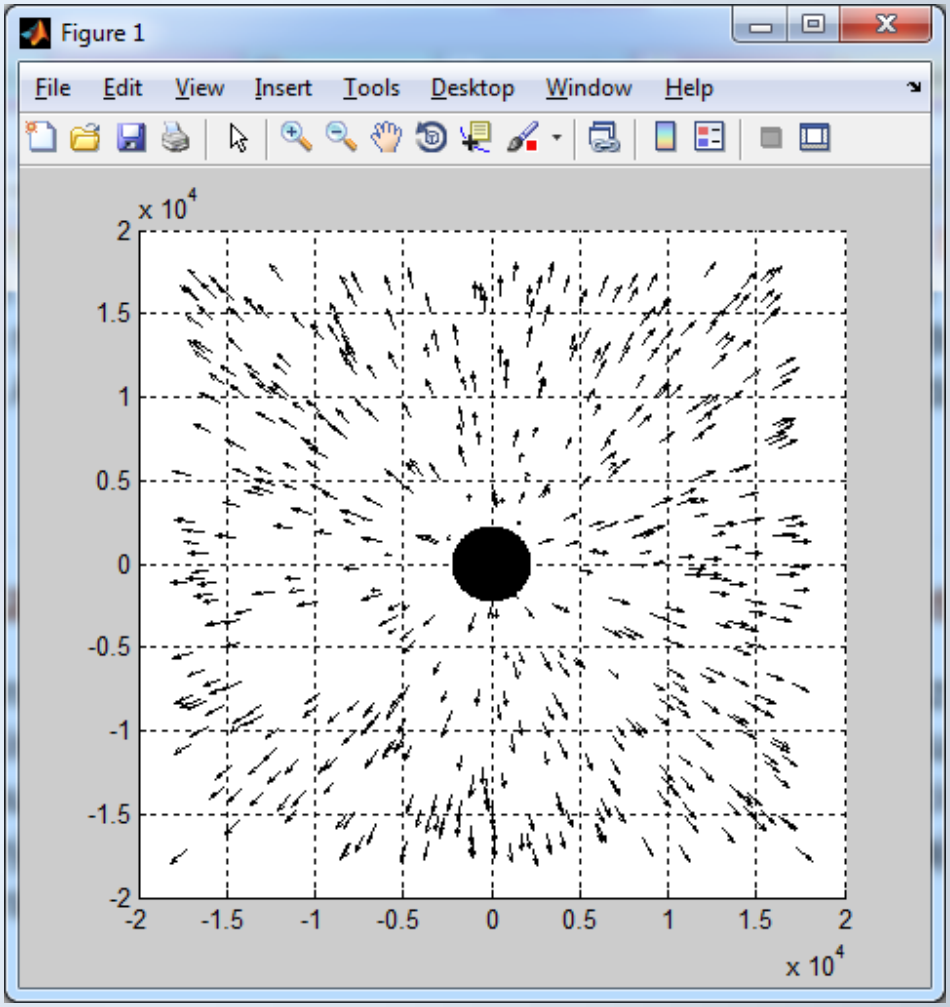
- Unified Solid Batch Test
- Optical Escape

Unified Solid Batch Test

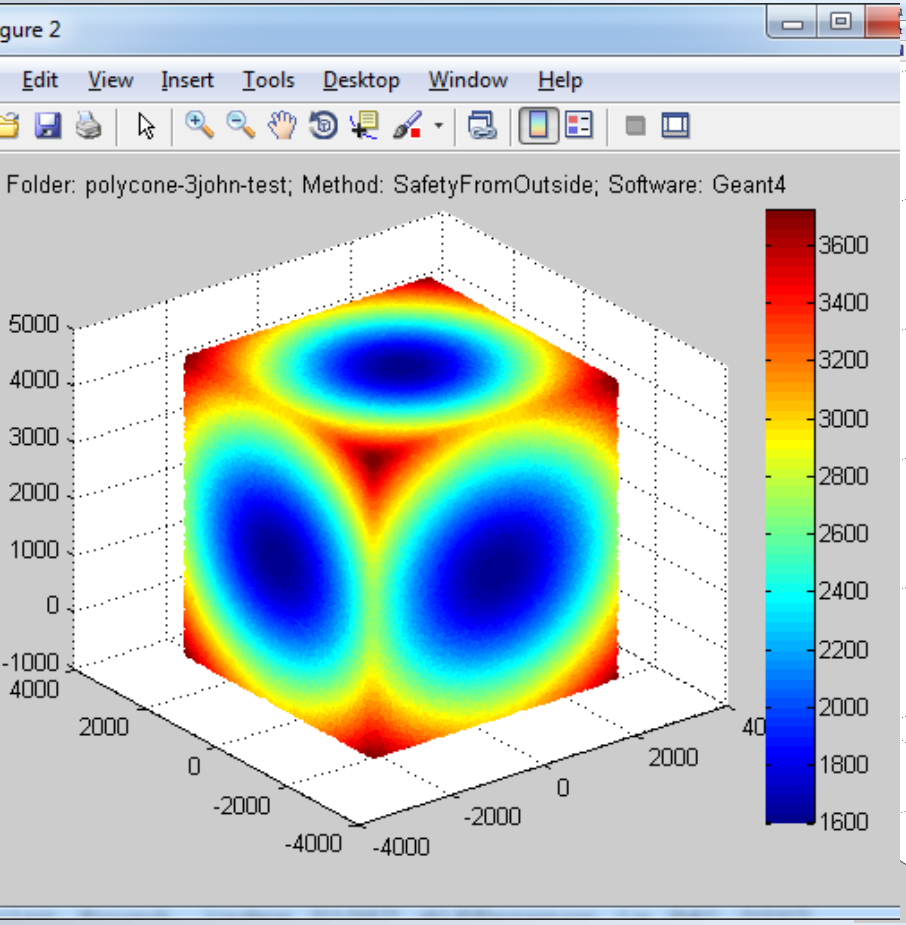
- **New powerful diagnosis tool for testing solids of ROOT, Geant4 and Unified Solids library designed for transition to Unified Solids**
  - The core part of Unified Solids testing
  - Testing Unified Solids together with existing Geant4 and ROOT solids
  - Testing methods with groups of points inside, outside and on surface
  - Compares performance and output values from different codes
  - Support for batch configuration and execution of tests
  - Contains also tests from the original SBT for points and voxels
- Helps us to make sure we have similar or better performance in each method and different test cases
- Useful to detect, report and fix numerous bugs or suspect return values in Geant4, ROOT and Unified Solids
- Two phases
  - Sampling phase (generation of data sets, implemented as C++ app.)
  - Analysis phase (data post-processing, implemented as **MATLAB** scripts)

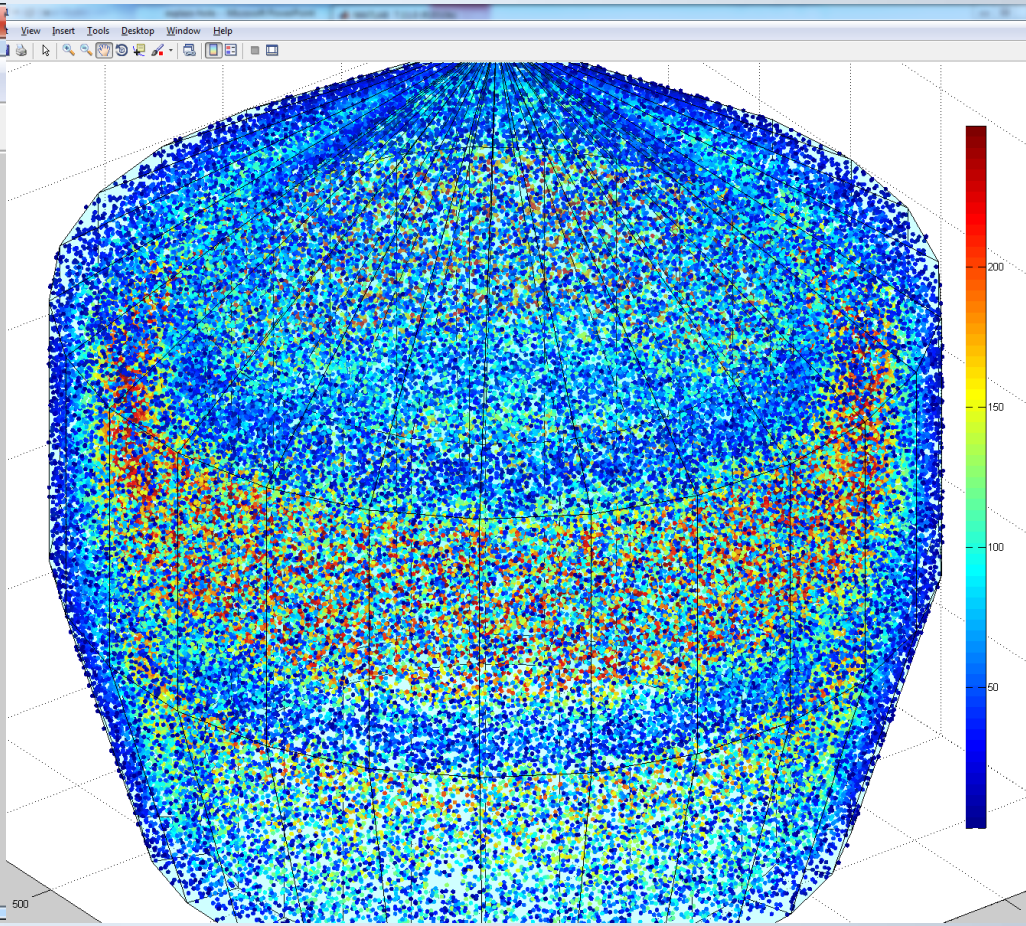
Visualization of scalar and vector data sets



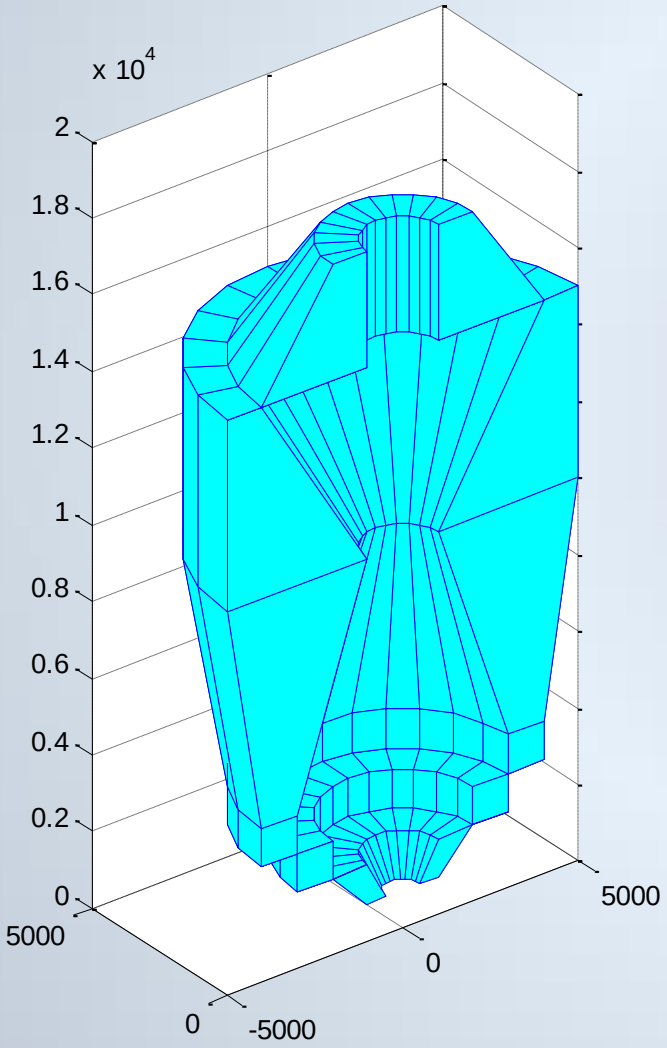


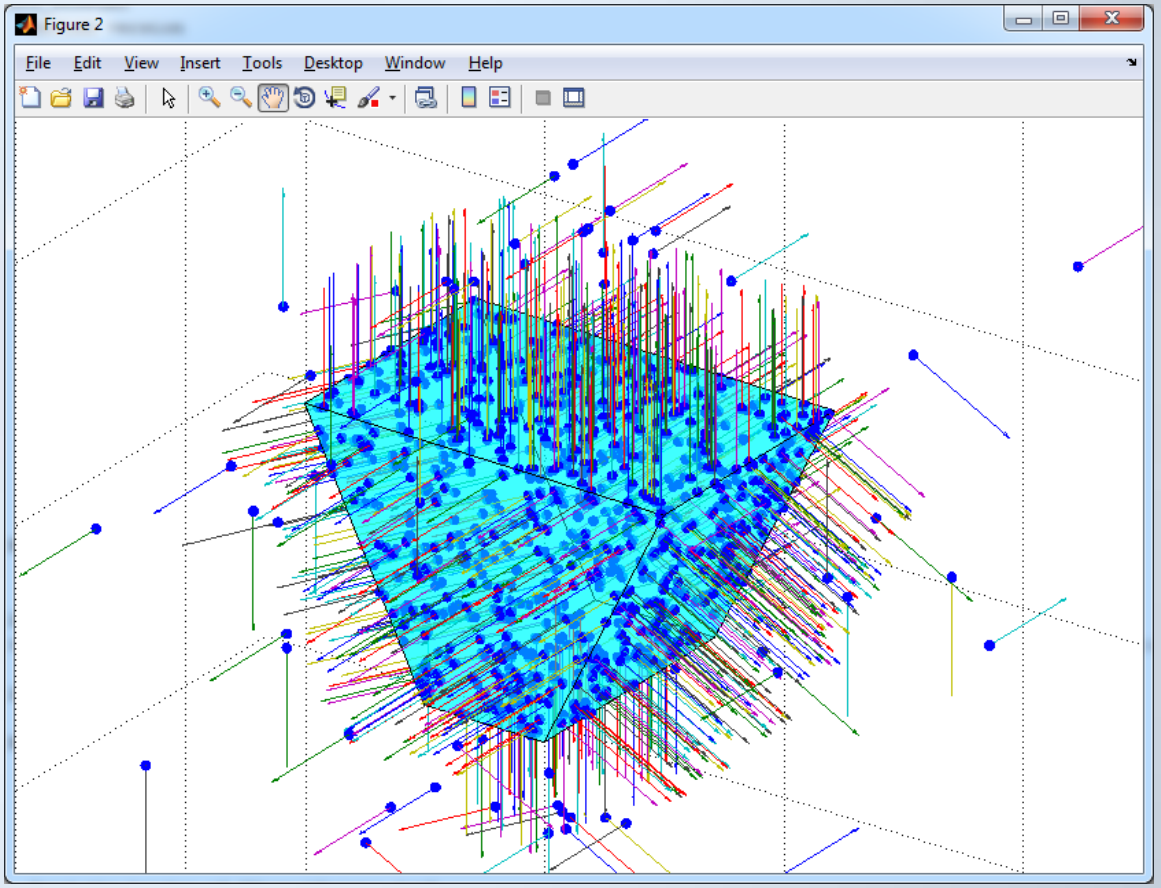
3D plots allowing to overview data sets



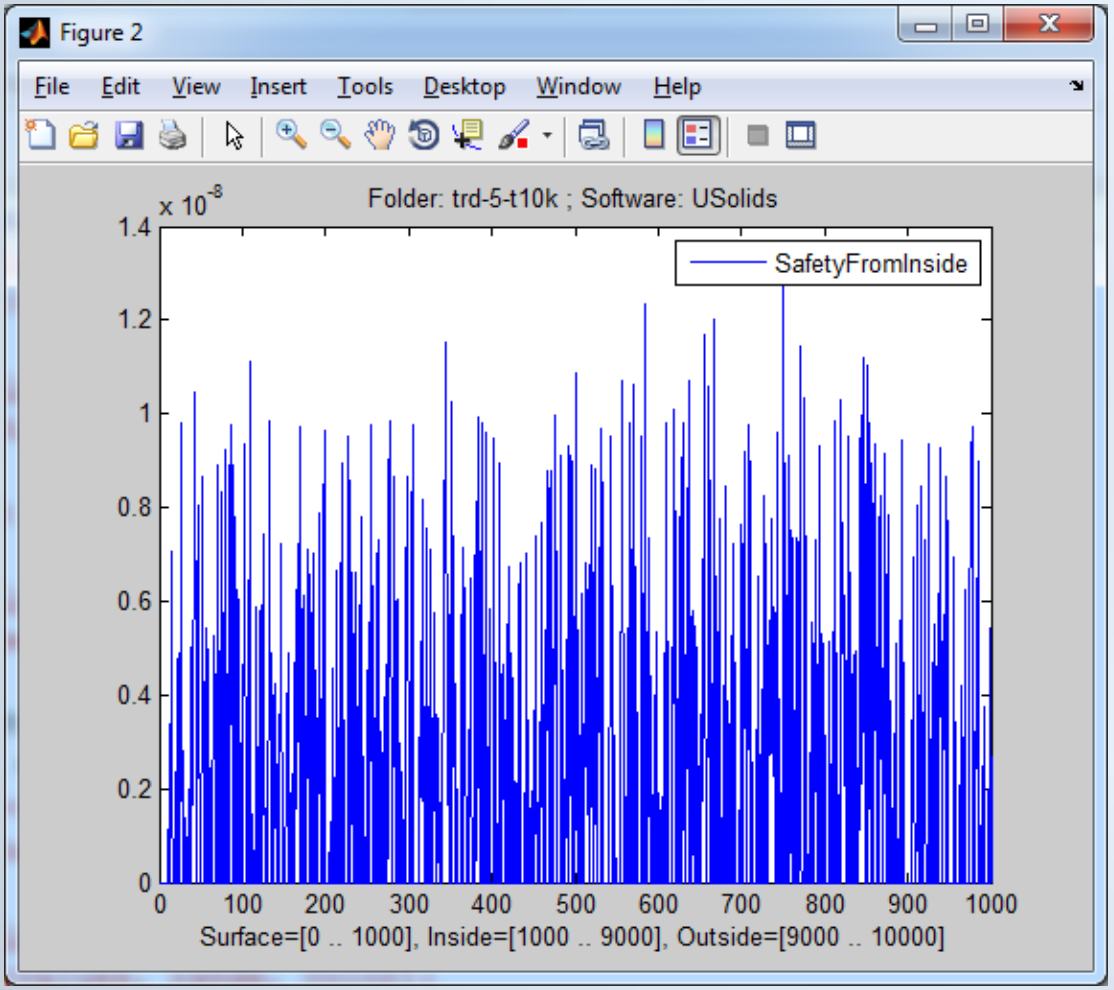


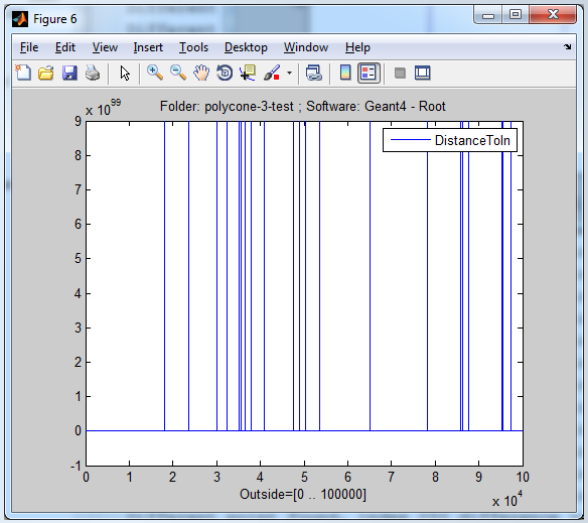
3D visualization of investigated shapes

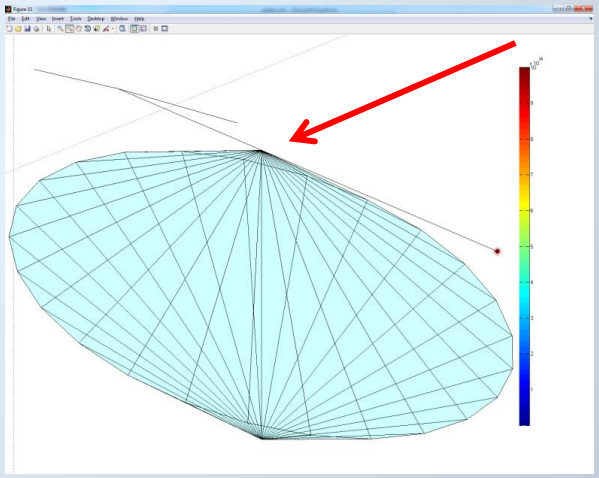




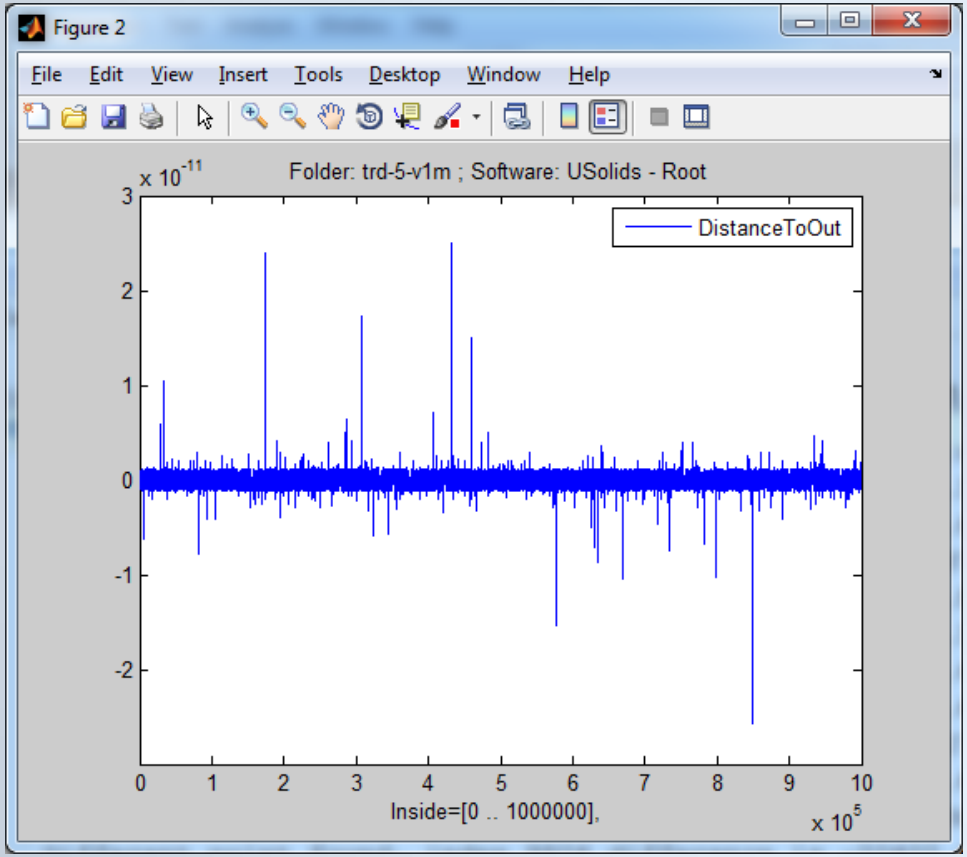
Support for regions of data, focusing on sub-parts

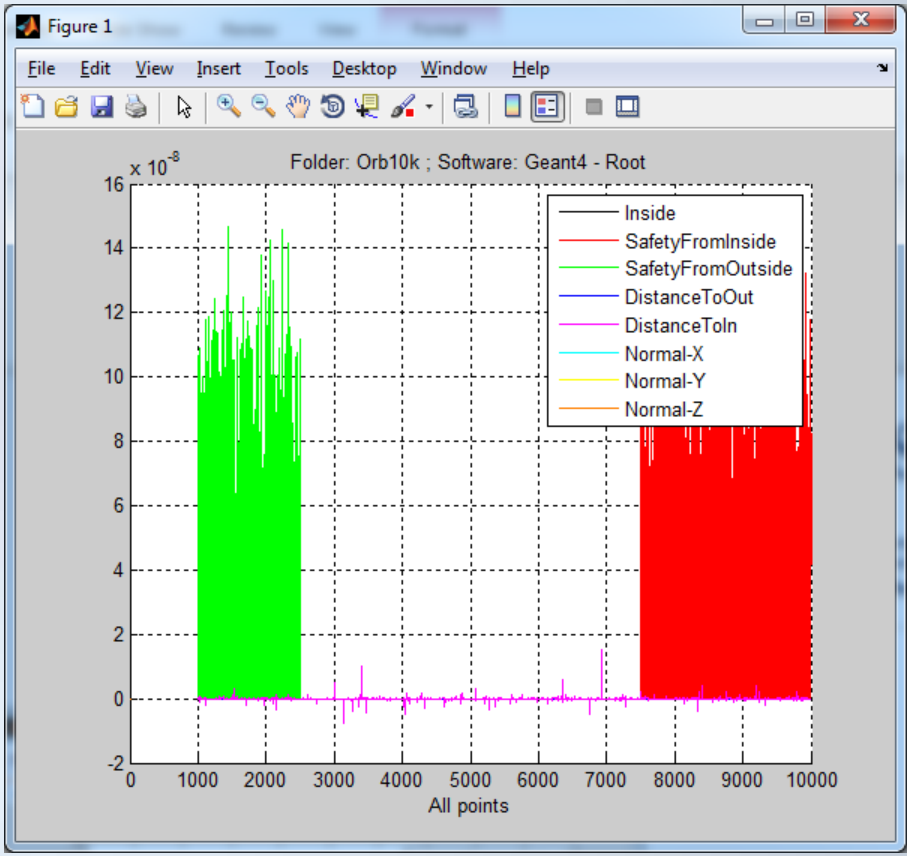




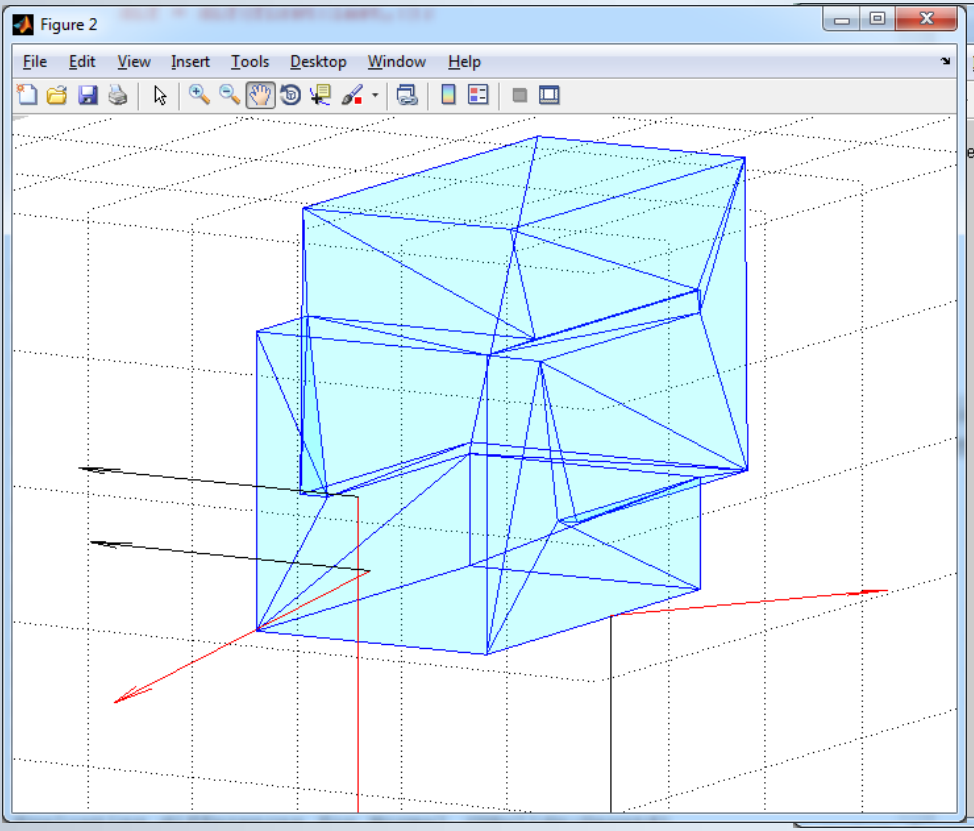


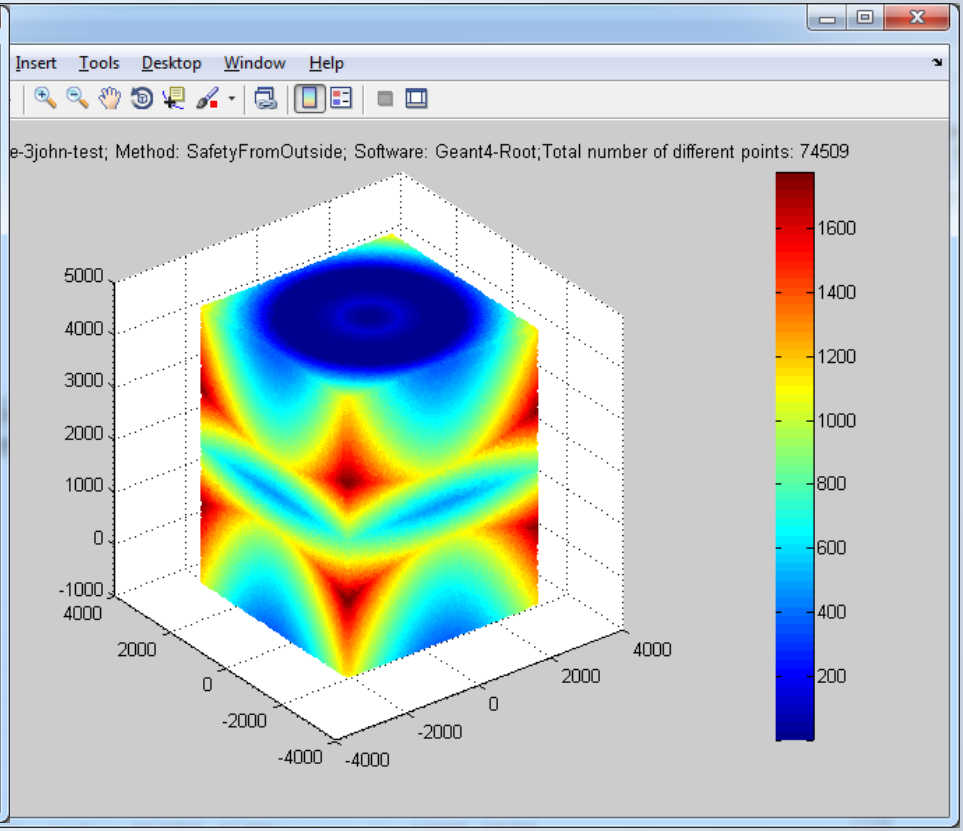
Visual analysis of differences



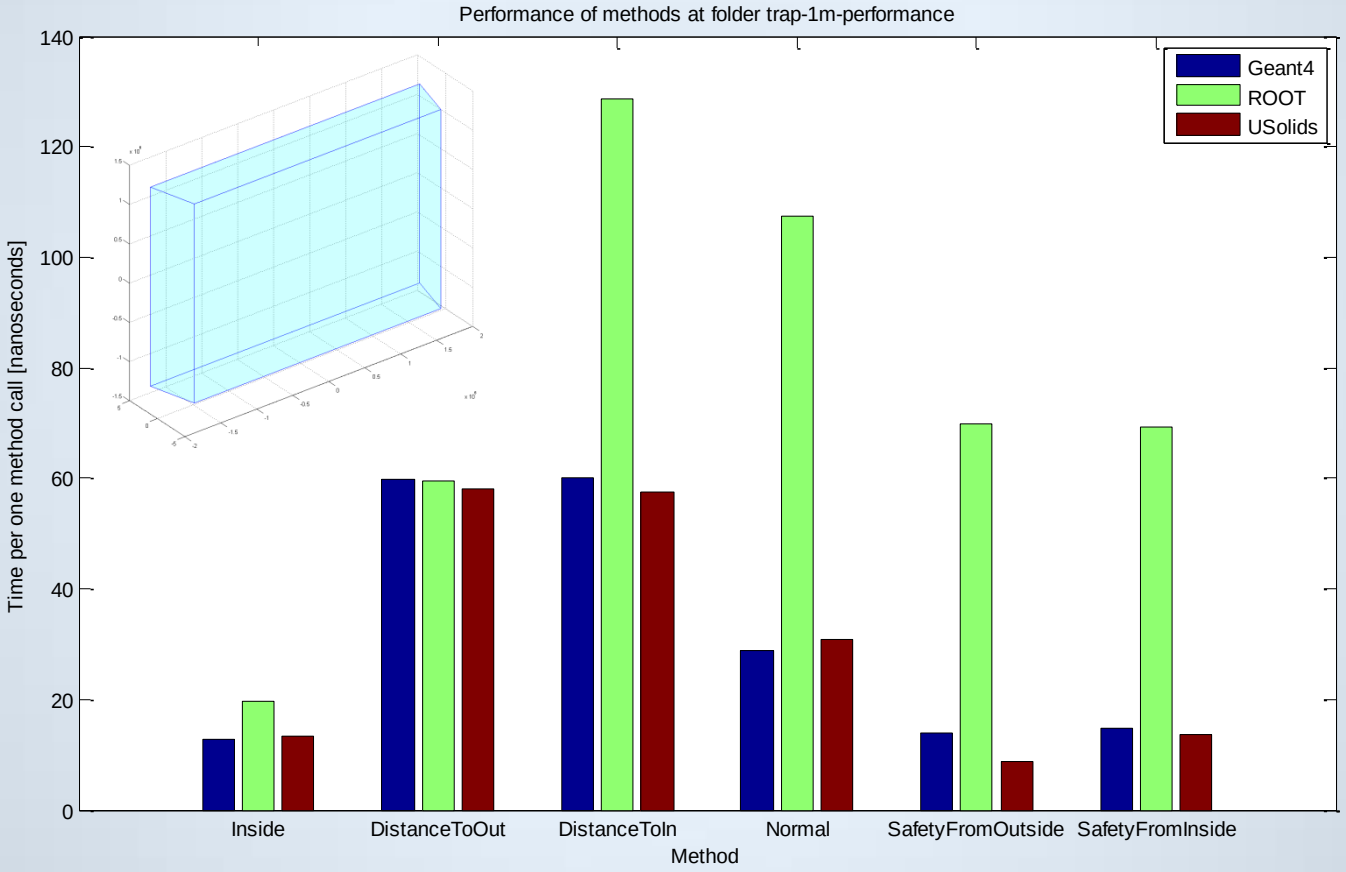


Visual analysis of differences in 3D

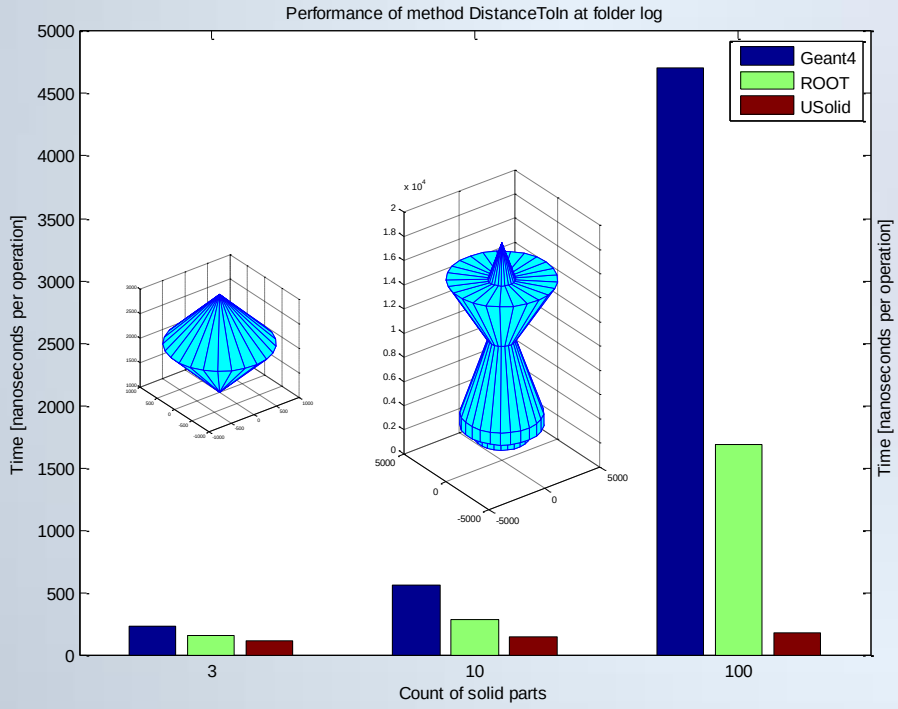


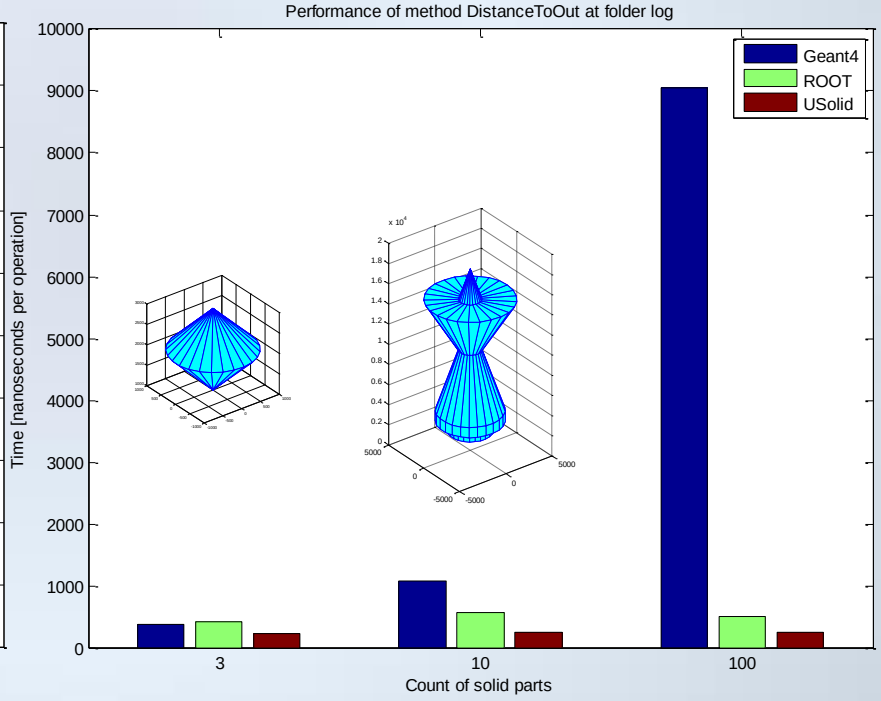


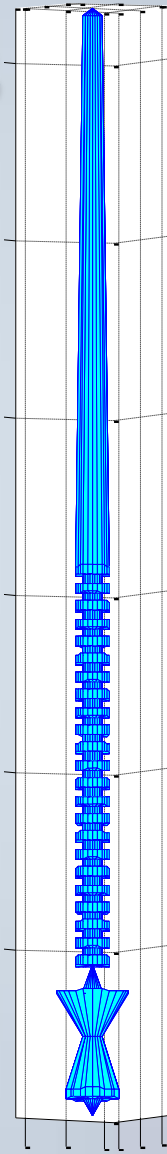
Graphs with comparison of performance



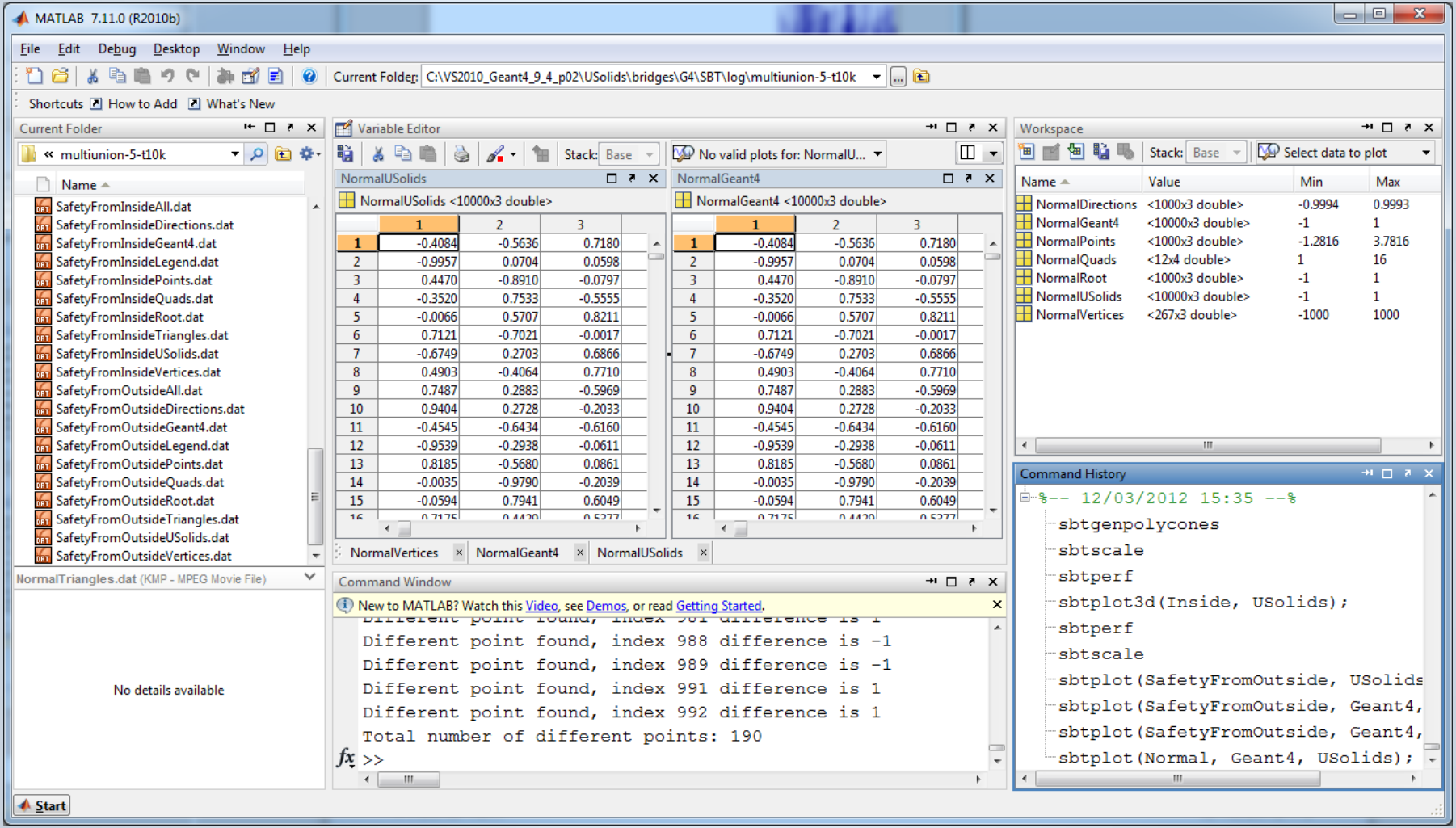
Visualization of scalability performance for specific solids







Inspection of values and differences of scalar and vector data sets

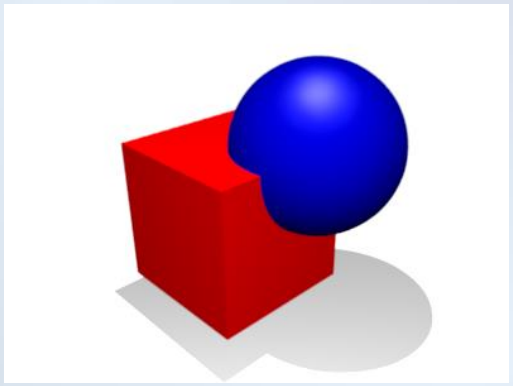


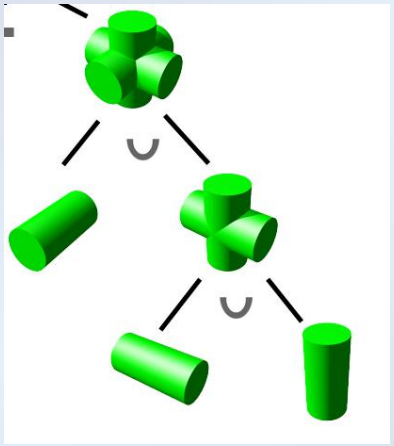
New Multi-Union solid

...

Boolean Union solids

- Existing CSG Boolean solids (ROOT and Geant4) are represented as binary trees
  - To solve navigation requests, most of the solids composing a complex one have to be checked
  - Scalability is typically linear => low performance for solids composed of many parts



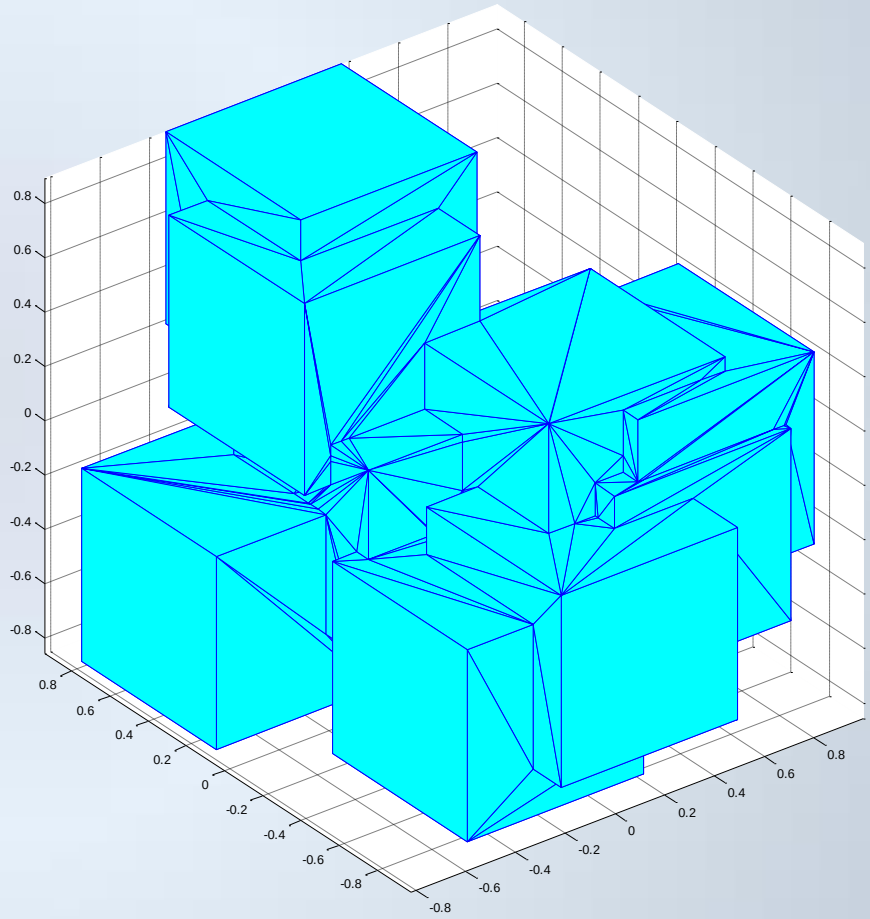


[The pictures were produced by users of Wikipedia "Captain Sprite" and "Zottie" and are available under Creative Commons Attribution-Share Alike 3.0 Unported license]

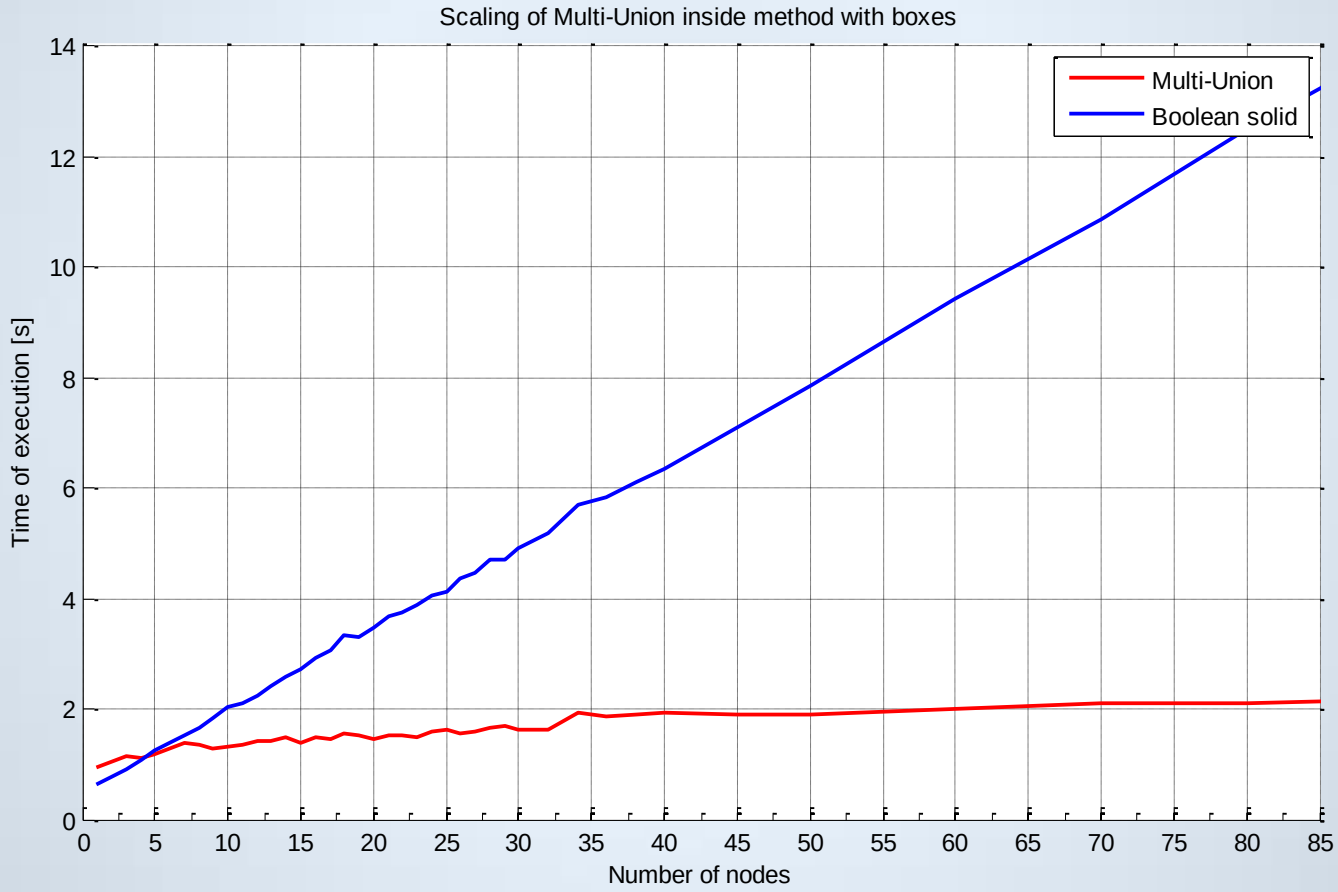
Boolean Union solid:
is binary tree of solids, either primitive or Boolean

Multi-Union solid

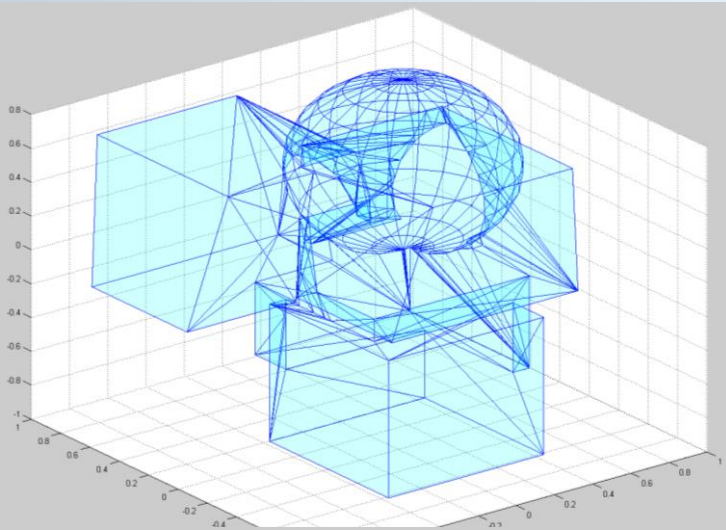
- We implemented a new solid as a union of many solids using voxelization technique to optimize speed and scalability
  - 3D space partition for fast localization of components
  - Aiming for a log(n) scalability
- Useful for complex composites made of many solids

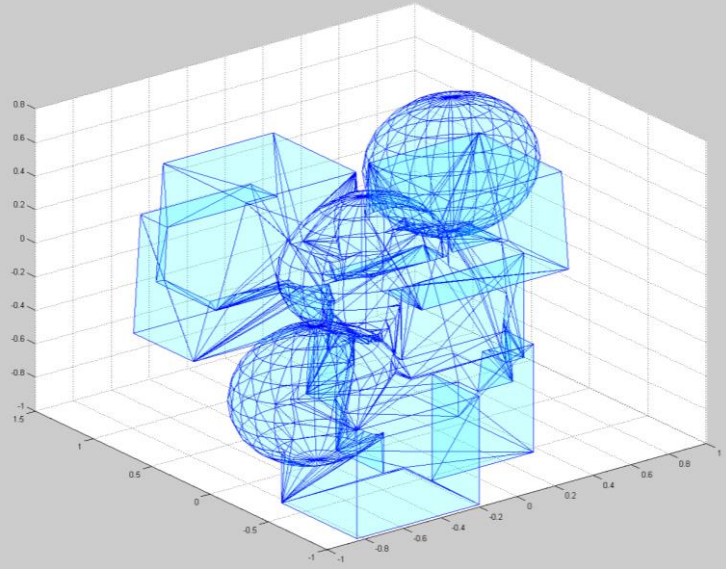


Scaling of Multi-Union vs. Boolean solid

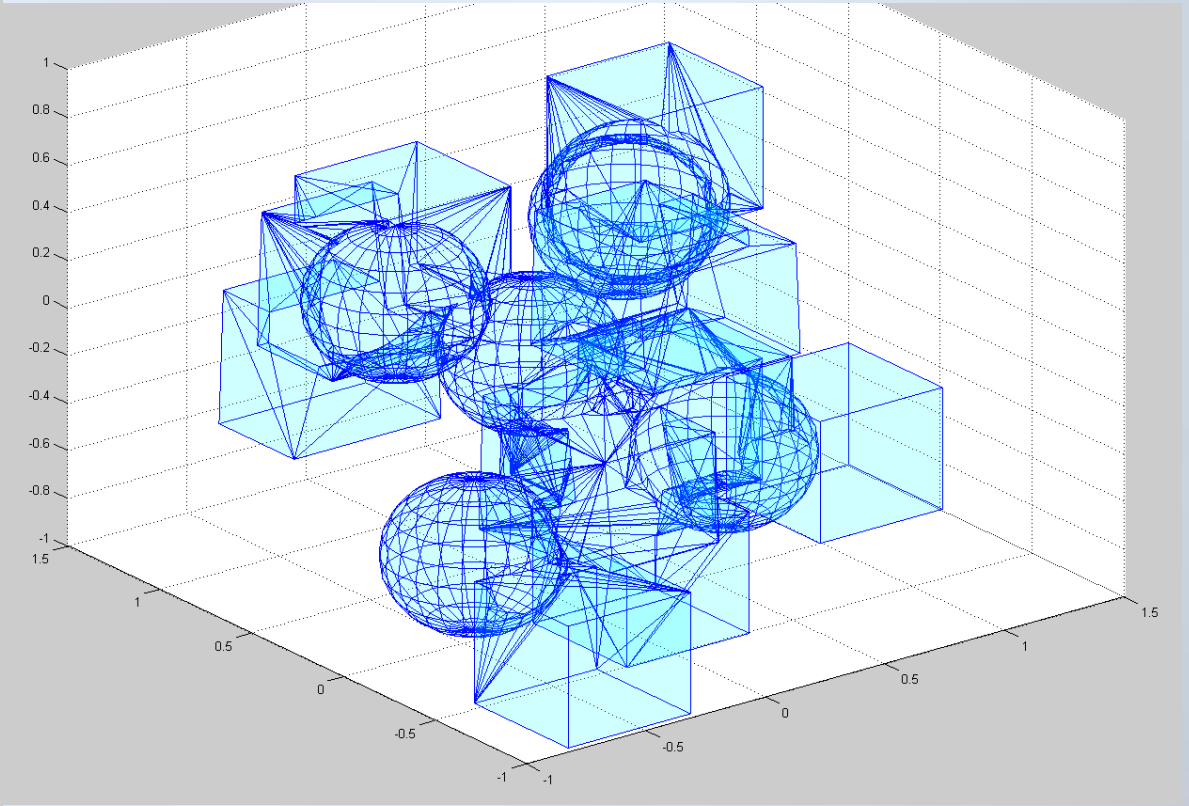


Test multi-union solid for scalability measurements

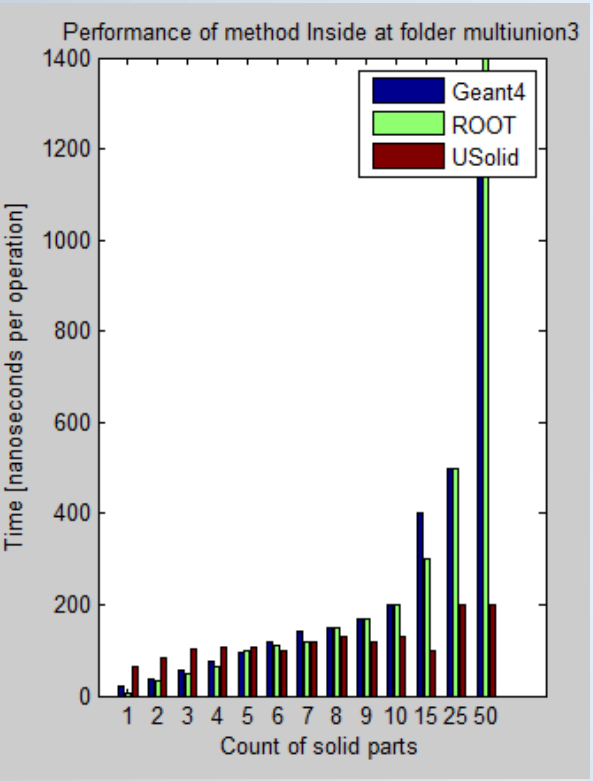


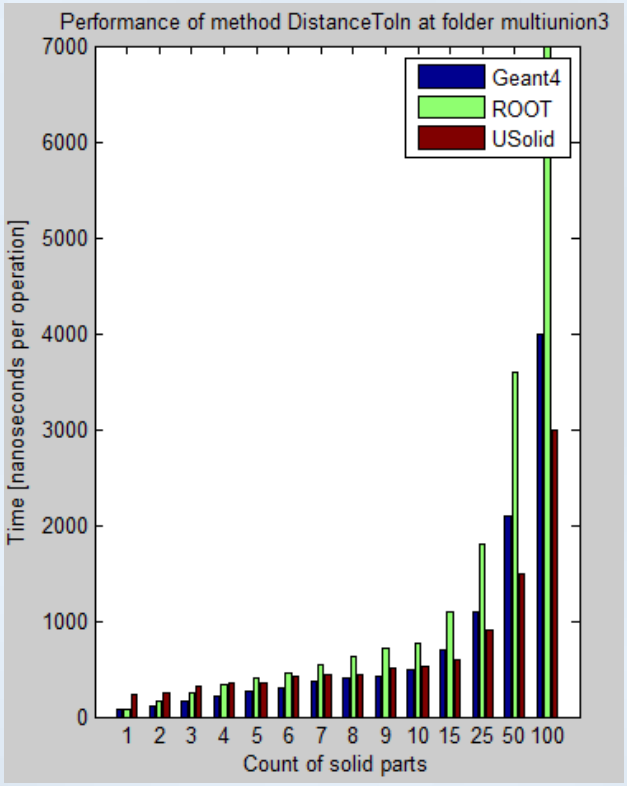


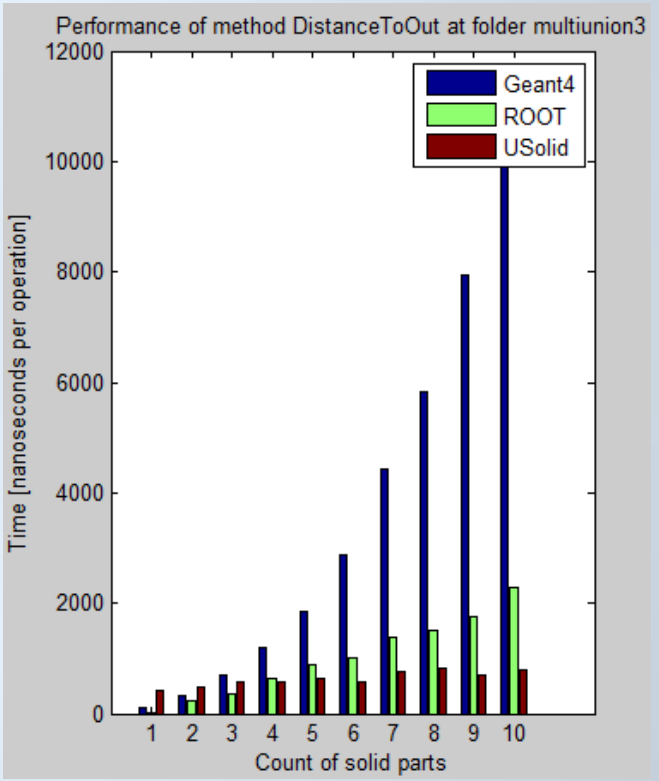
Union of random boxes, orbs and trapezoids (on pictures with 5, 10, 20 solids)



The most performance critical methods



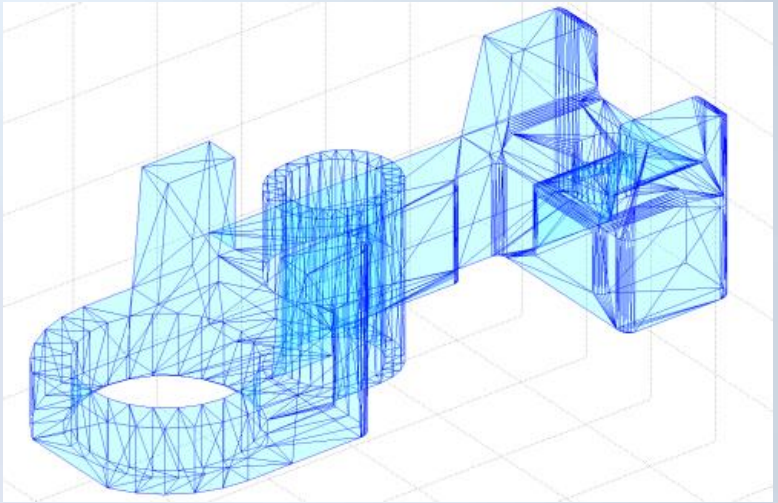


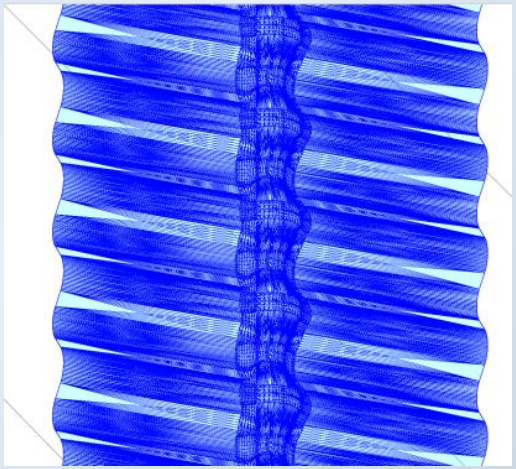


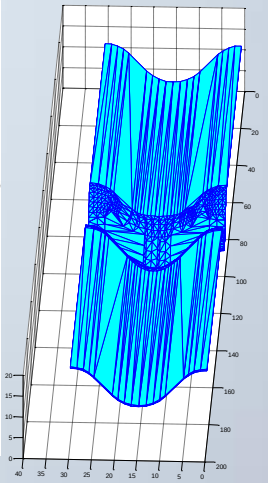
Fast Tessellated Solid

Fast Tessellated Solid

- Made from connected triangular and quadrangular facets forming solid
- Old implementation **was slow**, no spatial optimization
- We use spatial division of facets into 3D grid forming voxels (i.e. we are "voxelizing")
- Voxelizer is based on bitmasks logical and operations during initialization and on pre-calculated list of facets candidates during runtime
- We are **factor thousands** faster for LHCb provided foil test case (164k facets)



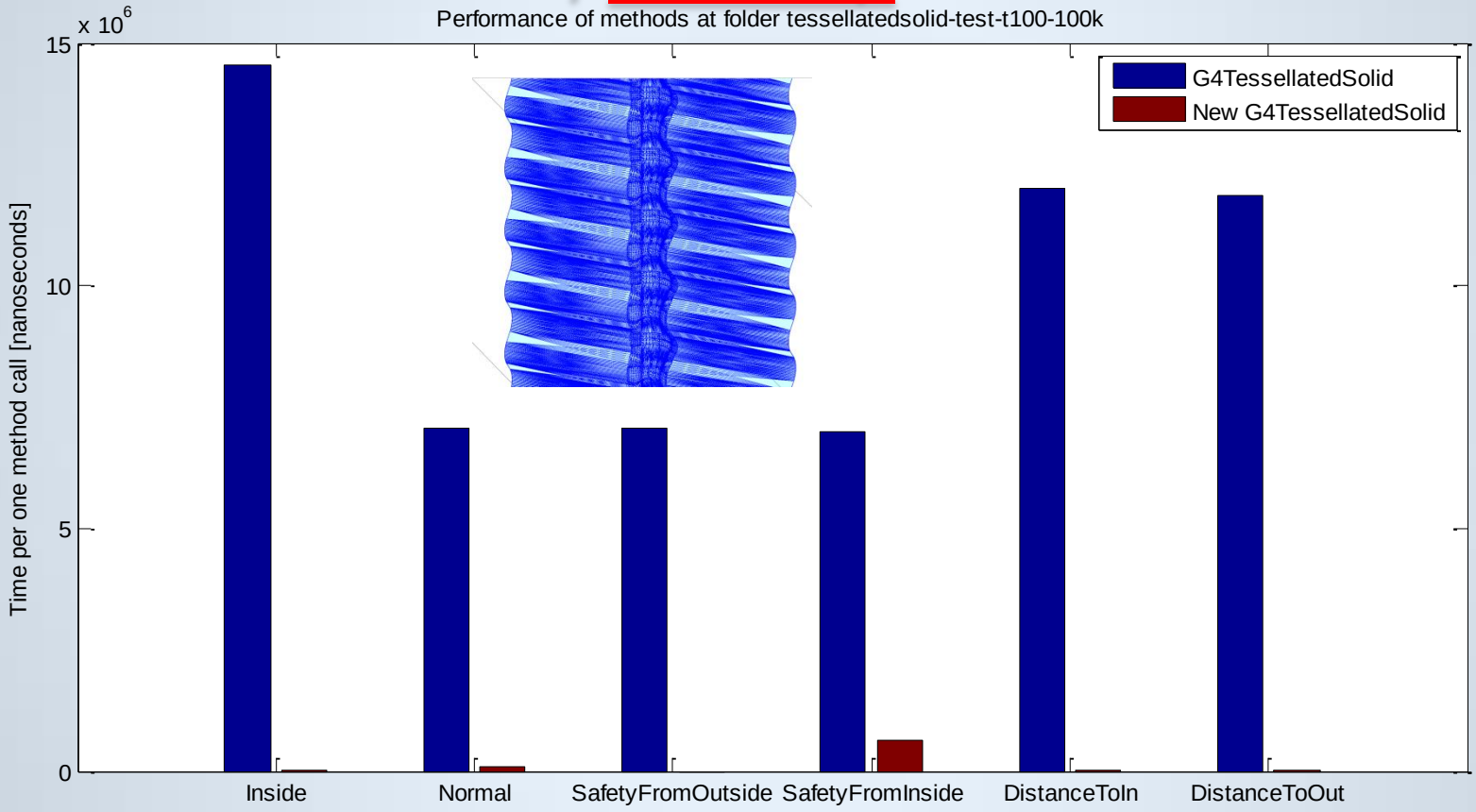




Fast Tessellated Solid

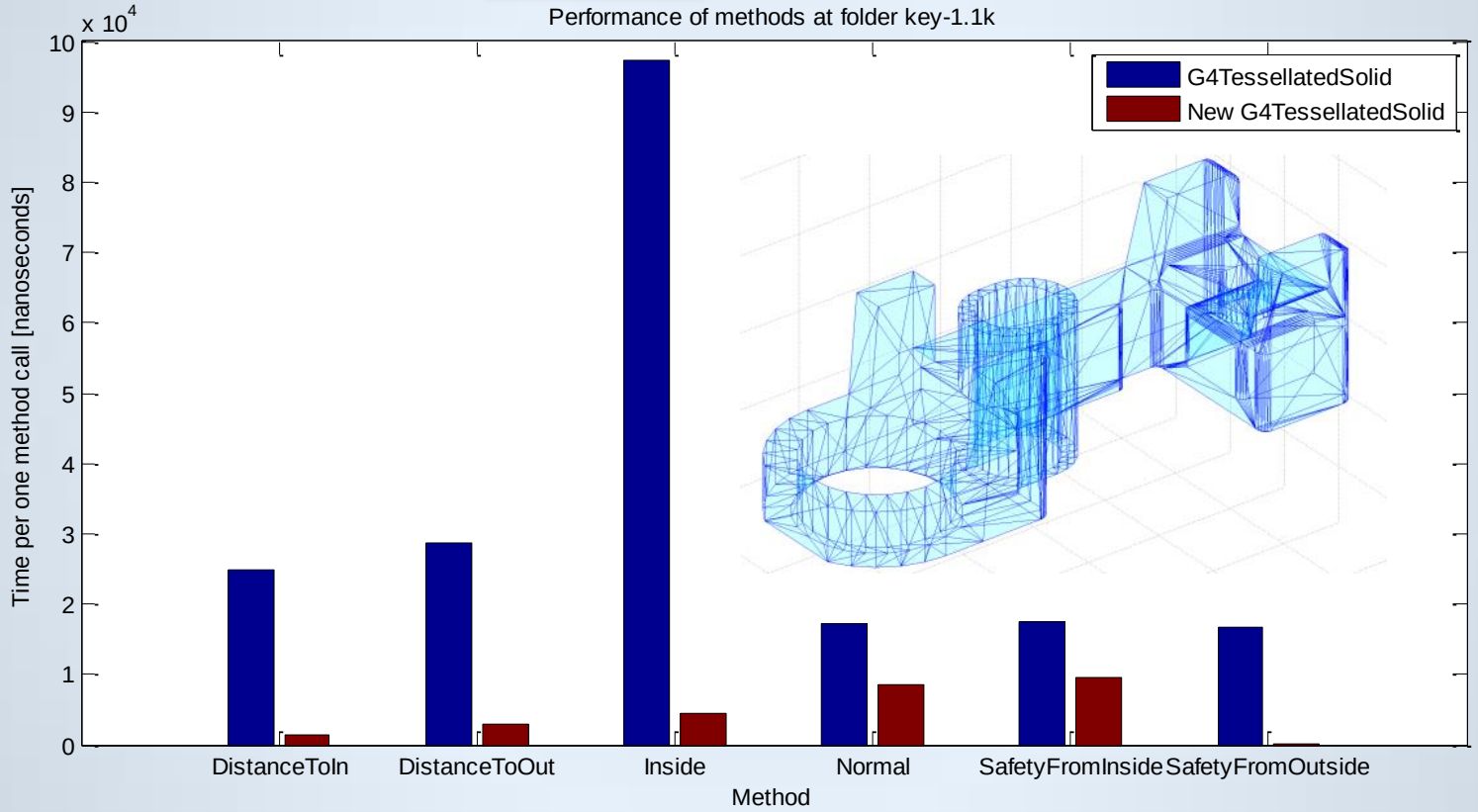
- Voxelization resulted in a significant performance enhancement in all performance critical methods
- Also, old Tesselated Solid had several weak parts of algorithm, used at initialization which had n^2 complexity.
  - This sometimes caused very huge delays when loading (e.g. in case of foil with 164k faces)
  - Rewrote to have $n \cdot \log n$ complexity
  - Loading the solid is much faster. What before took minutes, now takes seconds
- Implemented also in Geant4 **9.6** as G4TesselatedSolid
- By improving design of classes and removing inefficiencies, total memory save is about ~50%,
  - Voxelization has overhead vs. original G4TS, which reduces to a total memory save of ~25%

Performance – 164k/SCL5 with 10k/100k/1M voxels



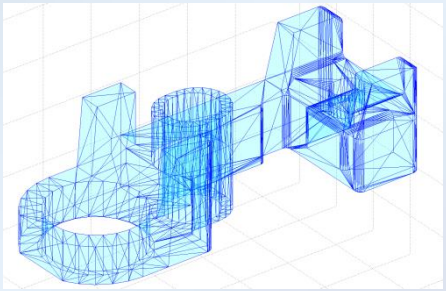
- Speedup: 894x 10.3x 1000x+ ~8.7x 412x 1183x
- Speedup: 2423x 72x 1000x+ ~10.9x 1334x 1976x
- Speedup: 2925x 46x 1000x+ ~9.8x 1332x 2968x

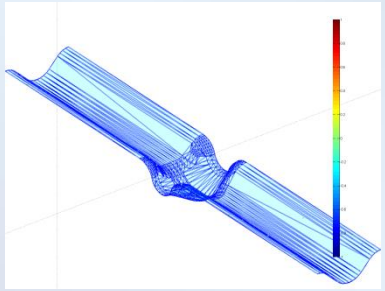
Performance – 1.1k/SCL 5 with 10k/1k voxels

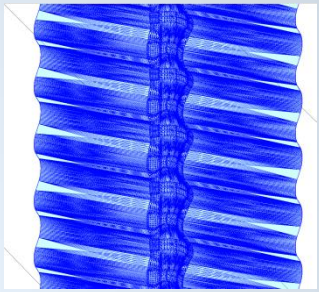


- **Speedup: 17x** **9.53** **22.0x** **2+** **1.8x** **166+**
- **Speedup: 22x** **9.33** **27.5x** **20+** **2x** **2000+**

Memory footprint change in comparison with old version



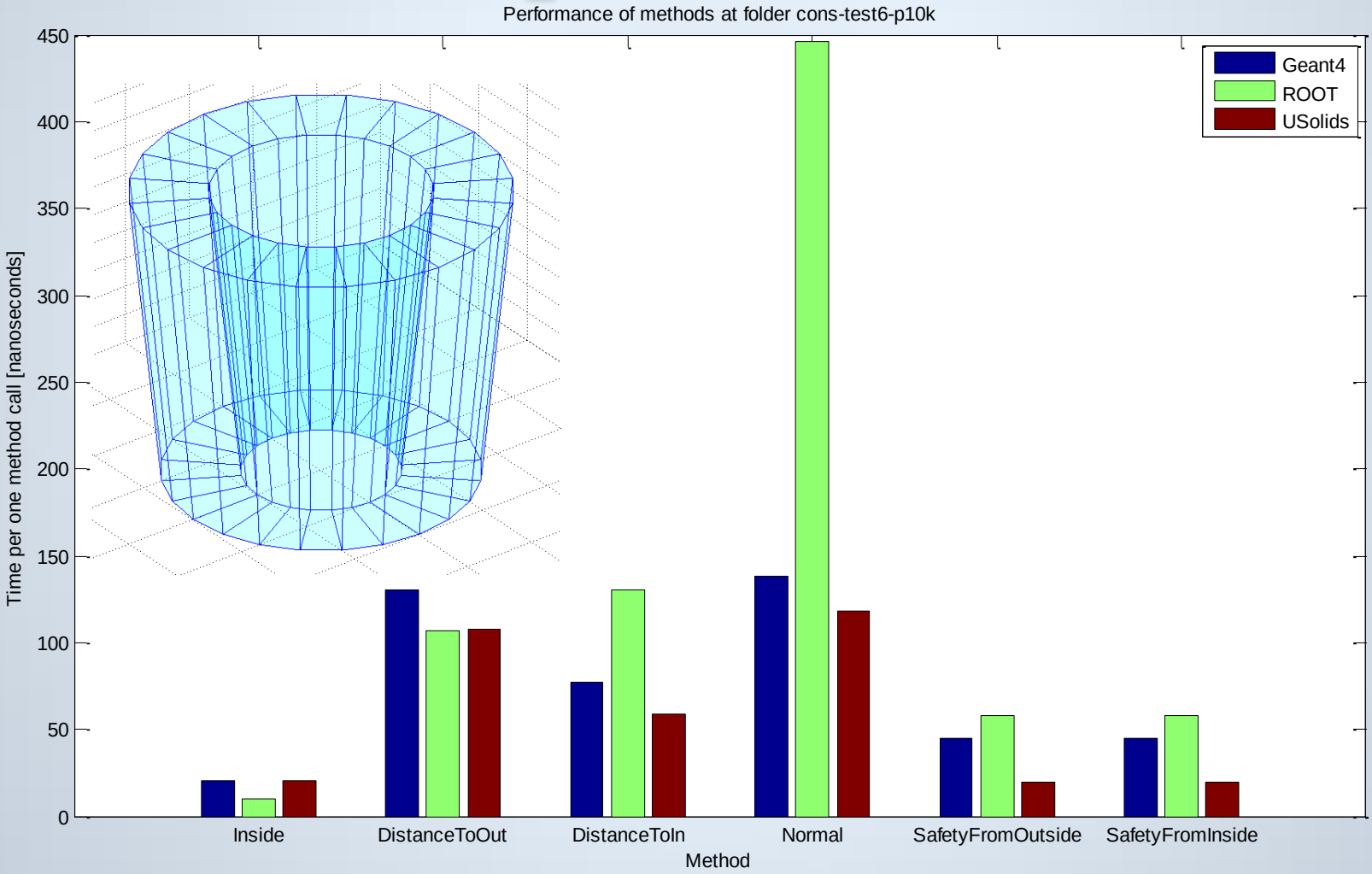




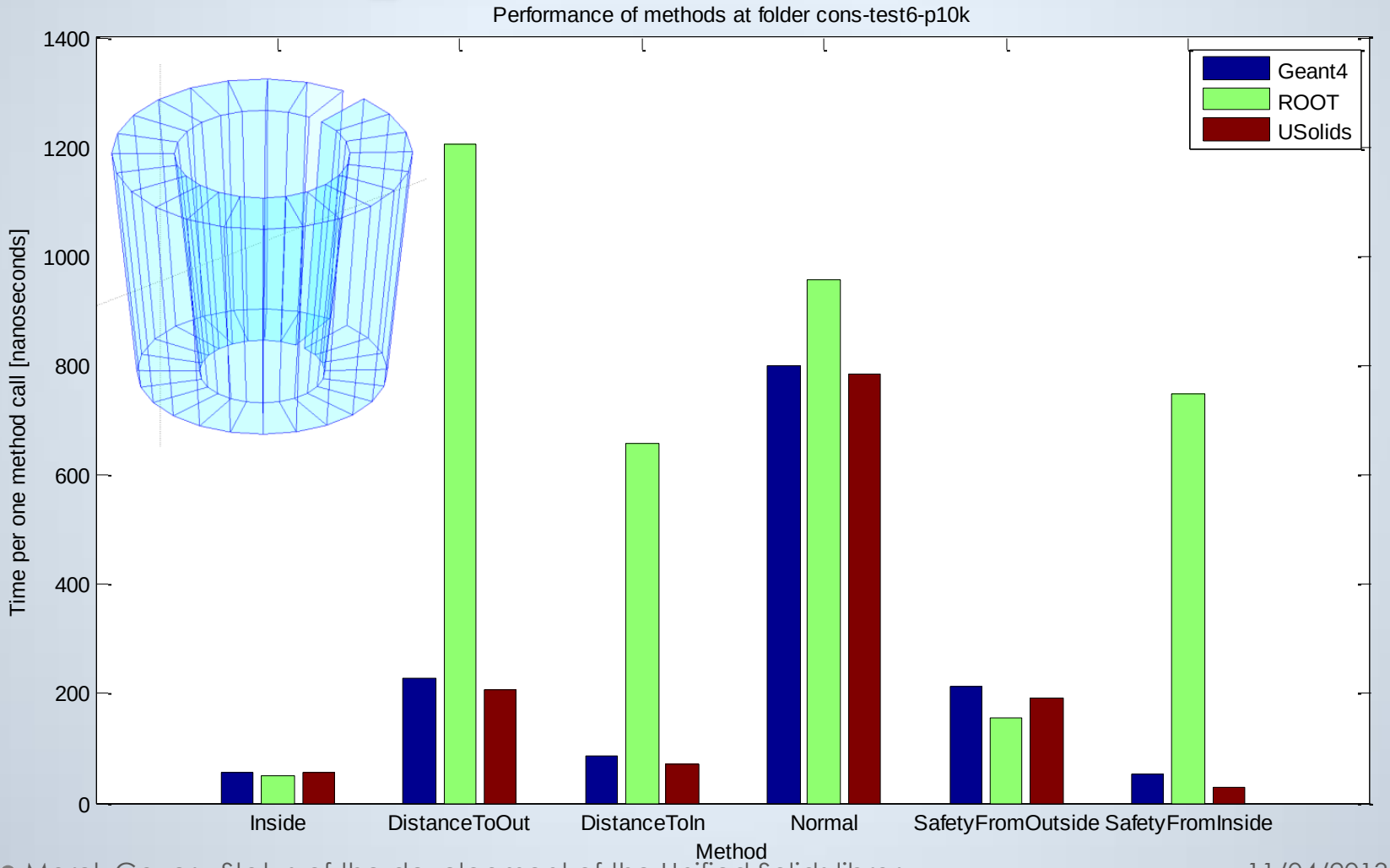
Voxels / Case	key-1.1k.gdml	foil-2.5k.gdml	foil-164k.gdml
1	**-51%** (291kB)	**-48%** (575kB)	**-50%** (32.8M)
1000	**-34%** (391kB)	**-23%** (843kb)	**-30%** (46M)
10.000	**-22%** (462kB)	**-15%** (935kb)	**-29%** (47.2M)
100.000	**+32%** (907kB)	**+45%** (1.6MB)	**-22%** (51MB)
1.000.000	**+523%** (3.7MB)	**+500%** (6.6MB)	**+6%** (70M)

Basic primitives – cone, tube, sphere …

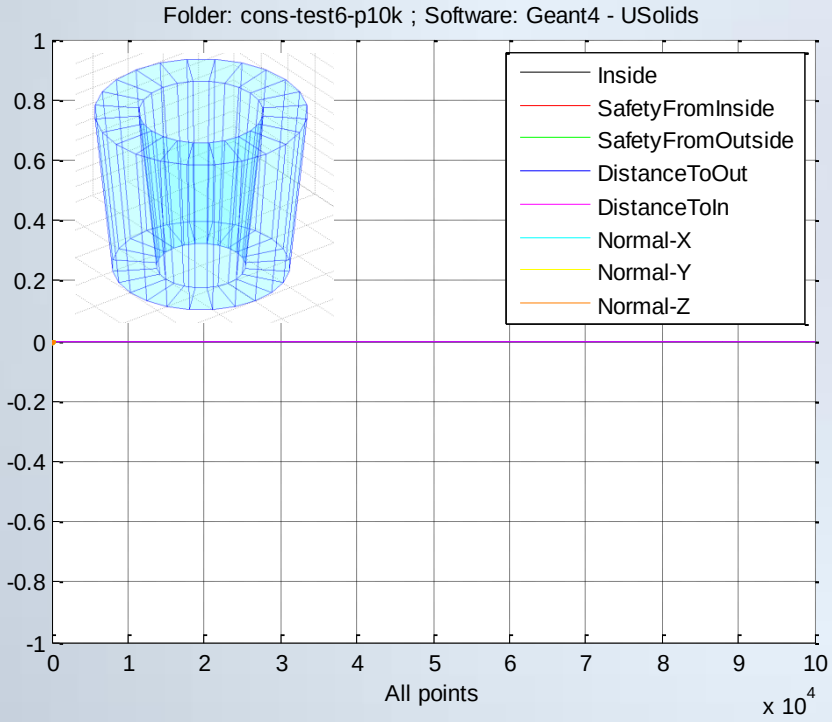
Cone performance

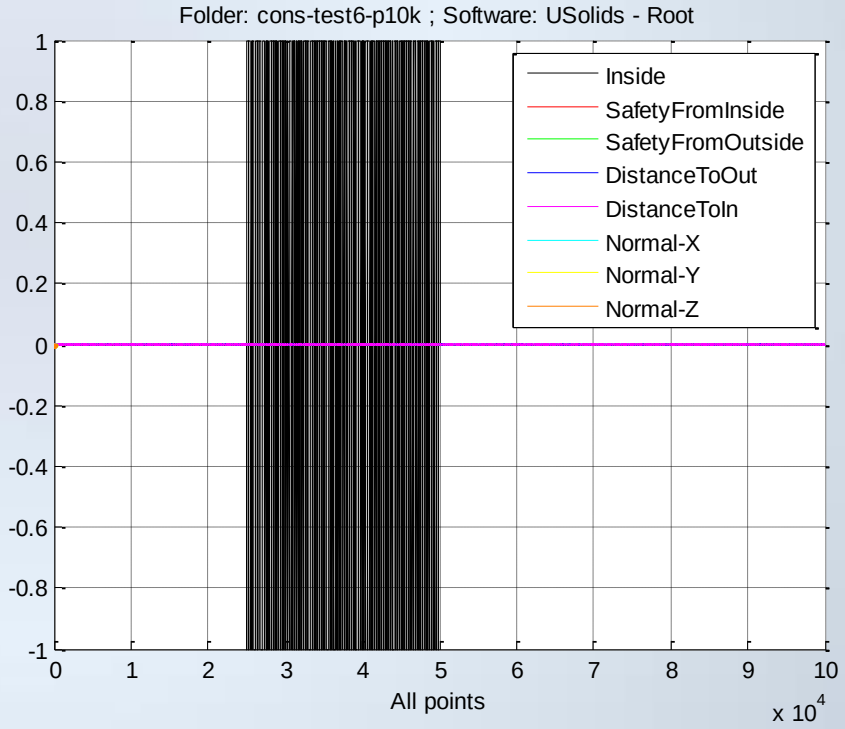


Cone with small wedge performance

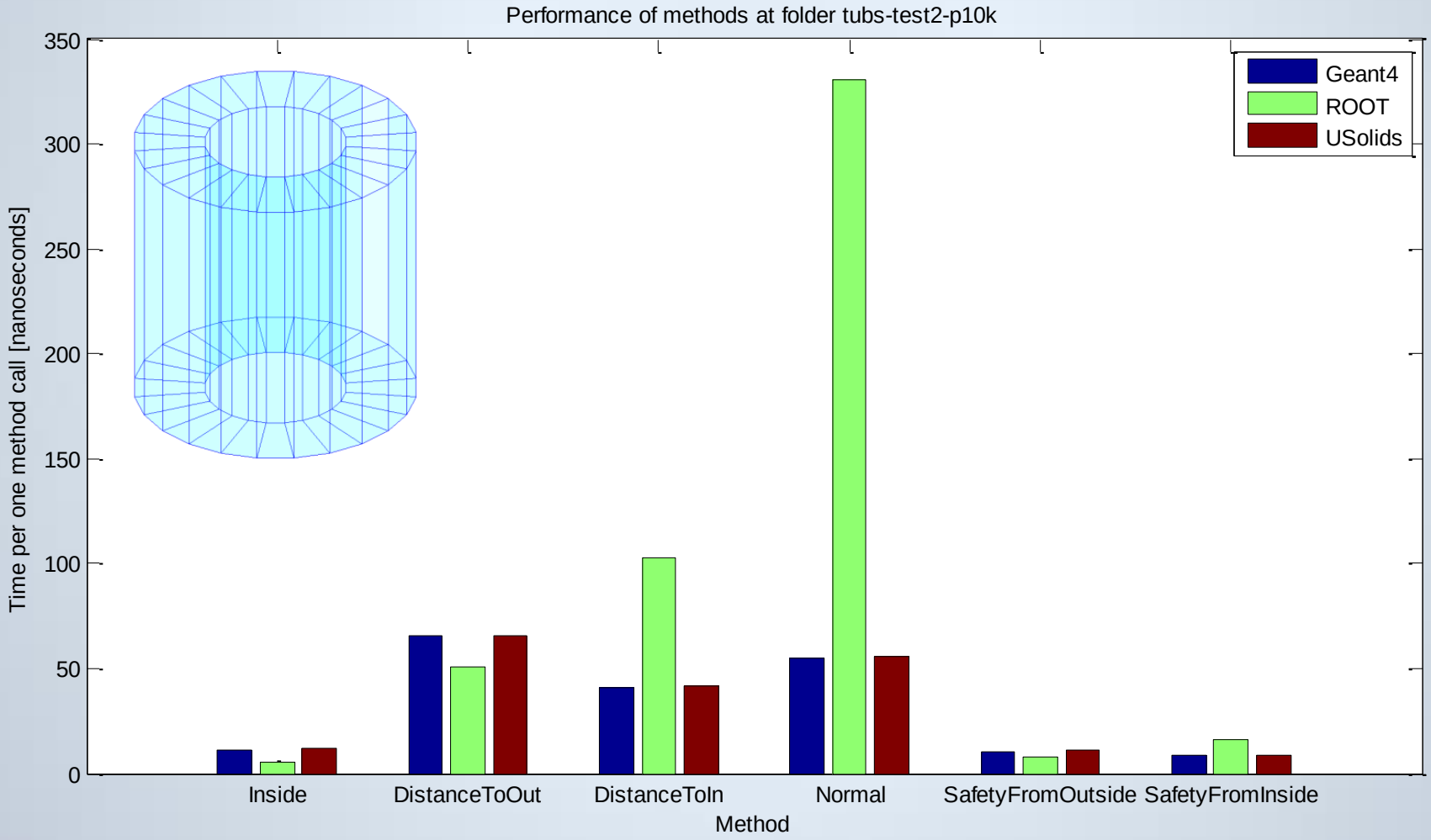


Cone values comparison

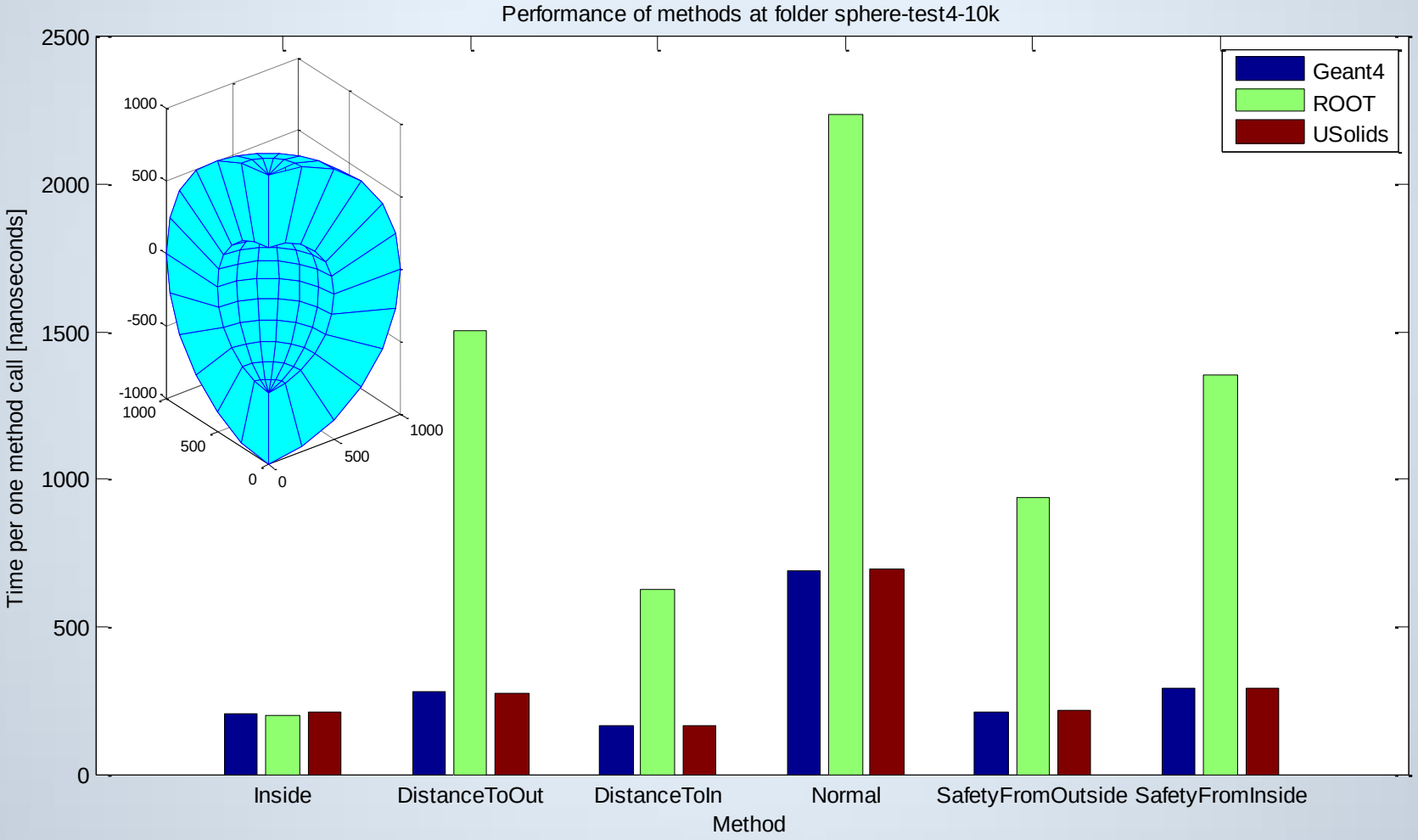




Tube performance



Spherical section performance

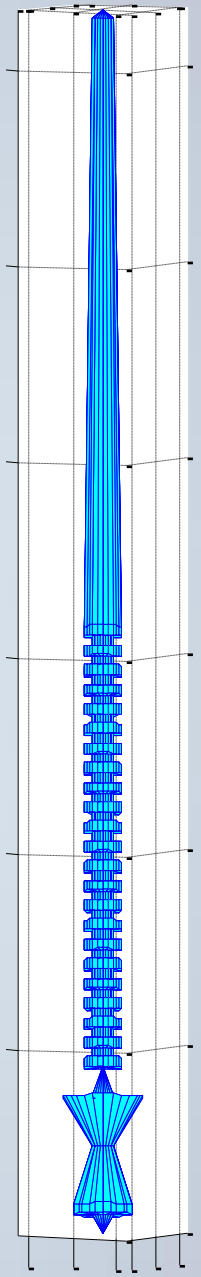


Polycone

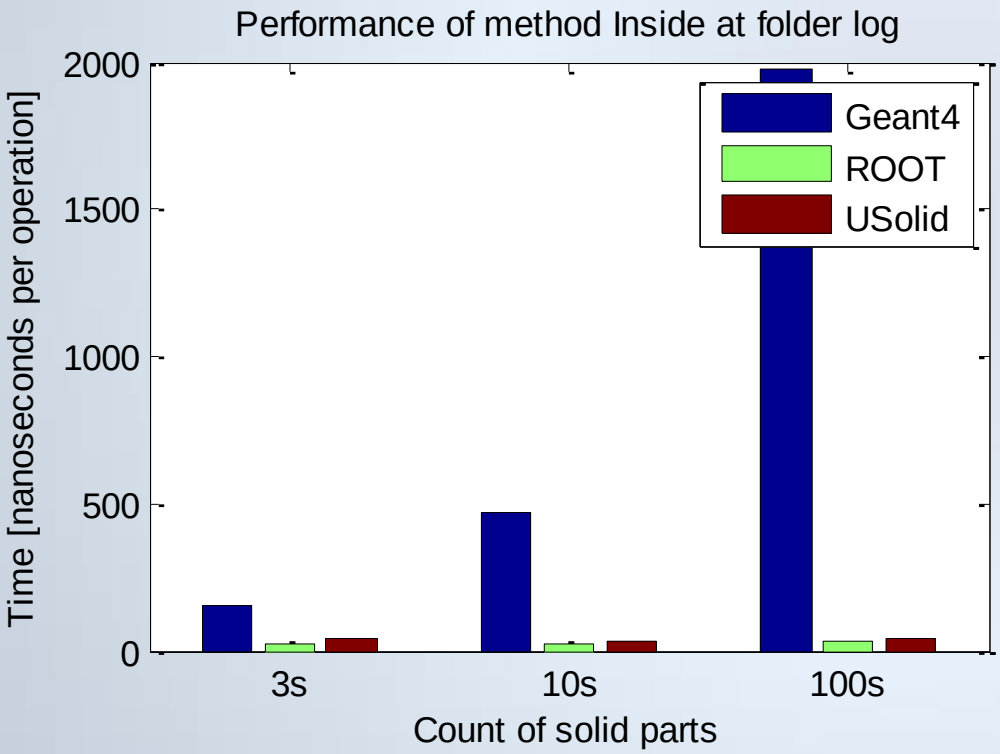
...

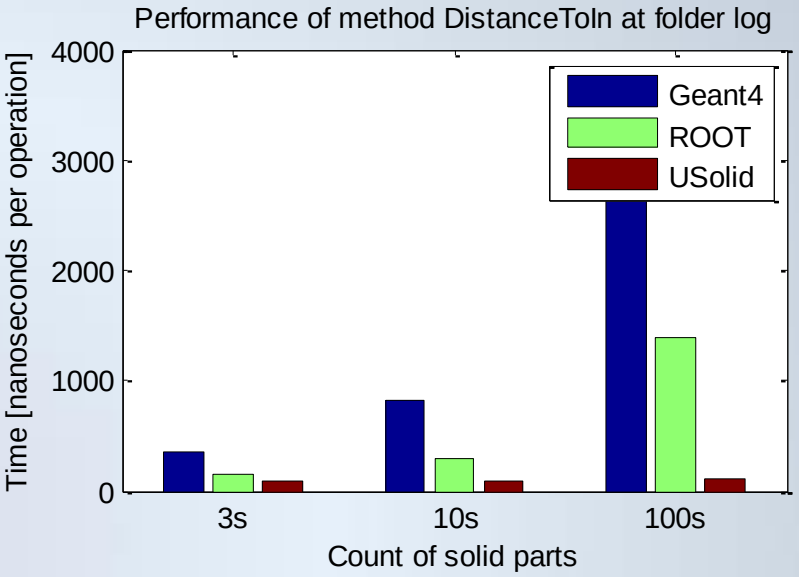
New polycone implementation

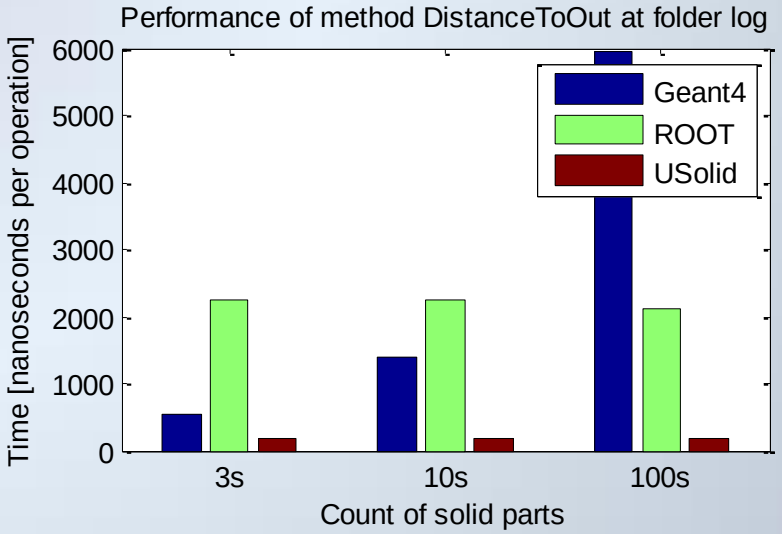
- Polycone is an important solid, heavily used in most experimental setups
- Optimization for constructs where coordinate of Z sections can only increase (big majority of real cases)
  - Careful rewrite of Polycone for these cases
- Based on composition of separate instances of cones, tubes (or their sections)
  - Providing brief, readable, clean and fast code
- Significant performance improvement over Geant4 & ROOT
- Could bring to visible CPU improvement in full simulation of complex setups like ATLAS



Polycone Scalability

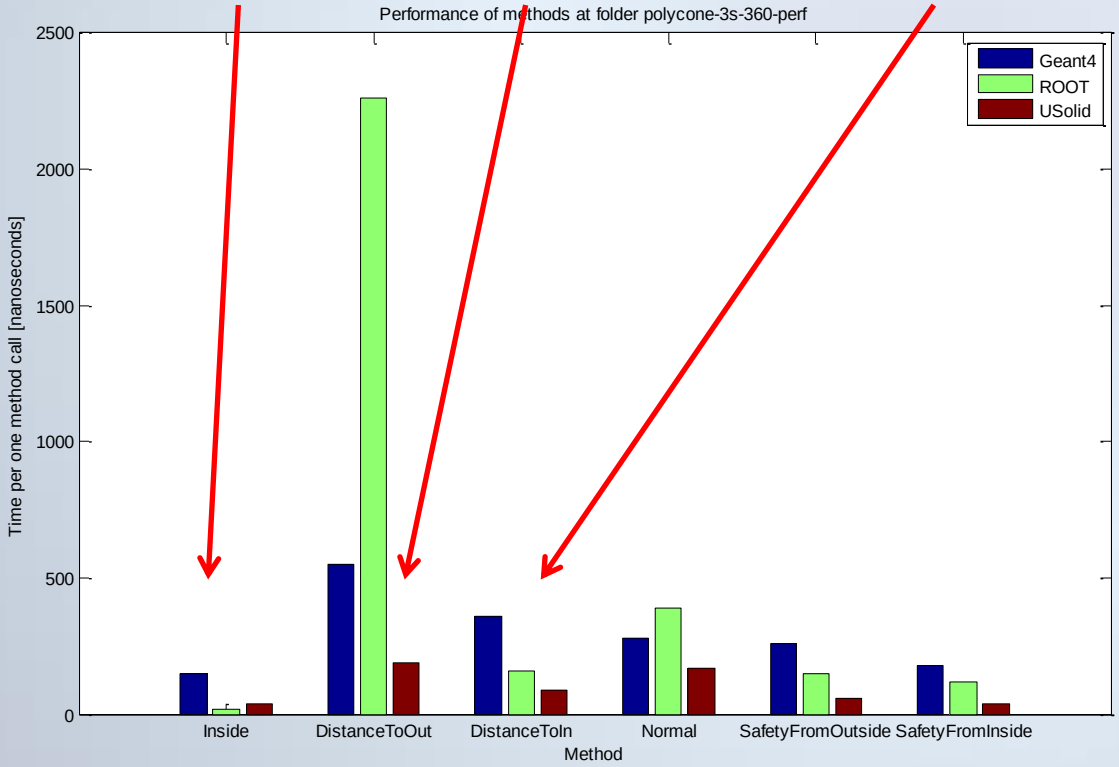


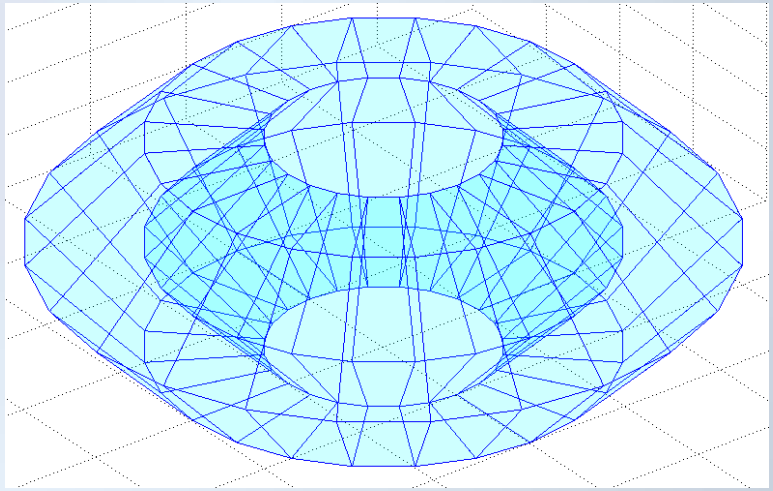




Current Polycone performance, 3 z sections

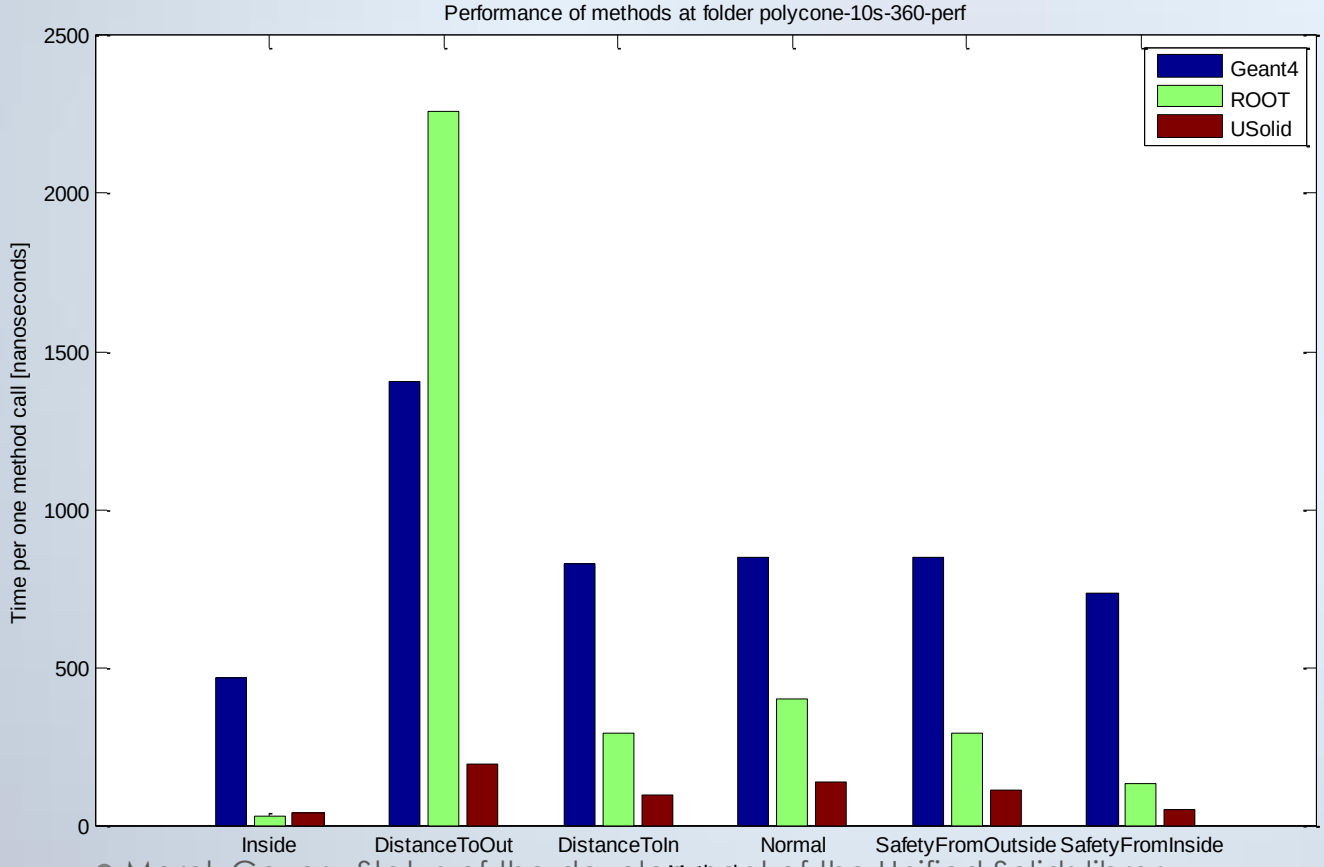
- Speedup factor **3.3x** vs. Geant4, **7.6x** vs. ROOT for most performance critical methods, i.e.: *Inside, DistanceToOut, DistanceToIn*

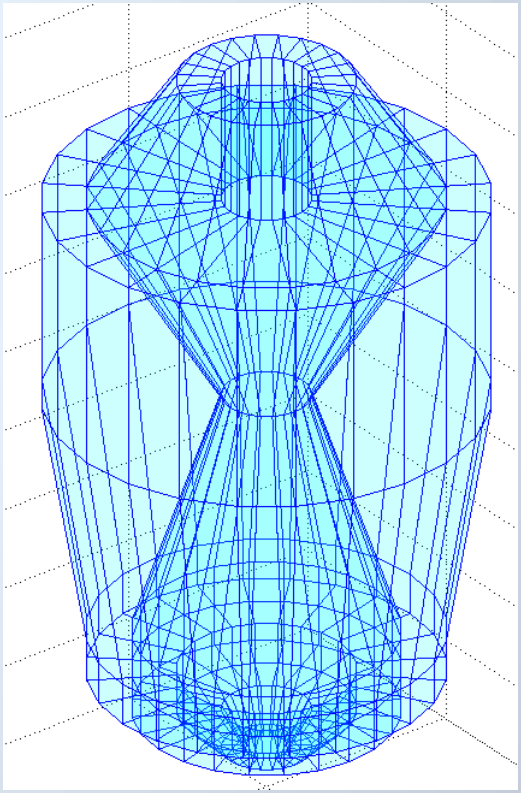




Current Polycone performance, 10 z sections

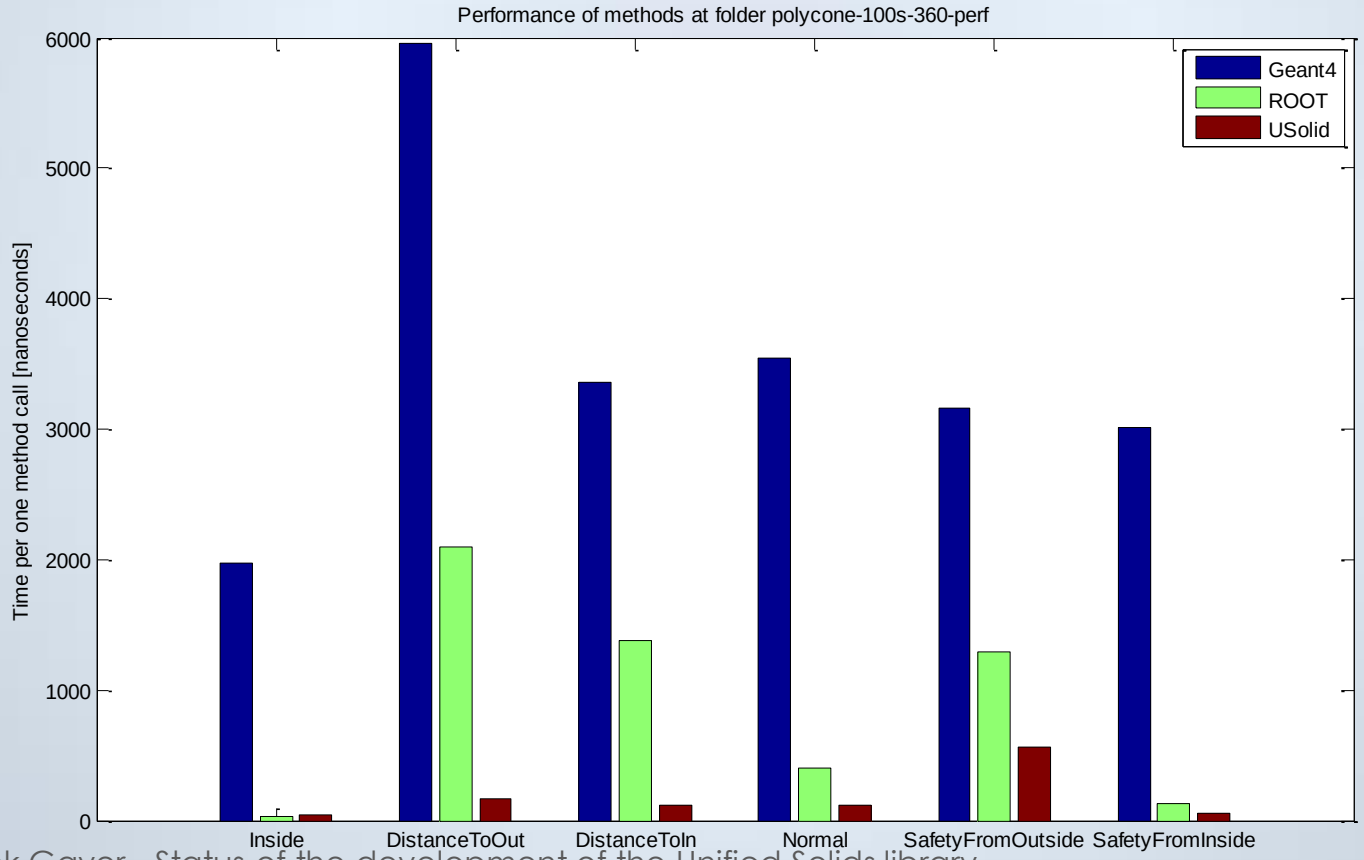
- Speedup factor **8.3x** vs. Geant4, **7.9x** vs. ROOT for most performance critical methods

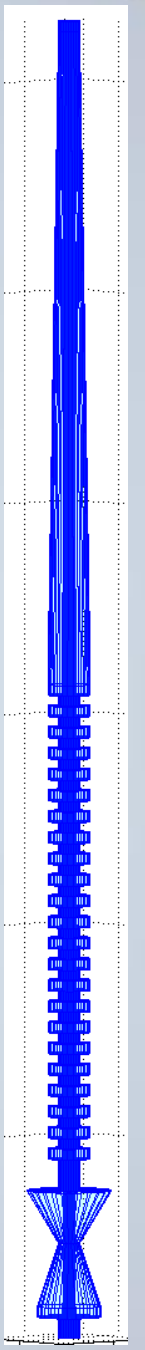




Current Polycone performance, 100 z sections

- Speedup factor **34.3x** vs. Geant4, **10.7** vs. ROOT for most performance critical methods





Status of work

- ✓ Types and Unified Solids interface are defined
- ✓ Bridge classes for comparison with Geant4 and ROOT implemented
- ✓ Testing suite defined and deployed
- ✓ Implemented 9 primitives (Box, Orb, Trapezoid, Sphere, Tube, Cone, Generic trapezoid, Tetrahedron, Arbitrary trapezoid)
- ✓ Implemented new composed solids **Multi-Union, Tessellated, Polycone** (close to be finished)

Future work plan

- Marek Gayer
  - Complete and eventually further optimize Polycone
  - Implement Polyhedra
  - Hand over the tasks to Tatiana Nikitina
- Tatiana Nikitina
  - Give priority to the remaining most critical solids and those where room for improvement can be easily identified
  - Systematically analyze and implement remaining solids in the new library

Thank you for your attention.

Questions and Answers