Unified Solids

Marek Gayer, **John Apostolakis, Gabriele Cosmo, Andrei Gheata, Jean-Marie Guyader, Tatiana Nikitina**
CERN PH/SFT

The 17th Geant4 Collaboration Meeting, Chartres, France, September 10-14, 2012

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 library of high quality implementations
  - Starting from what exists today in Geant4 and Root
  - Adopt a single type for each shape
  - Create a new Multi-Union solid
  - Make high quality, much faster Tessellated Solid
  - Aims to replace solid libraries in Geant4 and Root
  - Allowing to reach complete conformance to GDML solids schema
- Create extensive testing suite

Navigation functionality and library services for each solid

- **Performance critical methods:**
  - Location of point either inside, outside or on surface
  - Shortest distance to surface for outside points
  - Shortest distance to surface for inside points
  - Distance to surface for inside points with given direction
  - Distance to surface for outside points with given direction
  - Normal vector for closest surface from given point

- **Additional methods:** Bounding Box, Capacity, Volume, Generating points on surface/edge/inside of solid, creating mesh / polyhedra for visualization

Topics presented next:

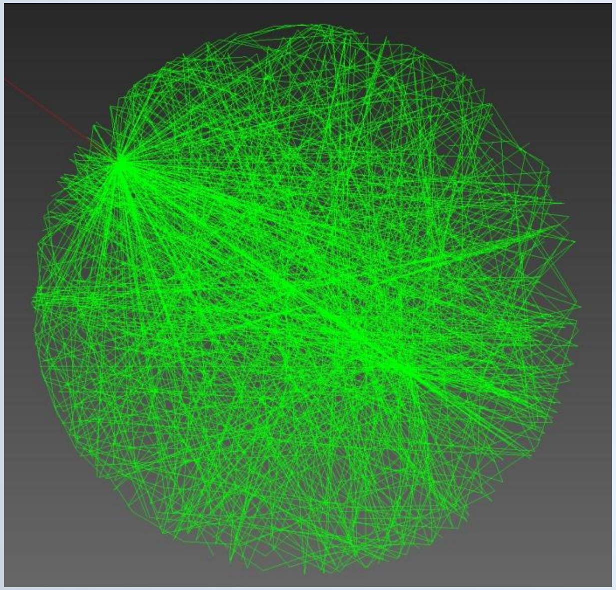
- Testing suite
- New Multi Union Solid
- Tessellated Solid made fast

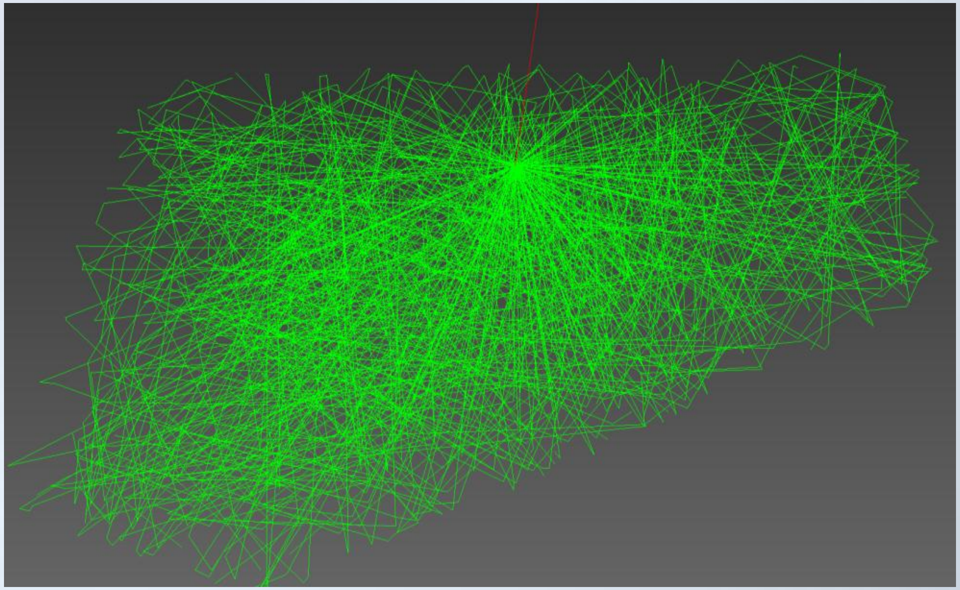
Testing Suite

- Solid Batch Test
- Optical Escape
- Data analysis and performance (SBT DAP)
- Specialized tests (e.g. quick performance scalability test for multi-union)

Optical Escape Test

- Optical photons are generated inside a solid
- Repeatedly bounce on the reflecting inner surface
- Particles must not escape the solid





Solids Batch Test (SBT)

- Points and vectors test
  - Generating groups of inside, outside and surface points
  - Testing all distance methods with numerous checks
    - E.g. for each inside random point *p*, *SafetyFromInside(p)* must be > 0
- Voxels (boxes) tests
  - Randomly sized voxels with random inside points
- Scriptable application, creates logs
- Extendible C++ framework
  - Allowing easy addition of new tests

Data Analysis and Performance (DAP)

• • •

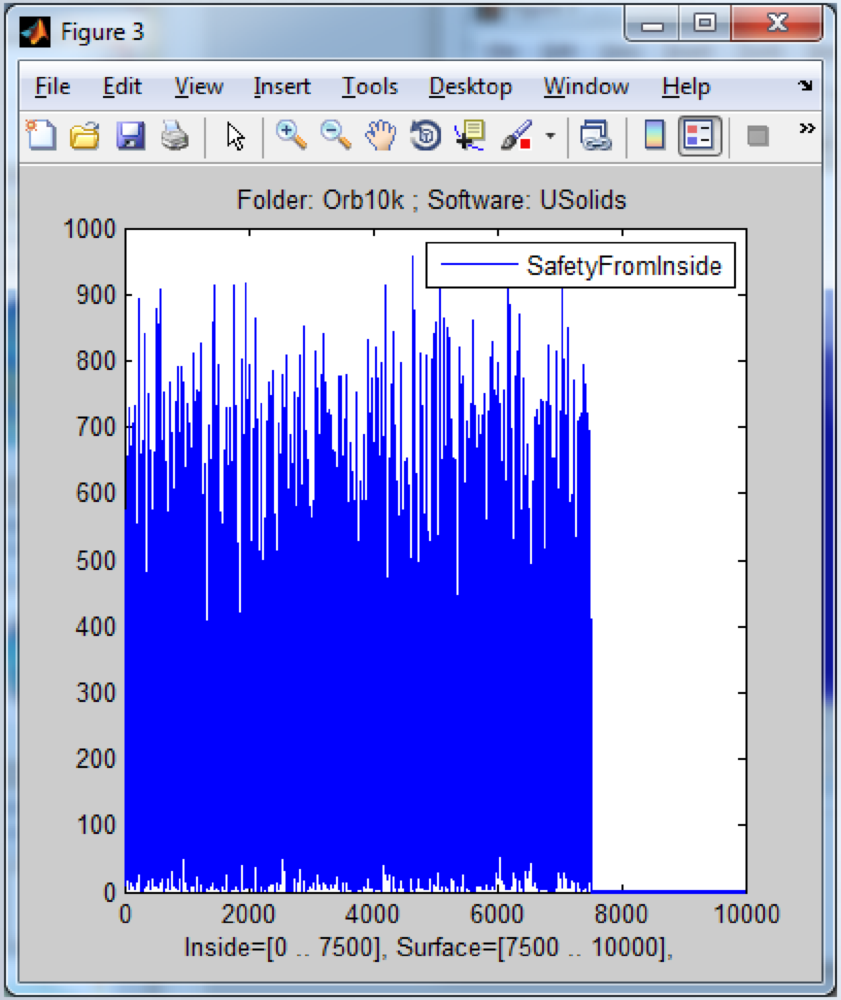
DAP features

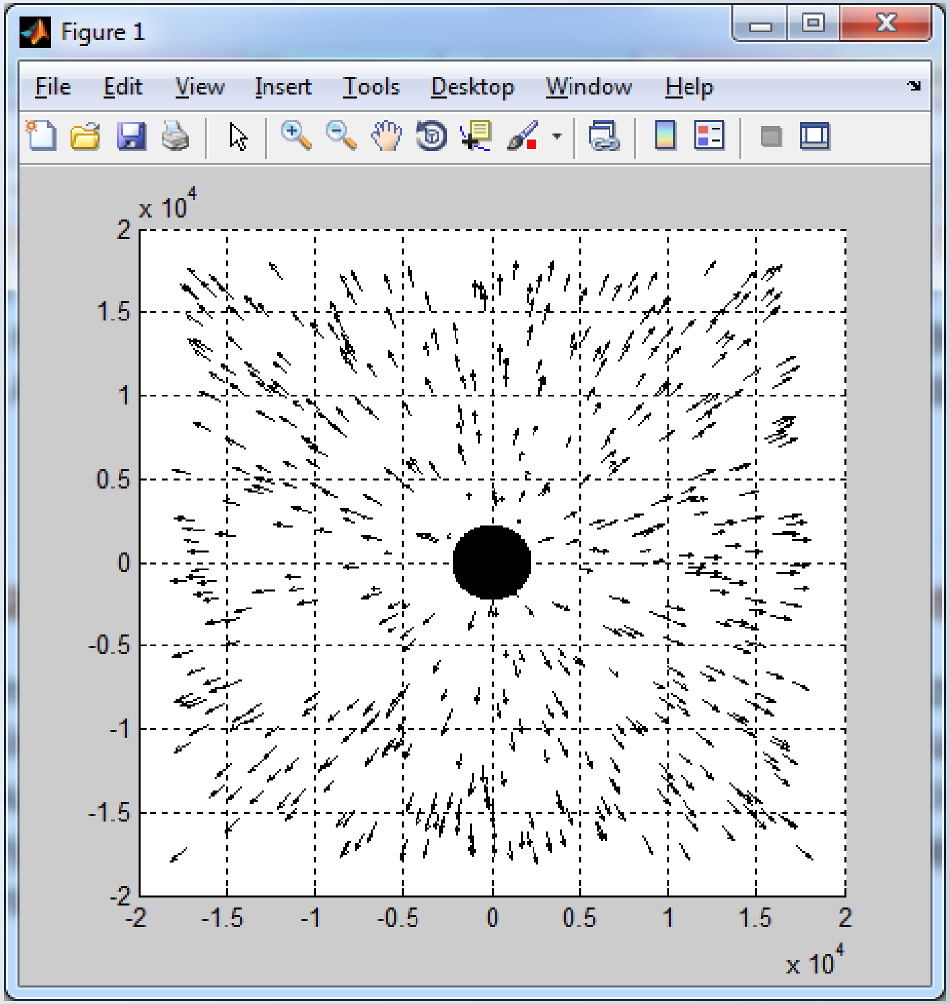
- Extension of the SBT framework
- Centred around testing Unified Solids together with existing Geant4 and Root solids
- Performance and values their differences from different codes can be compared
- Tests with pre-calculated, randomly generated sets of points and vectors
- Constrain: aim to reach similar or better performance in each method
- The core part of Unified Solids testing
- Two phases
    - Sampling phase (generation of data sets, implemented as C++ app.)
    - Support for batch scripting
        - Detailed configuration of conditions in the tests
        - Invoking several tests sequentially
    - Analysis phase (data post-processing, implemented as **MATLAB** scripts)
- Portable: Windows, Linux, Mac

DAP - Analysis phase

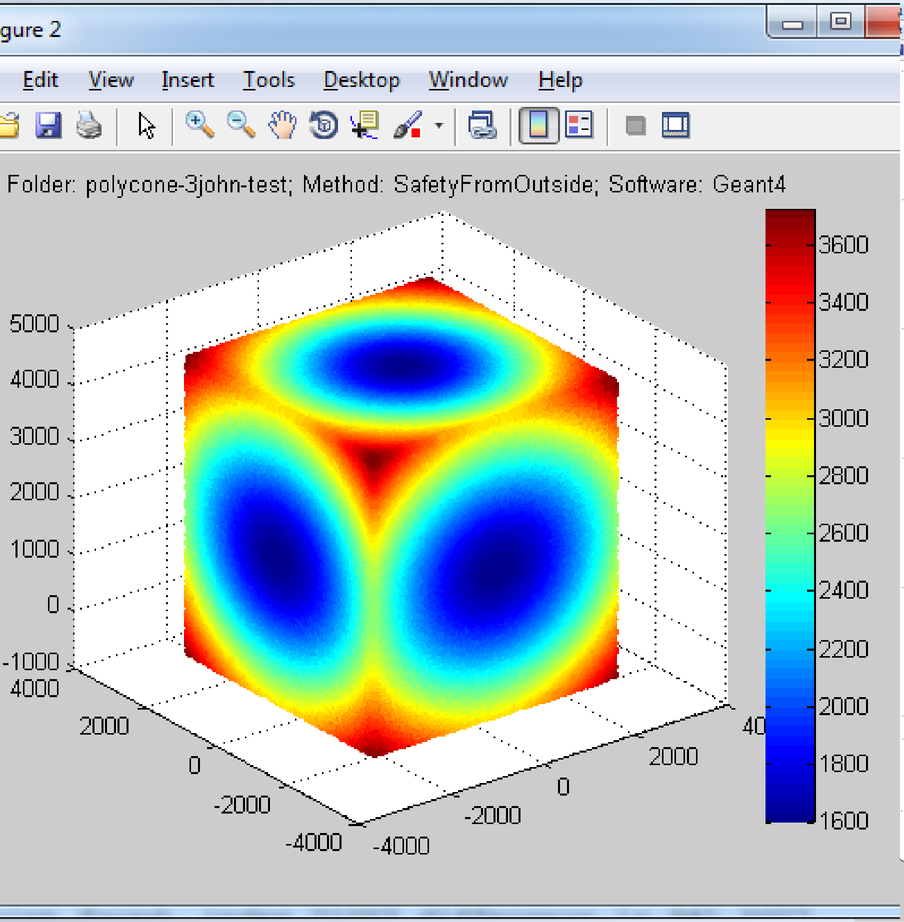
- Visualization of scalar and vector data sets and shapes
- Visual analysis of differences
- Graphs with comparison of performance and scalability
- Inspection of values and differences of data sets

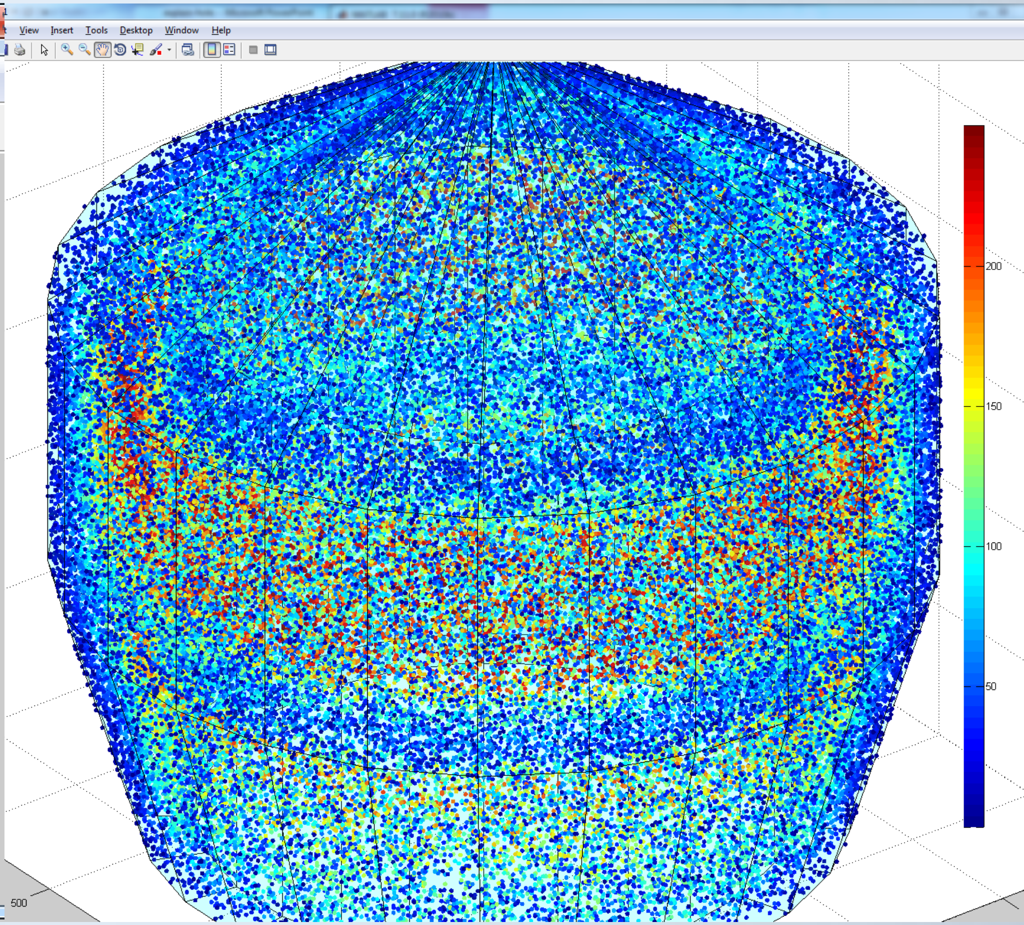
Visualization of scalar and vector data sets



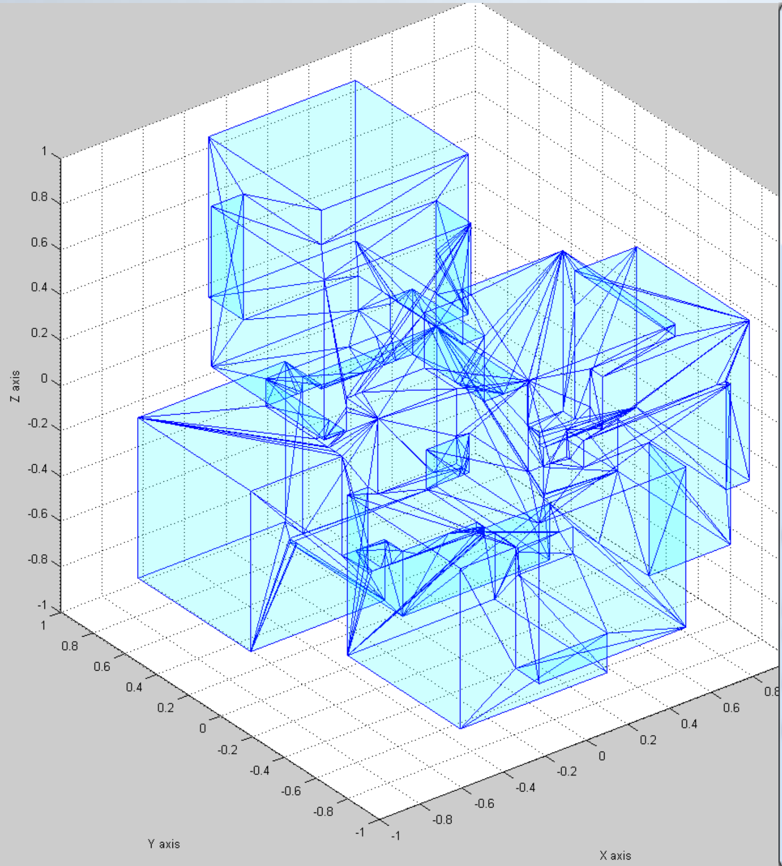


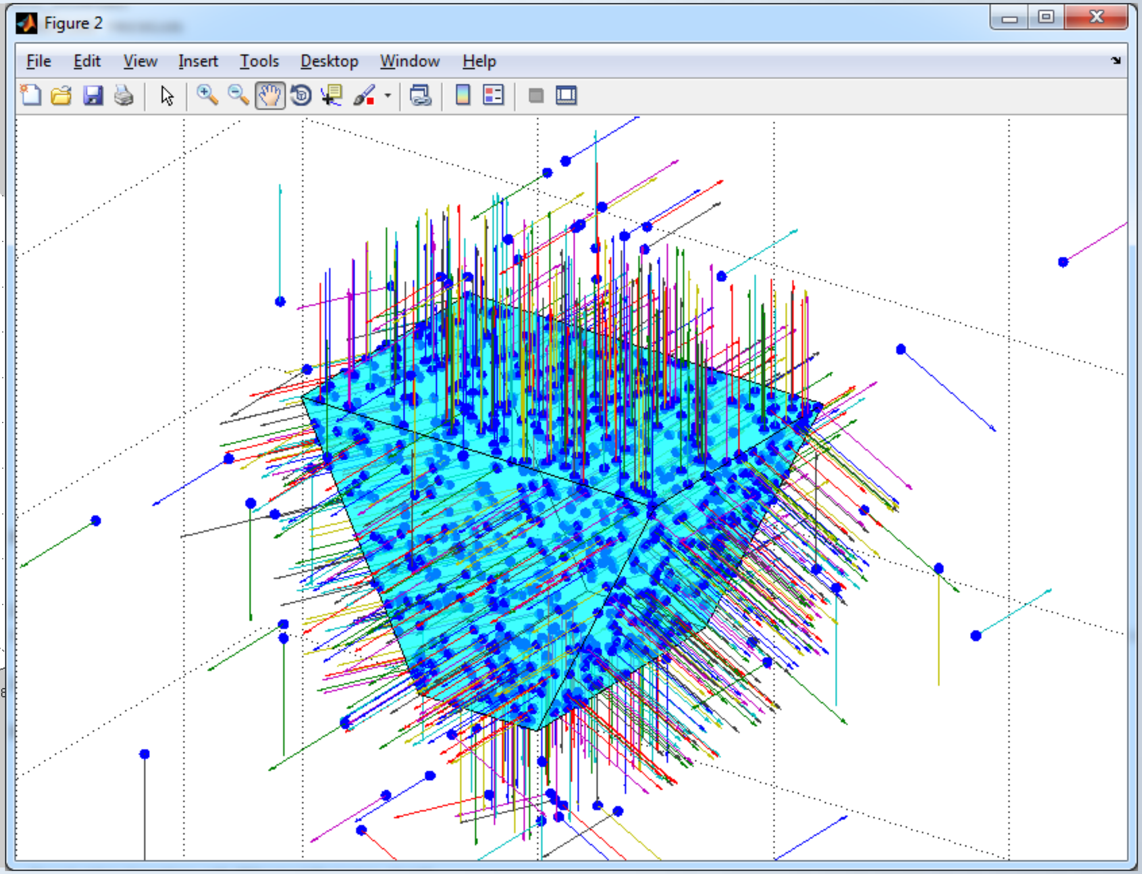
3D plots allowing to overview data sets



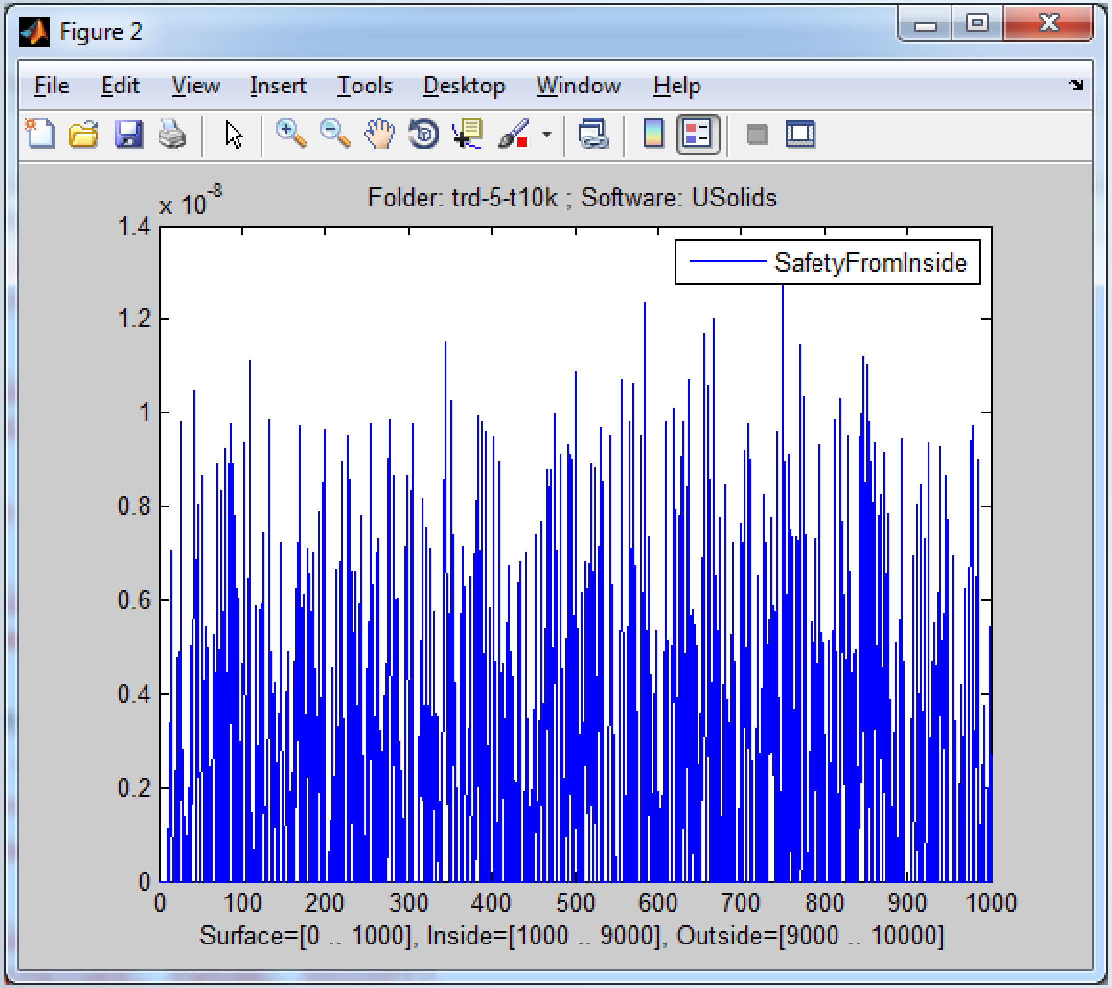


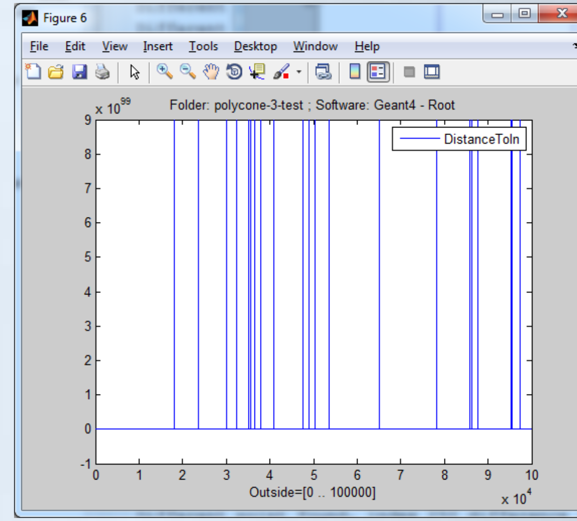
3D visualization of investigated shapes

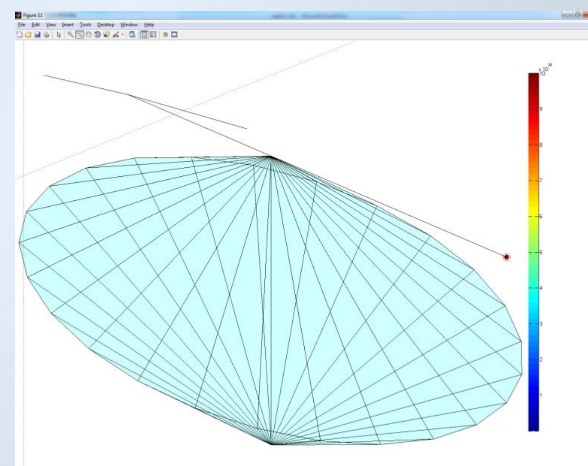




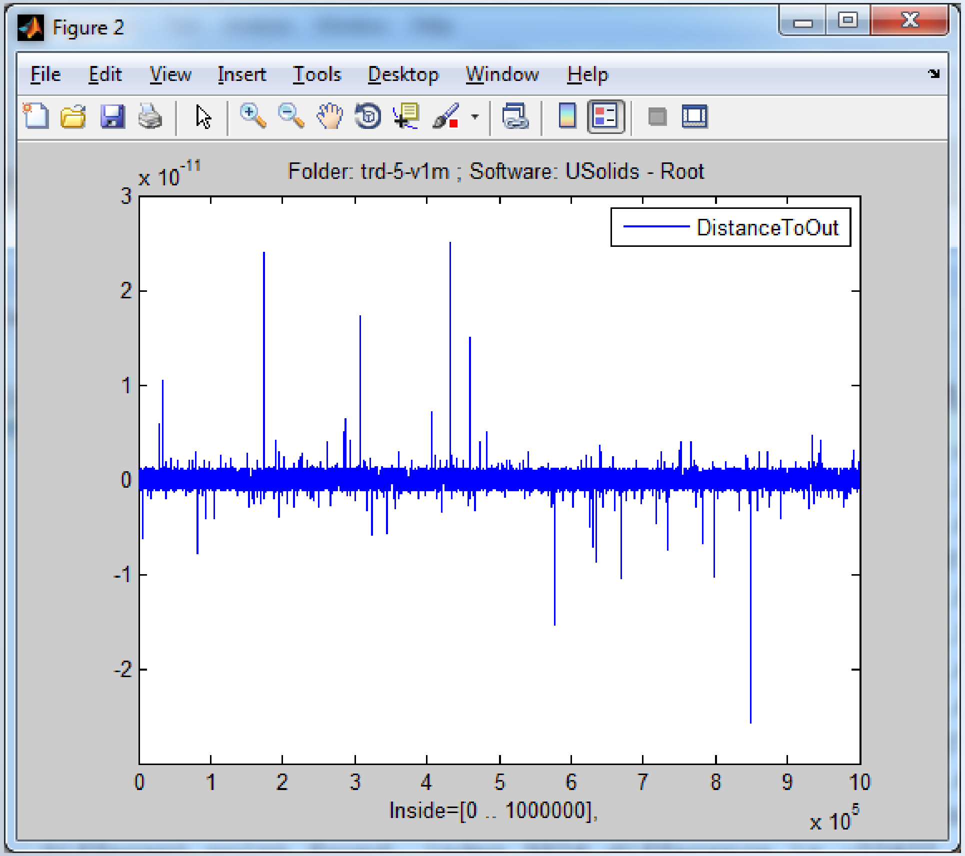
Support for regions of data, focusing on sub-parts

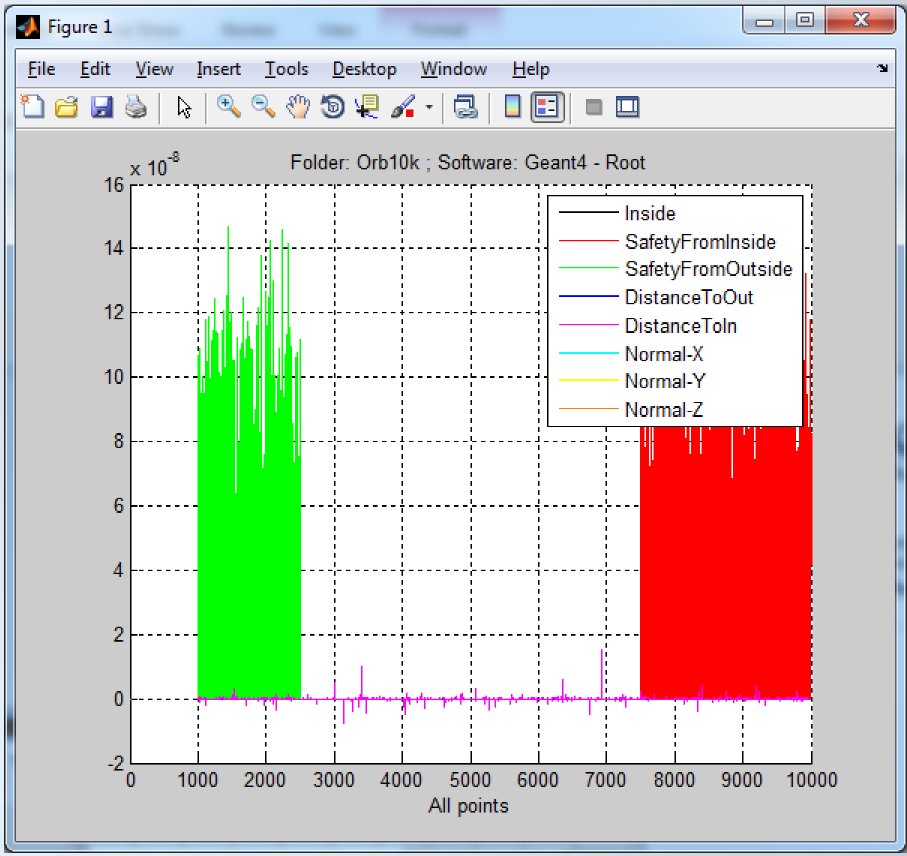




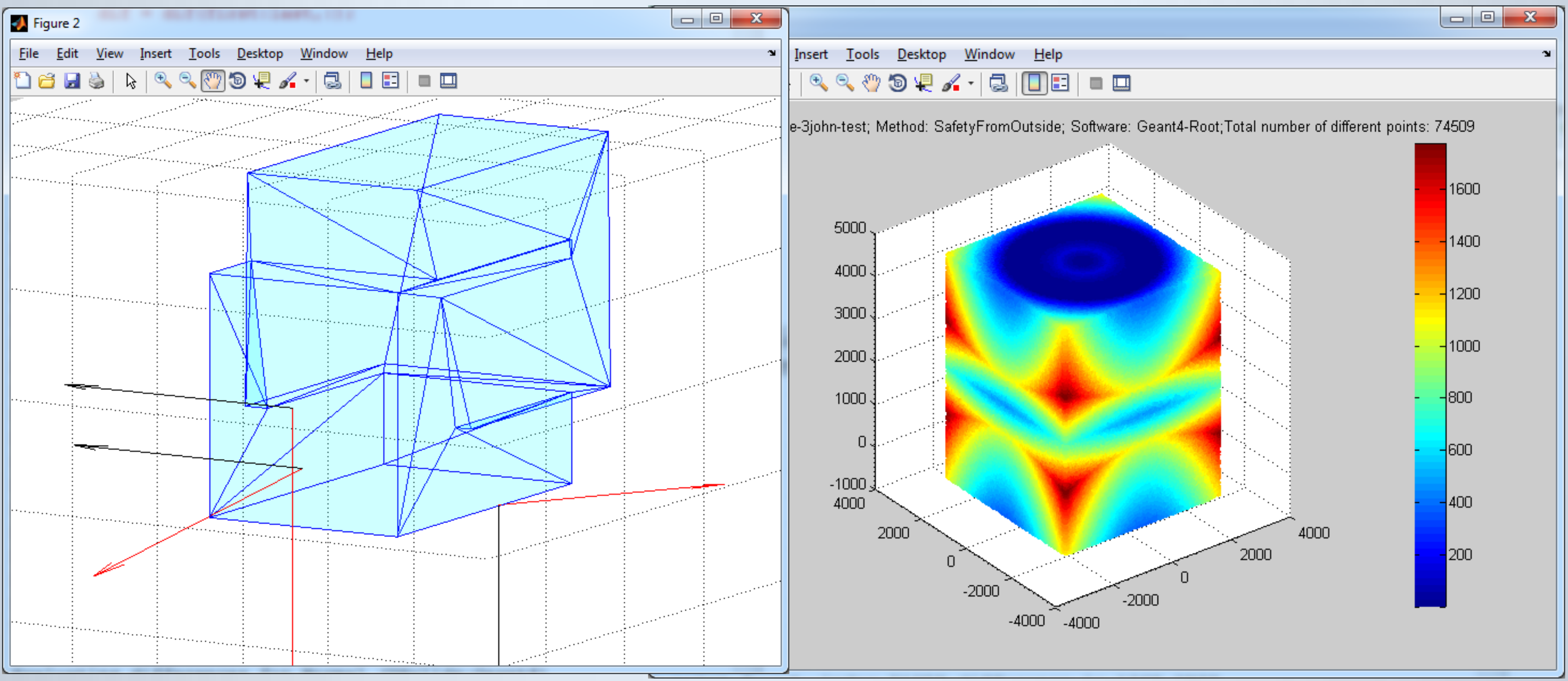


Visual analysis of differences

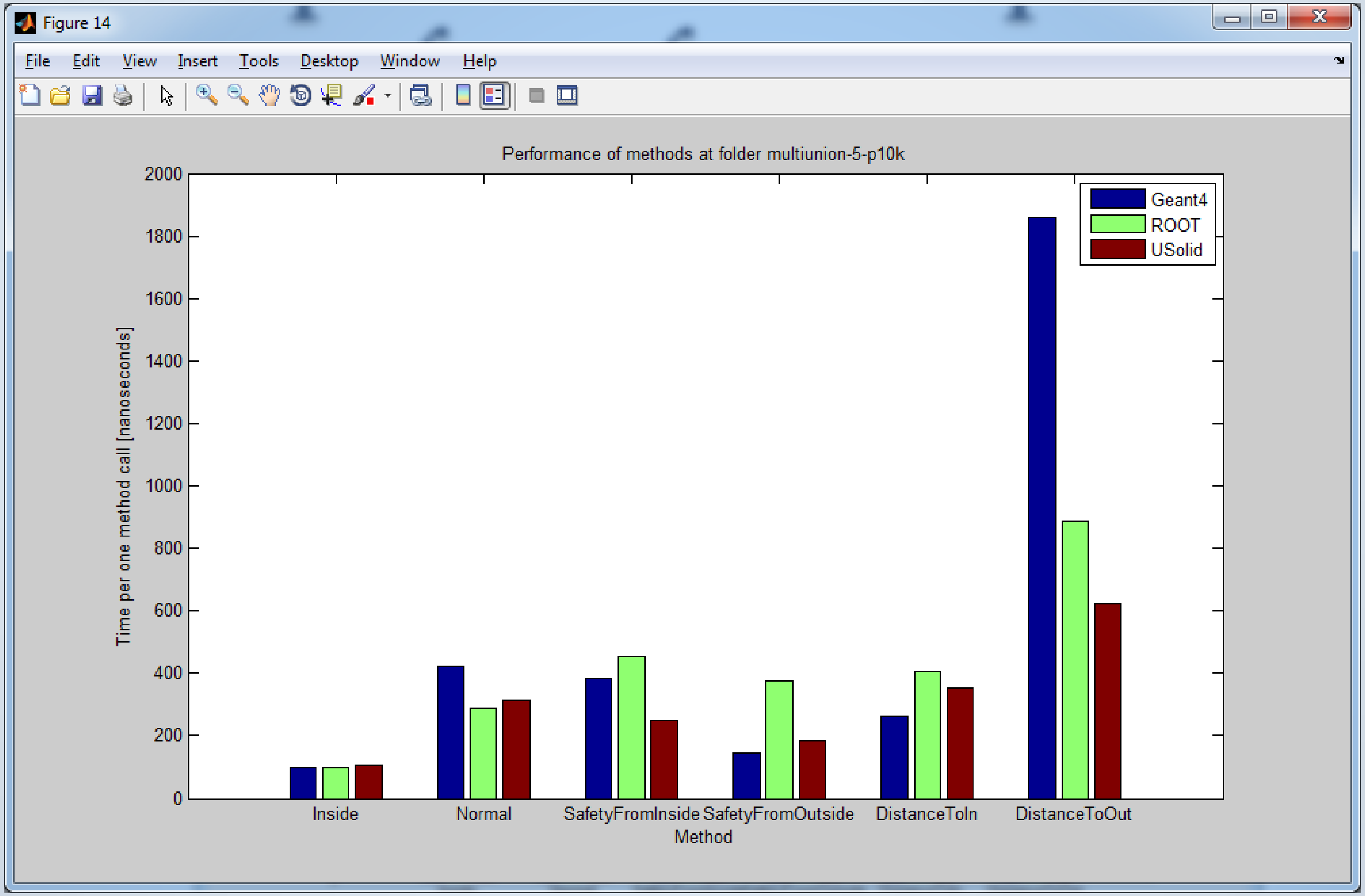




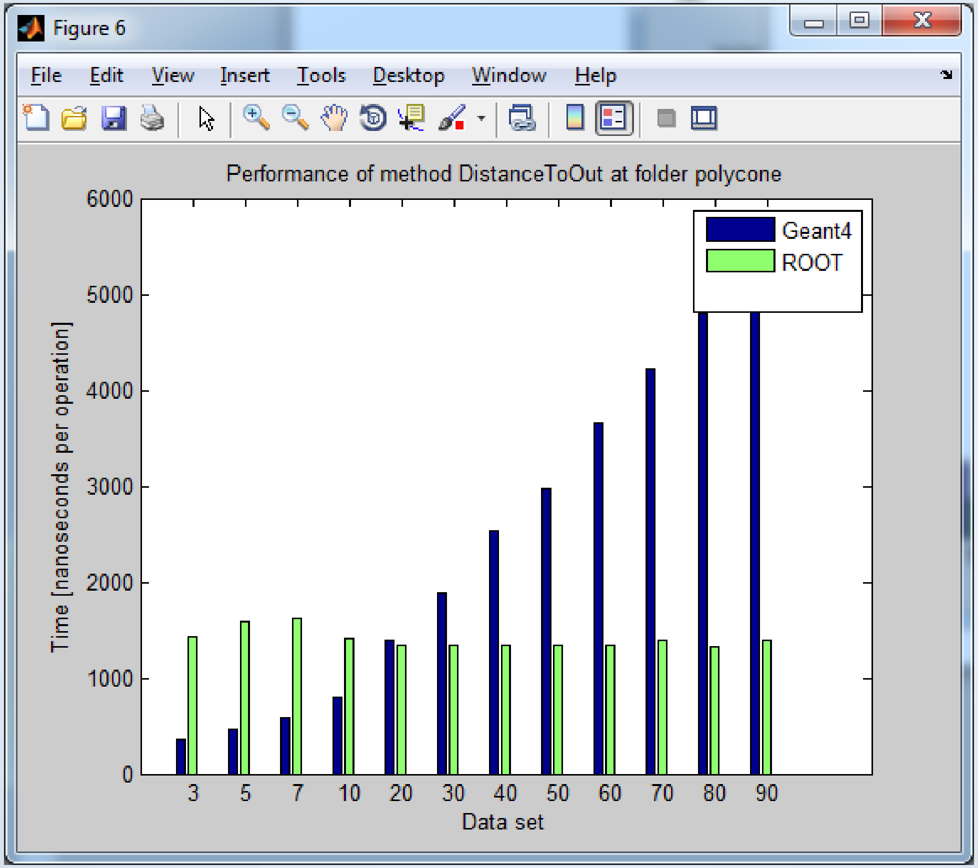
Visual analysis of differences in 3D



Graphs with comparison of performance

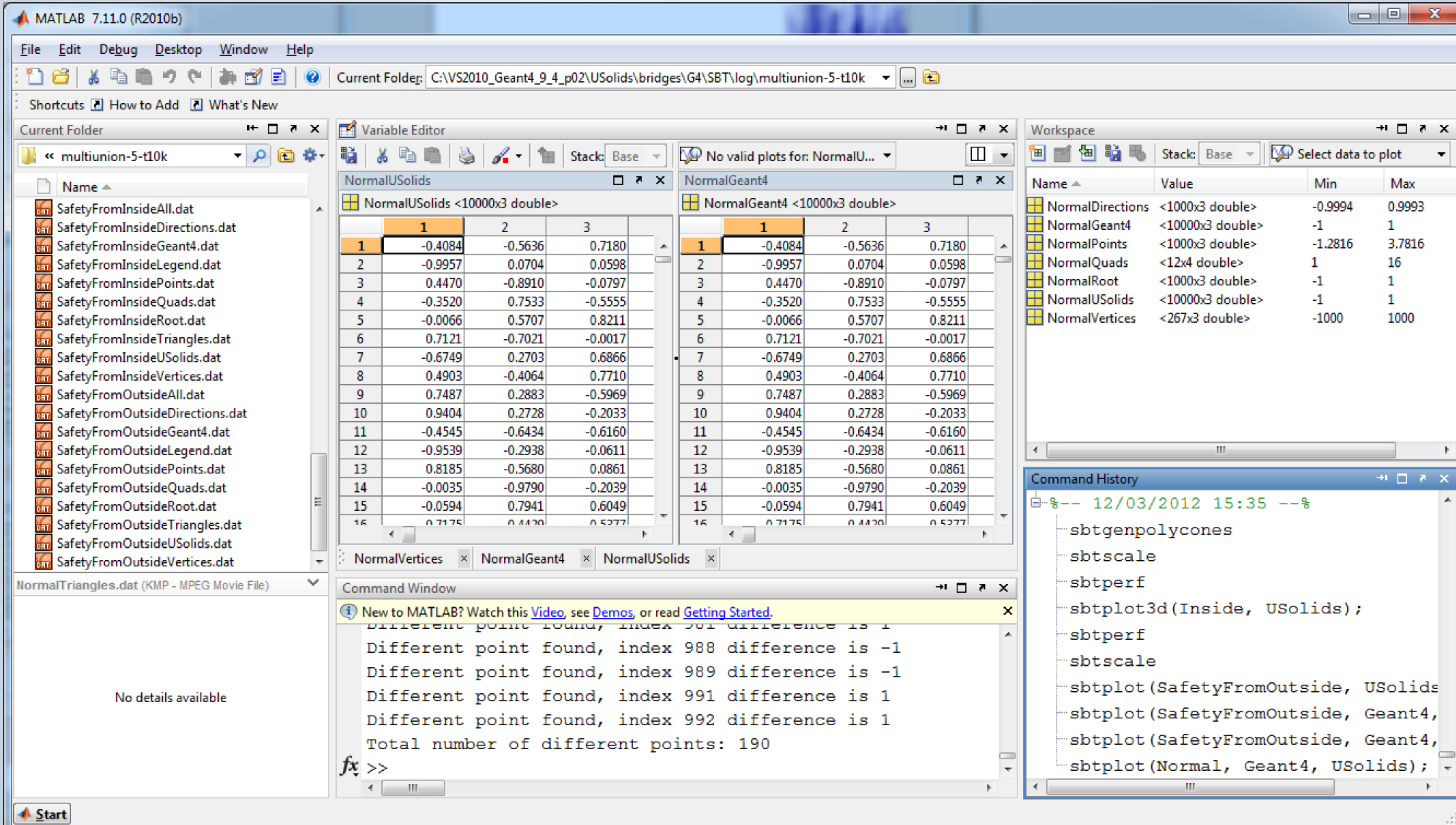


Visualization of scalability performance for specific solids



Number of z sections ->

Inspection of values and differences of scalar and vector data sets

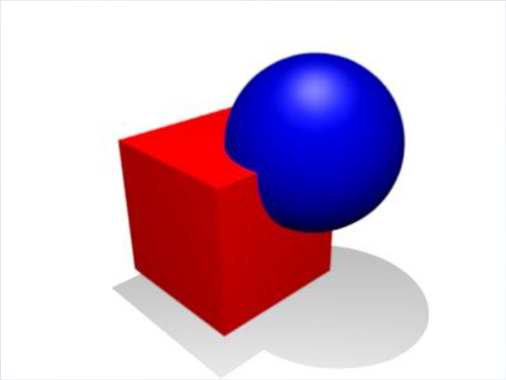


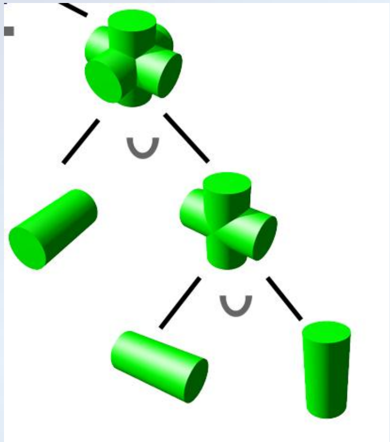
New Multi-Union solid

• • •

Boolean Union solids

- Existing CSG Boolean solids (Root and Geant4) 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

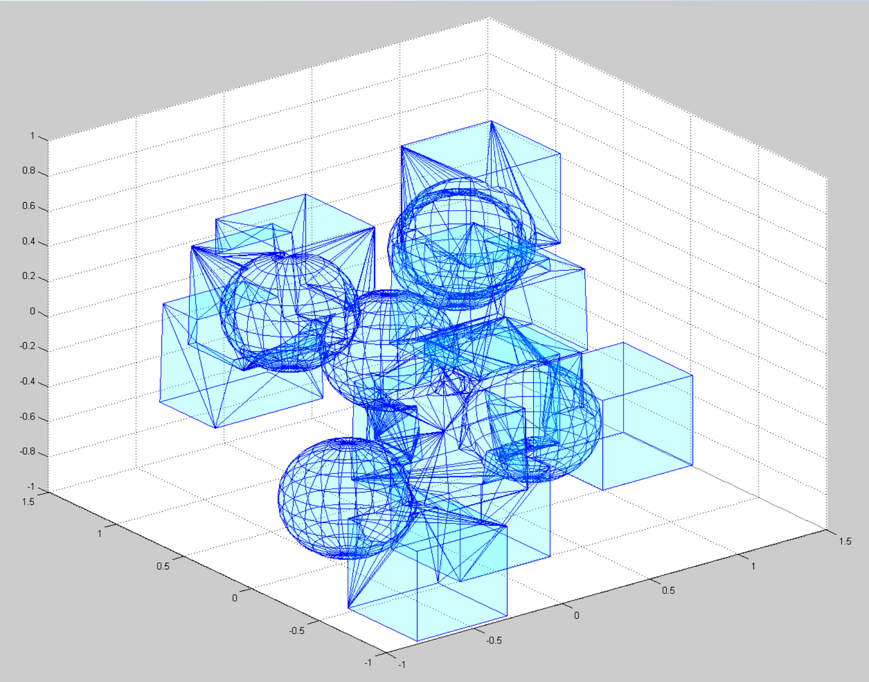




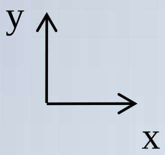
Boolean Union solid:
is composite of two solids, either primitive or Boolean

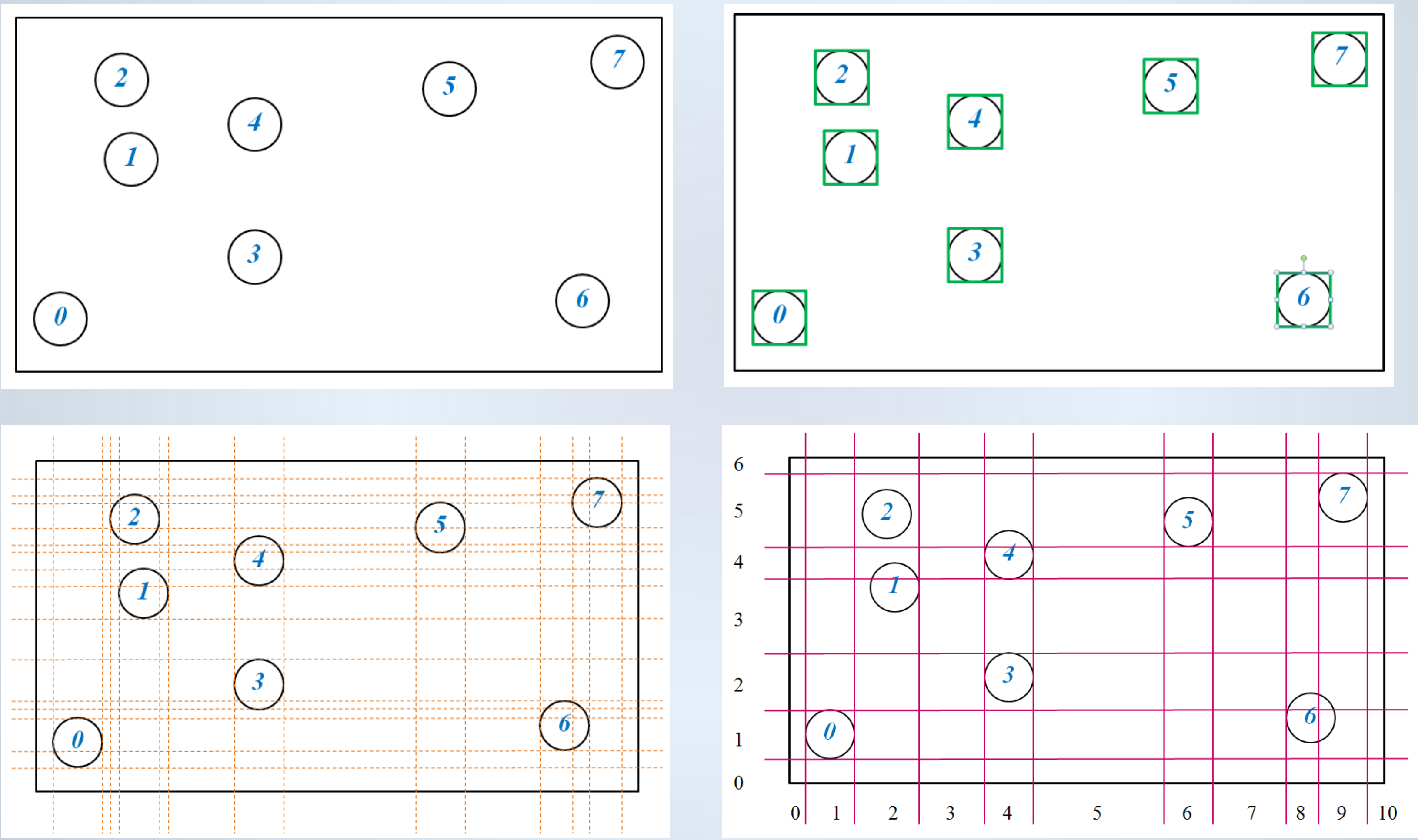
Multi-Union solid

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

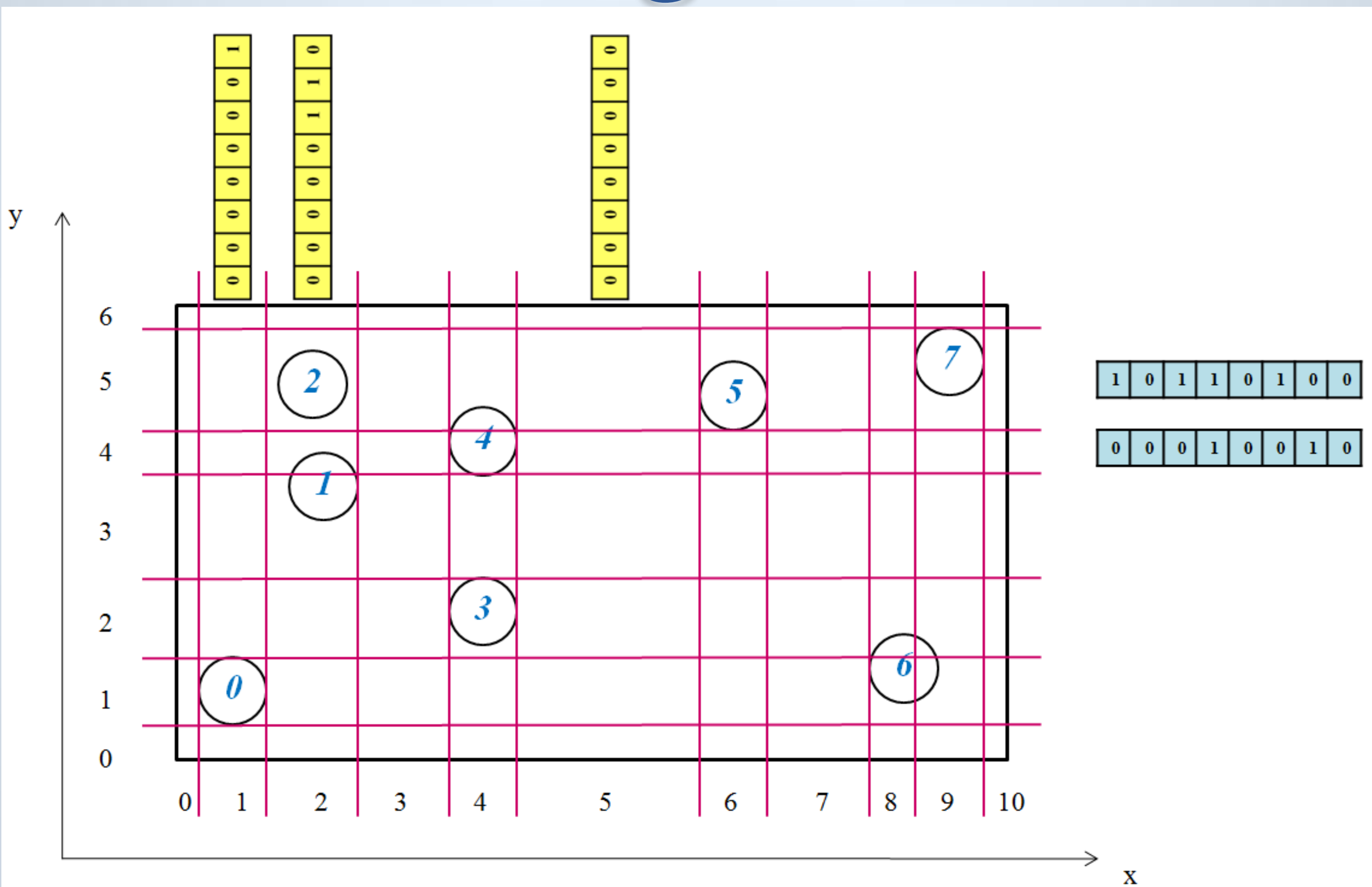


1. Create voxel space (2D simplification)

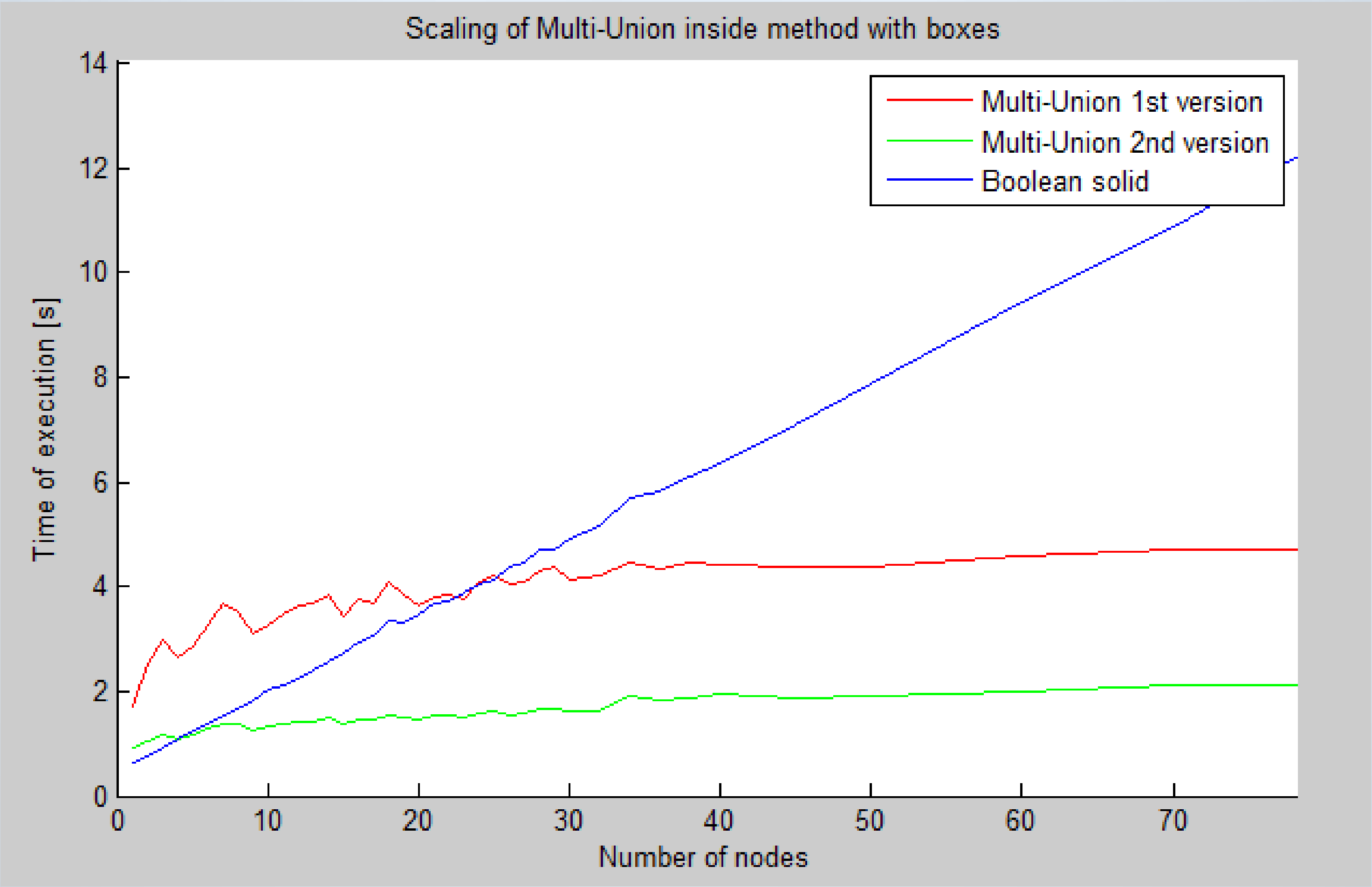




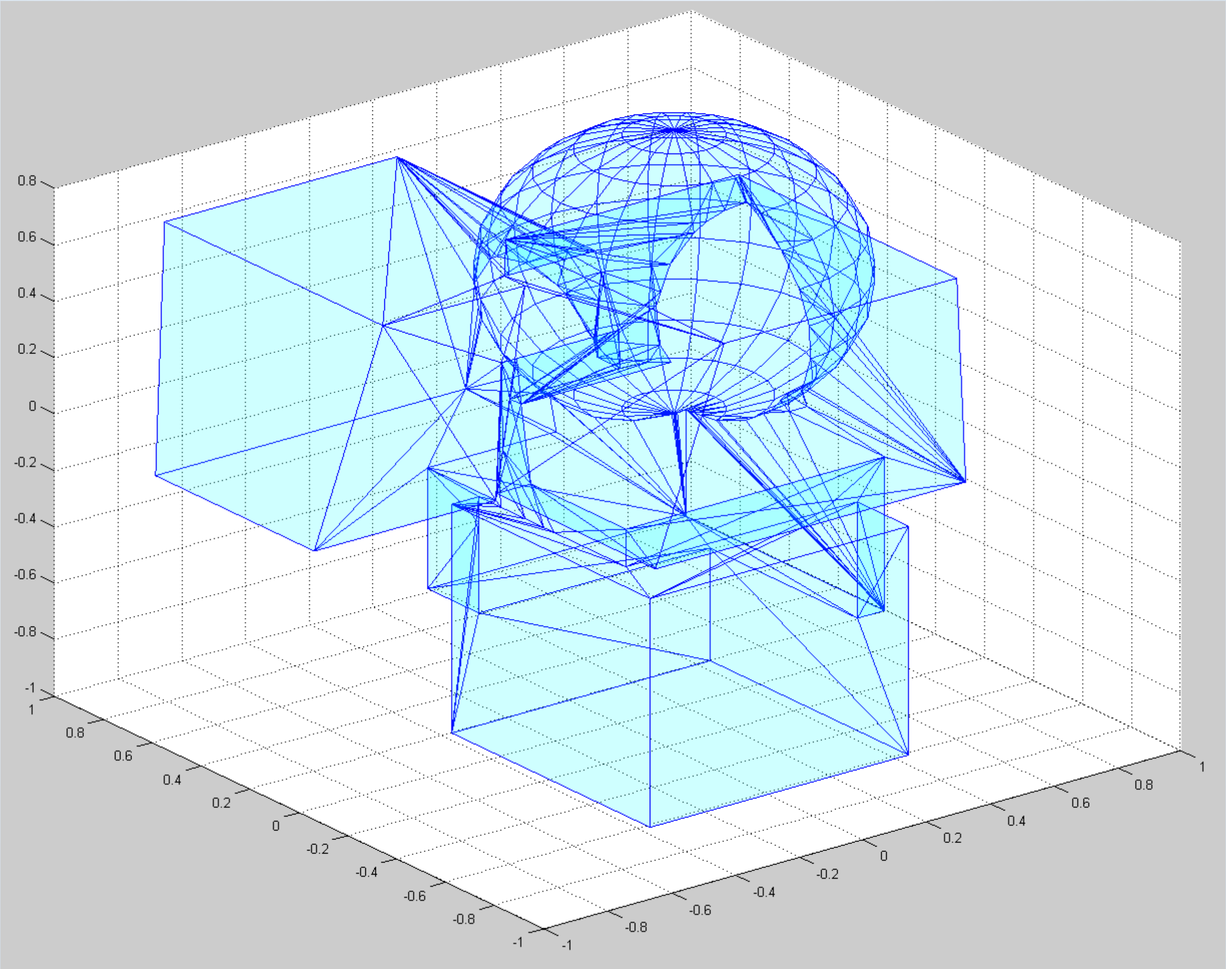
2. Usage of bit masks for storing voxels



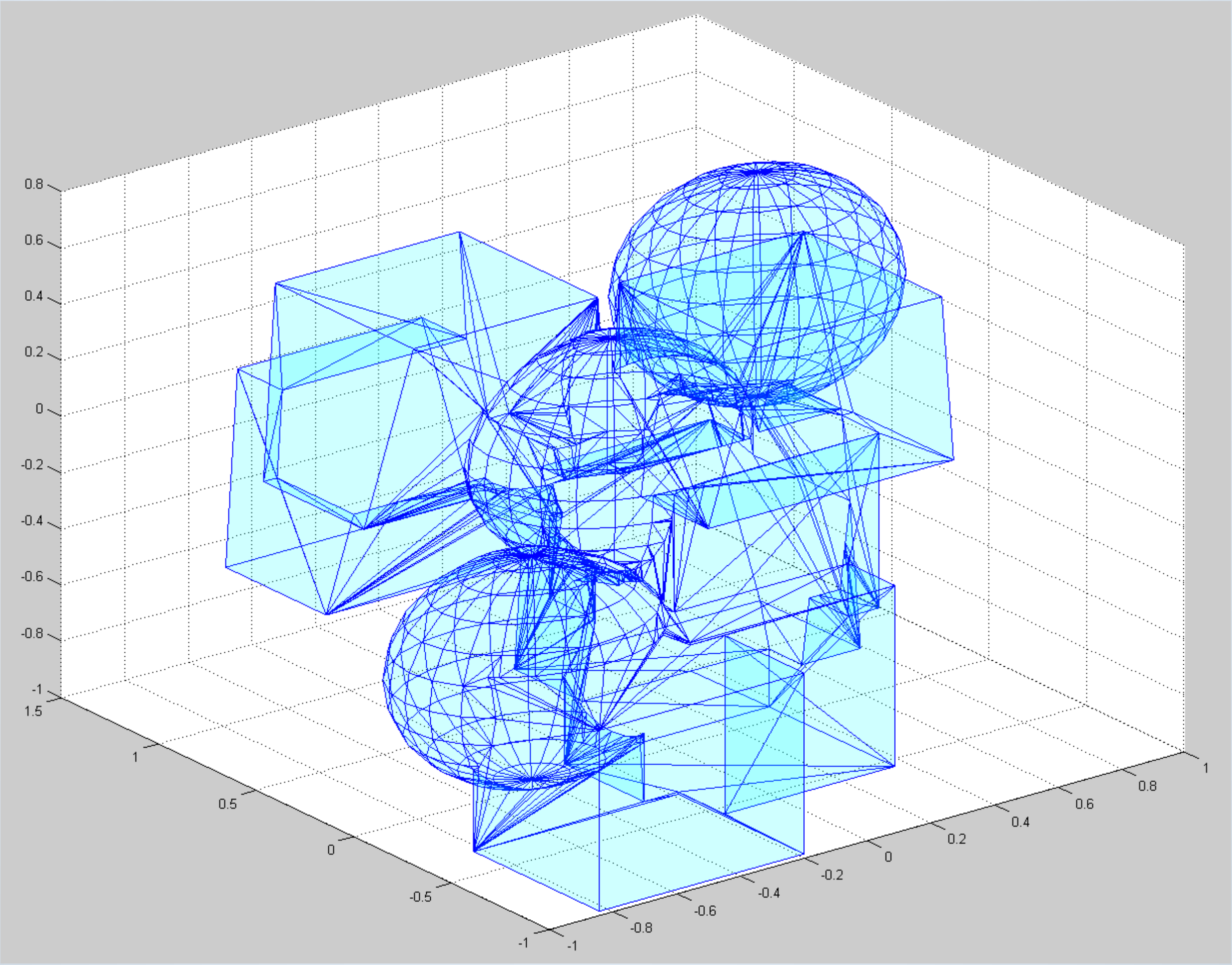
Scaling of Multi-Union vs. Boolean solid



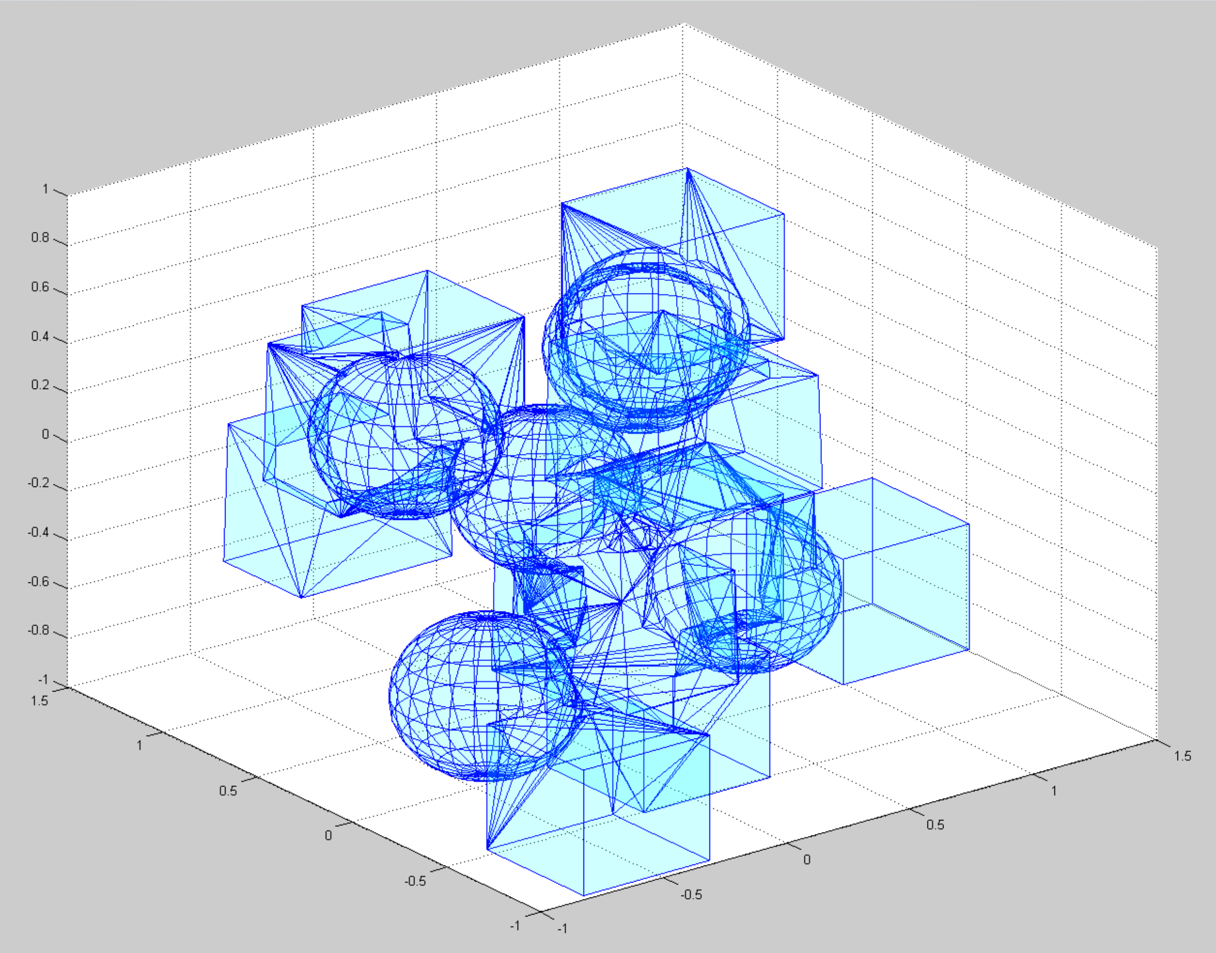
Test union solids for scalability measurements



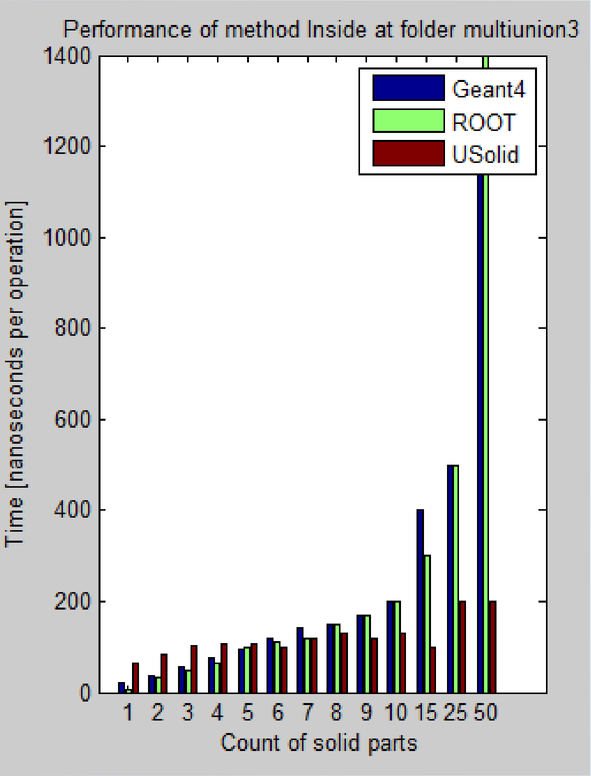
Test union solids for scalability measurements

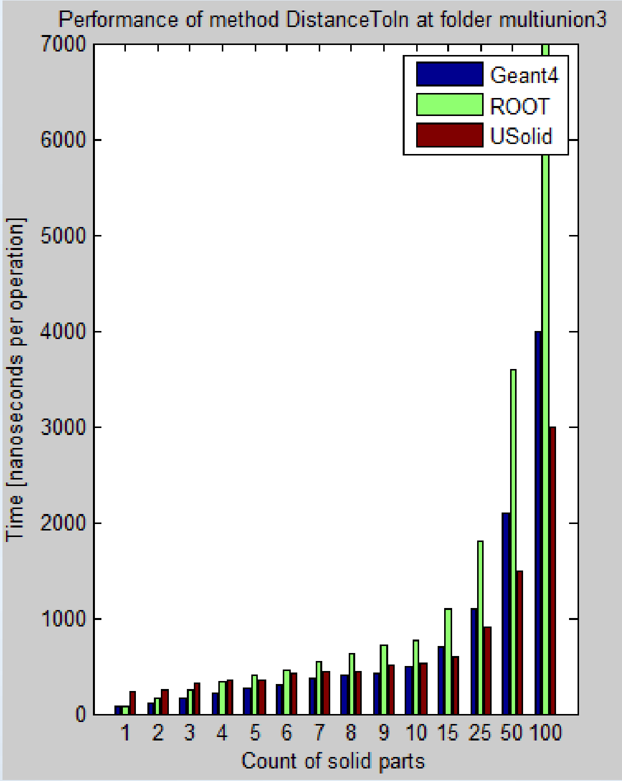


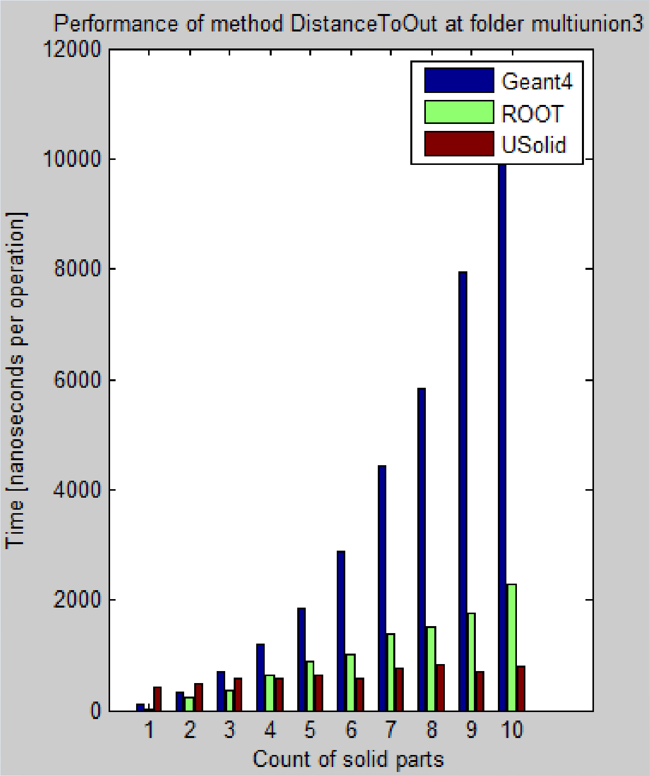
Test union solids for scalability measurements



The most performance critical methods



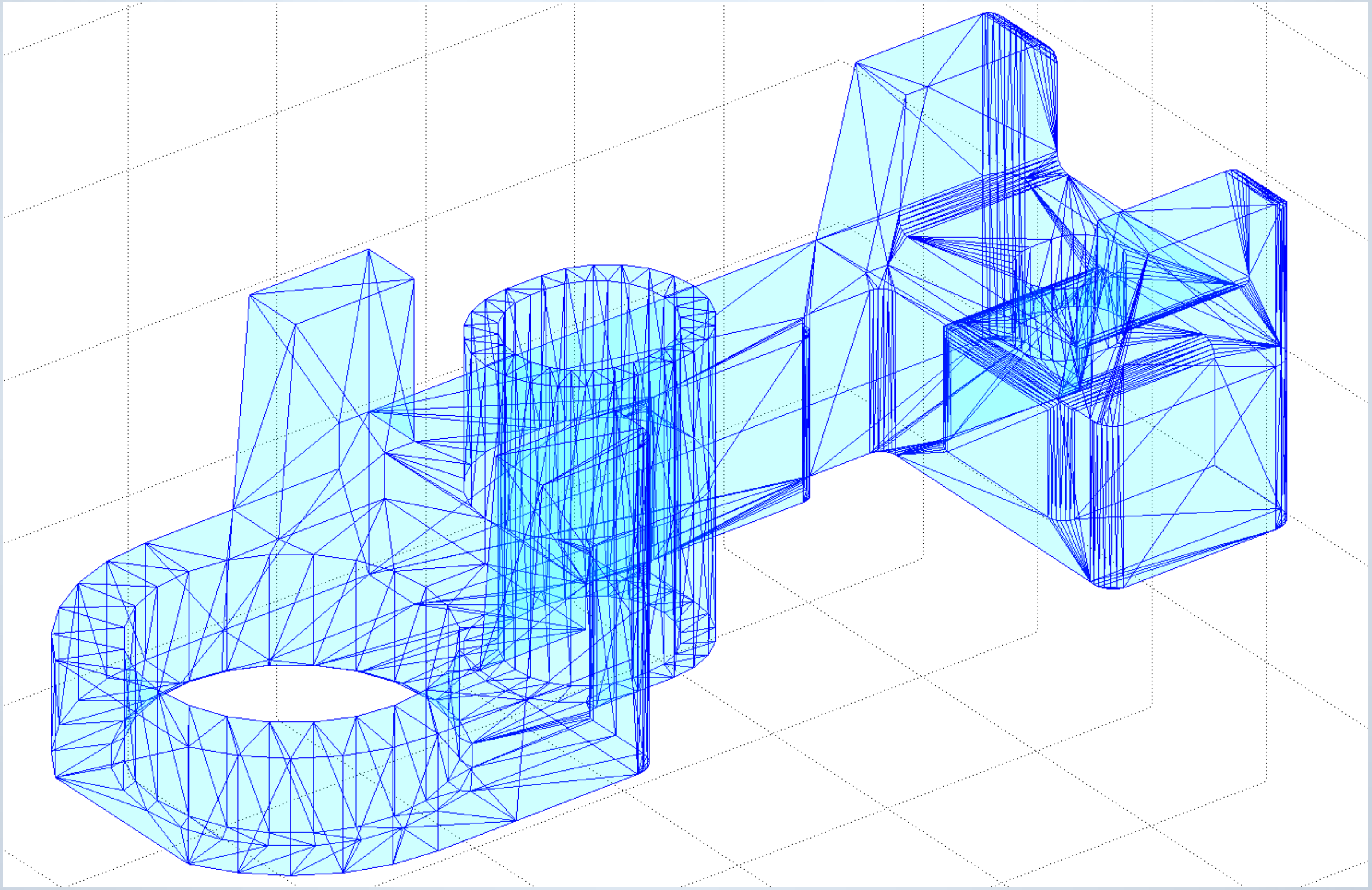




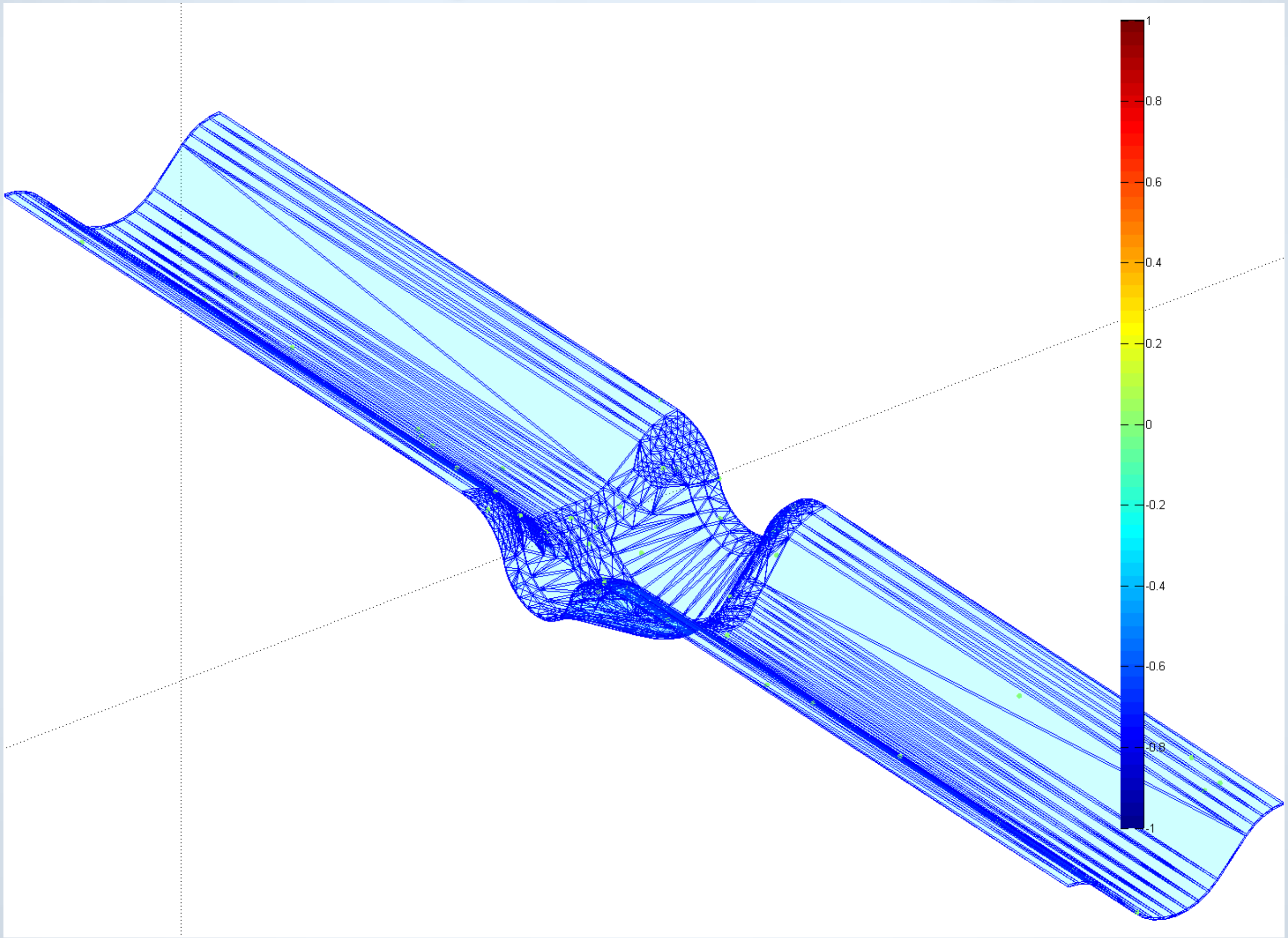
Tessellated Solid made fast

• • •

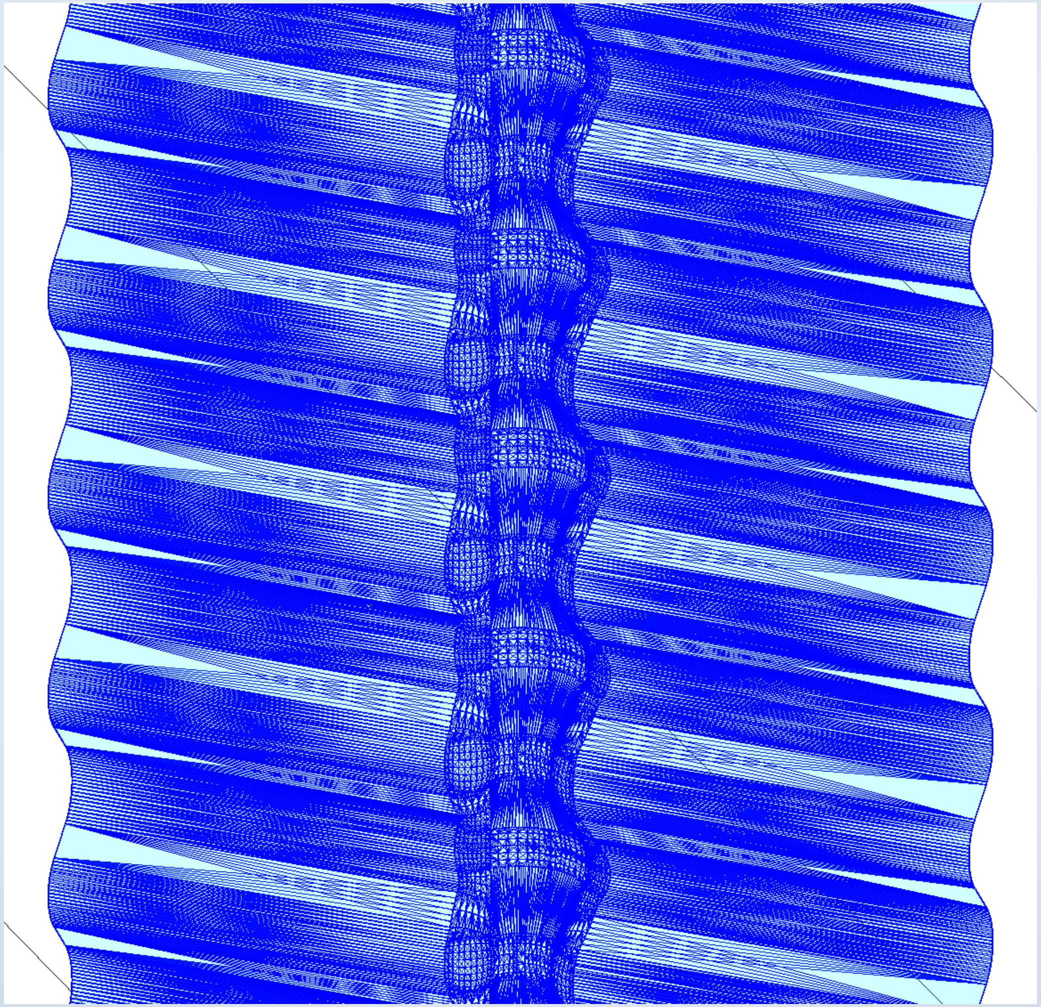
Test case a mechanical part with ~1.100 faces – *key-1.1k.gdml*



Test case foil with ~2.500 faces – *foil-2.5k.gdml*



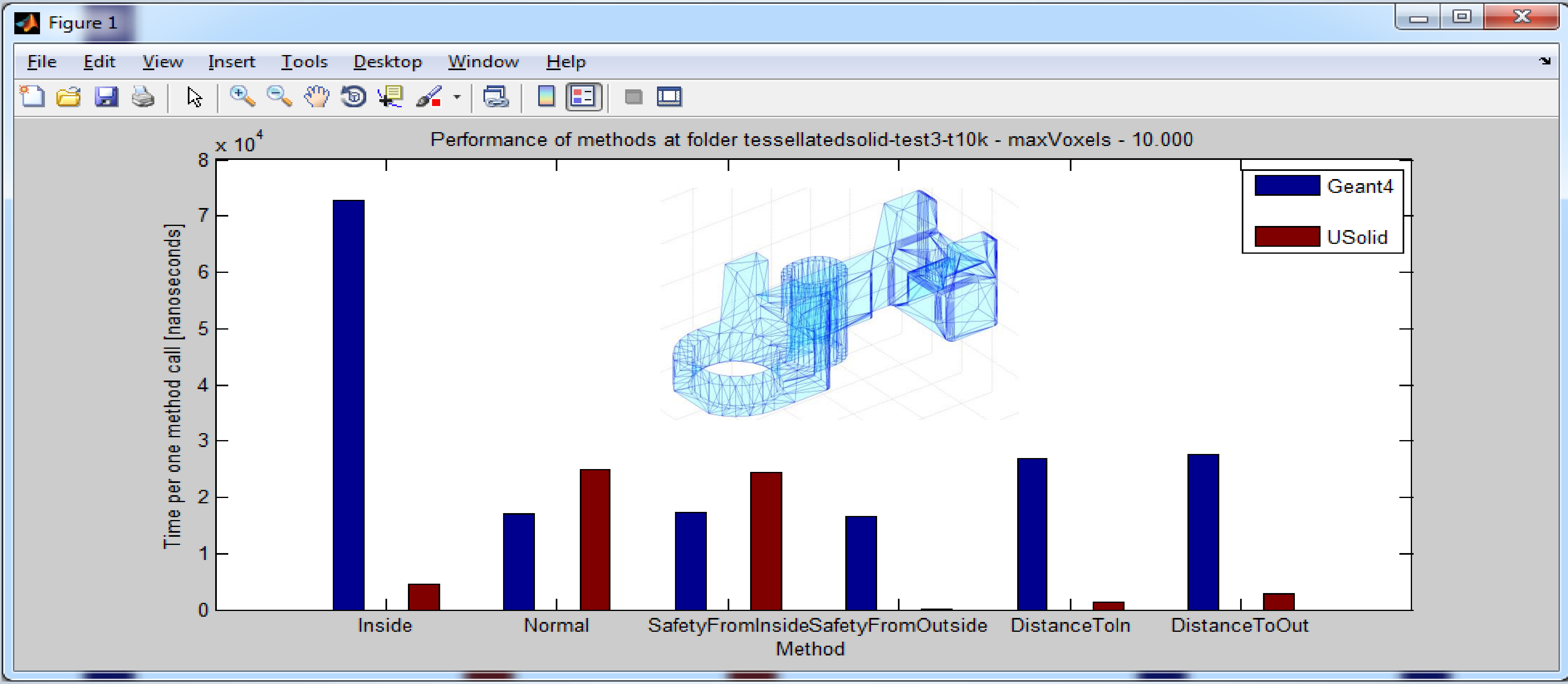
Test case foil ~164.000 faces for LHC experiment – *foil-164k.gdml*



Tessellated Solid notes

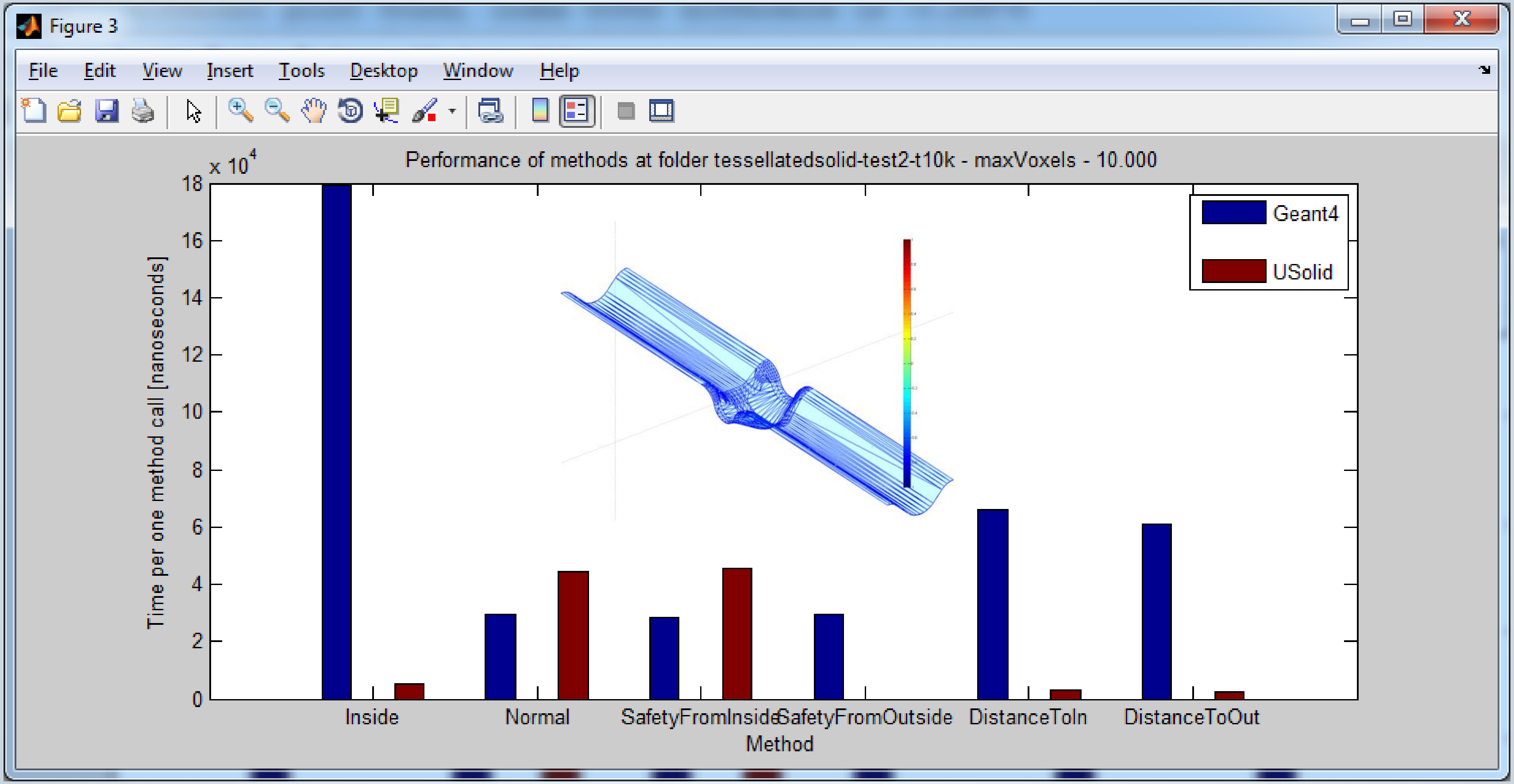
- The algorithms and datastructures were voxelized resulting in **dramatical performance enhancement in the all most performance critical methods**
- Also, G4TesselatedSolid had several weak parts of algorithm, used at initialization which had n^2 complexity.
- This sometimes caused very huge delays (e.g. in case of foil with 164k faces)
- We rewrote them to have $n \cdot \log n$ complexity
- Analysis for huge speedup of *Normal, SafetyFromInside, SafetyFromOutside* methods done, implementing now
- Analysis for lowering of ~33%+ of original memory requirements done, implementing now
- Very soon to be implemented for Geant4 **9.6** as G4TesselatedSolid (without bridge)

Performance – 1.1k/SCL 5 with 10k voxels



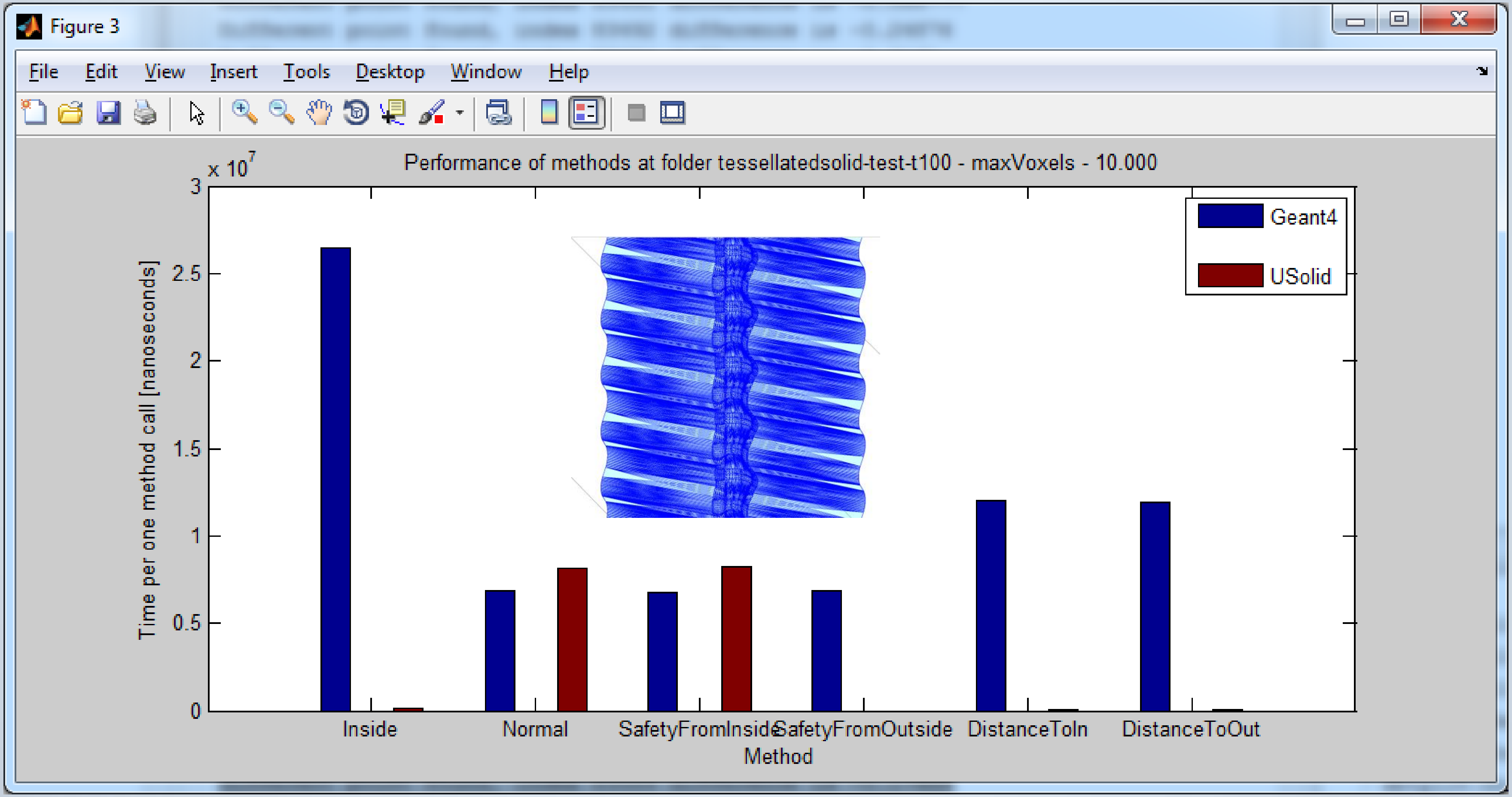
- **Speedup: 15x 1638x+ 19.4x 9.04x**

Performance – 2.5k/SCL5 with 10k voxels



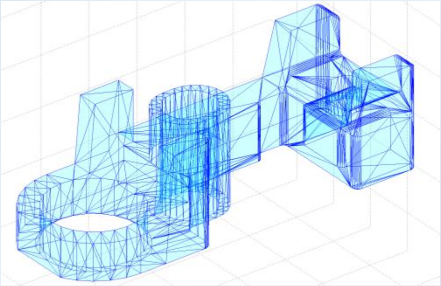
- **Speedup: 33x 1000x+ 22x 23x**

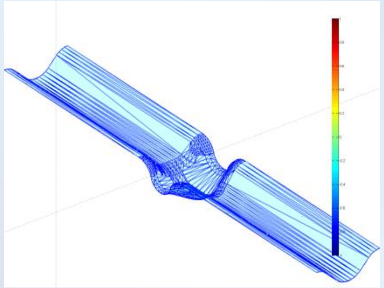
Performance – 164k/SCL5 with 10k voxels

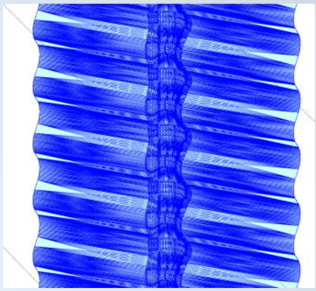


- **Speedup: 240x 1000x+ 133x 397x**

Memory overhead requirements for voxelization







Voxels / Case	key-1.1k.gdml	foil-2.5k.gdml	foil-164k.gdml
1000	1.6%	1.6%	1.6%
10.000	**4.5%**	**4%**	**3.9%**
100.000	16.5%	12.2%	8.7%
1.000.000	101% | 593/1193kB	60% | 1.1/1.8MB	19.5% | 66/79MB

In addition, G4TesselatedSolid memory requirements will be lessened by ~33%

- Each of tessel (can be millions) contains class G4VFacet
- Each of these facets self-contain between others these fields:
  - string geometryType: "G4TriangularFacet"
  - int nVertices = 3
  - double radiusSq // used only in constructor, as temp. variable
  - std::vector used (2x), even in cases of triangle; but std::vector can take e.g. 16+ bytes even when is empty
  - G4TessellatedGeometryAlgorithms *tGeomAlg
  - Data fields are planned to be moved to inherited classes (also for the reason that quadrangular facet is currently implemented as two triangular facets).
  - G4VFacet will be without data fields, only methods, becoming a real interface
  - There are more things to improve there, we listed only some of most obvious things needed to be replaced

Status of work

- ✓ Types and USolid interface are defined
- ✓ Bridge classes defined and implemented for both Geant4 and Root
- ✓ Testing suite defined and deployed
- ✓ Implementation of **Multi-Union as well as Tessellated solid** performance optimized and nearly completed
- ✓ Started implementation of primitives:
  - ✓ First implementation of Box, Orb (simple full sphere) and Trd (simple trapezoid)
  - ✓ Currently implementing: Cone, Tube and their segment version

Future work

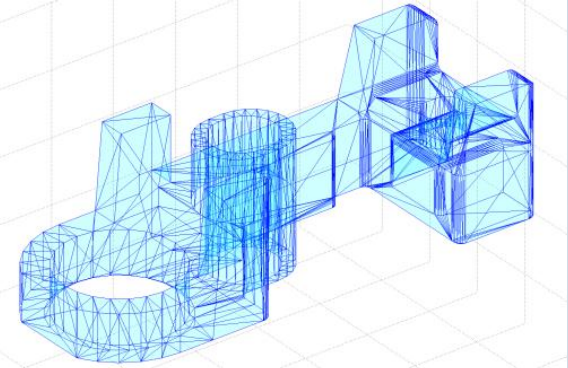
- Give priority to the most critical solids and those where room for improvement can be easily identified
- Systematically analyze and implement remaining solids in the new library

Thank you for your attention.

? ? ?

Questions ?

BakS - Visualizing mesh in Matlab (sbtpolyhedra.m)

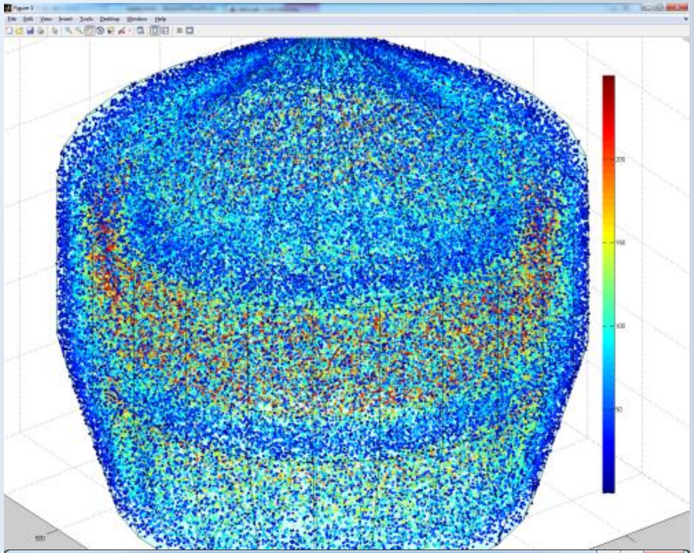


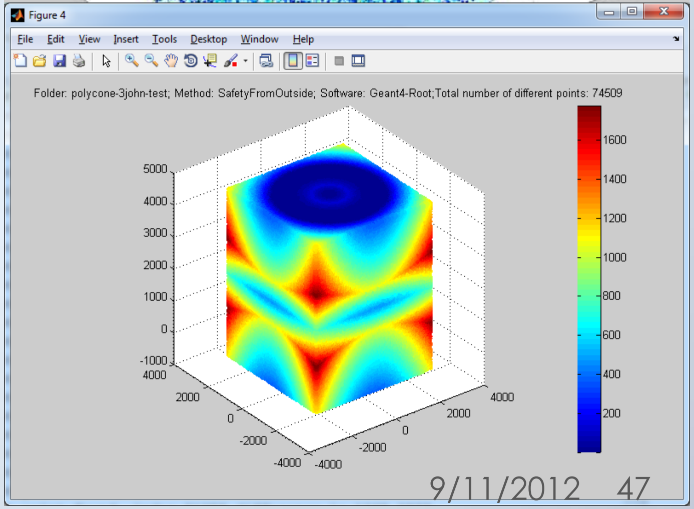
```
function res = sbtpolyhedra(method)
    filenameVertices = [method 'Vertices.dat'];
    filenameTriangles = [method 'Triangles.dat']
    filenameQuads = [method 'Quads.dat'];
    vertices = load(filenameVertices);
    quads = load(filenameQuads);
    triangles = load(filenameTriangles);
    hold on;
    h = patch('vertices',vertices,'faces',quads,'facecolor','c','edgecolor','b'); % draw faces in blue
    alpha(h,.1);
    h = patch('vertices',vertices,'faces',triangles,'facecolor','c','edgecolor','b'); % draw faces in blue
    alpha(h,.1);
    view(3), grid on;% default view with grid
end
```

BakS - Visualizing vectors of points in Matlab with color bar

- Key matlab commands:
  - `colormap('default');`
  - **scatter3** `(points(:,1), points(:,2), points(:,3), pointsize, values, 'filled');`
  - **`colorbar`**`;`

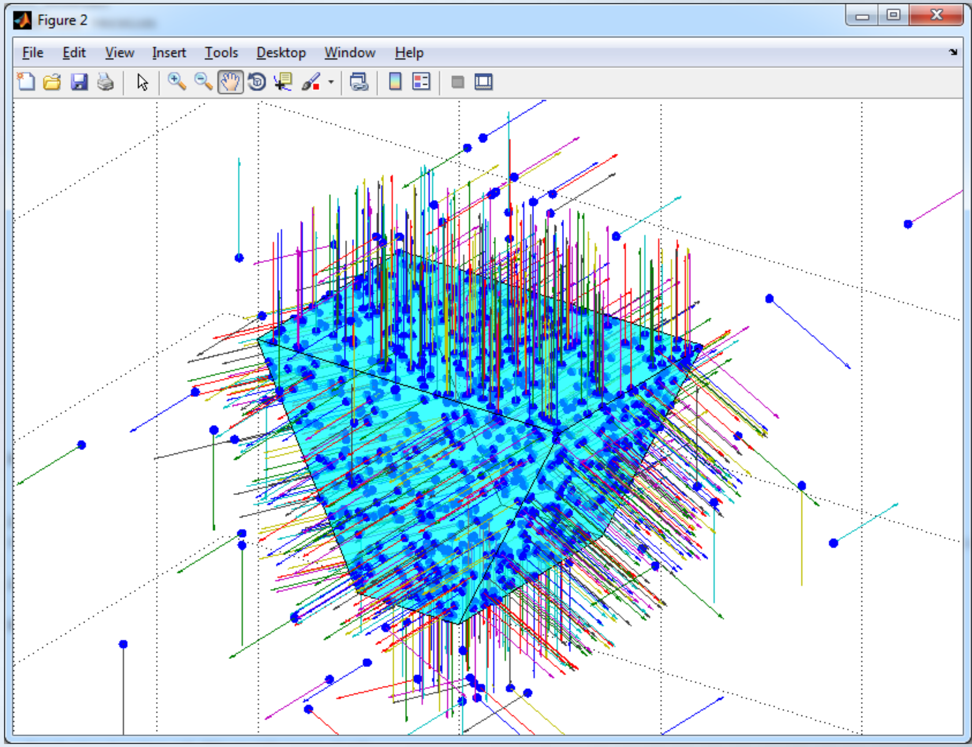
Scatter3 here uses table of *points* – each row consists of x, y, z than array of pointsize. But pointsize can be as well a numeric constant, which would be used for all points





BakS - Visualizing vectors with Matlab

- `quiver3(x,y,z,u,v,w,color);`



- x,y,z : array of points
- U,v,w: array of vector directions for corresponding point
- Color – colours used for vectors