Recent updates to voxelisation system and solids

Evgueni Tcherniaev

Geant4 development team

Tomsk State University

Contents

- Improved calculation of Extent
  - G4BoundingEnvelope class
  - Re-implementation of CalculateExtent()
- Revision of G4Solids
  - Revision of G4Box, G4Trd, G4Para, G4Trap
  - Revision of G4Tubs
  - G4ExtrudedSolid for right prism
- Next steps

Improved calculation of Extent

CalculateExtent()

- Each Geant4 solid should have a member function **`CalculateExtent()`** which calculates the minimum and maximum extent of the solid under the specified transformation and within the specified limits. This function is used for building the voxel structure for optimisation of the geometry

```
G4bool CalculateExtent(const EAxis pAxis,
                       const G4VoxelLimits& pVoxelLimit,
                       const G4AffineTransform& pTransform,
                       G4double& pMin, G4double& pMax) const;
```

- In Geant4-10.03 the **`CalculateExtent()`** method has been re-implemented for all the Geant4 solids and all wrappers for the VecGeom solids. The motivation was:
  - Unclear, difficult for understanding code
  - In many cases the calculation of the extent was not precise, for example, the extent for a sector of a tube was ~25% bigger than in reality
  - Inconsistency in calculation of the extent between the Geant4 solids and the wrappers for the VecGeom solids
  - Need to allow more exact comparison of the G4 primitives with VecGeom ones (i.e. adopt the same voxelisation structure)
- In addition to **`CalculateExtent()`**, all the Geant4 solids and wrappers now have a method **`BoundingLimits(pMin,pMax)`** which returns two extreme corners of the bounding box

G4BoundingEnvelope class

To facilitate implementation of the **`CalculateExtent()`** method in the solids, a new class **`G4BoundingEnvelope`** has been introduced:

- *Bounding envelope* is a sequence of 3D convex polygons that bounds the solid. The polygons should have equal number of vertices except first and last polygons, which may consist of a single vertex. The bounding box is a bounding envelope
- **`G4BoundingEnvelope`** has it own method **`CalculateExtent()`**, so there is no longer a need to implement the calculation of extent explicitly inside the solid
- The method expects transformation as an object of **`G4Transform3d`** class (see next slide) instead of **G4AffineTransform**, like a similar method in the Geant4 solids (see previous slide). That makes it convenient for the calculation of extent in reflected, scaled and Boolean constructs

G4BoundingEnvelope class: interface

Interface of the G4BoundingEnvepole class is rather simple - it has two constructors and two public methods:

```
G4BoundingEnvelope(const G4ThreeVector& pMin,
                   const G4ThreeVector& pMax);
// Constructor from an axis aligned bounding box (AABB)

G4BoundingEnvelope(const G4ThreeVector& pMin,
                   const G4ThreeVector& pMax,
                   const std::vector<const G4ThreeVectorList*>& polygons);
// Constructor from AABB and a sequence of polygons

G4bool BoundingBoxVsVoxelLimits(const EAxis pAxis,
                                const G4VoxelLimits& pVoxelLimits,
                                const G4Transform3D& pTransform3D,
                                G4double& pMin, G4double& pMax) const;
// Fast method, it returns true if the extent can be computed from the position of
// the bounding box relative to the voxel

G4bool CalculateExtent(const EAxis pAxis,
                       const G4VoxelLimits& pVoxelLimits,
                       const G4Transform3D& pTransform3D,
                       G4double& pMin, G4double& pMax) const;
```

CalculateExtent(): examples

- An example for «lazy developers» - using only the bounding box without building bounding polygons:

```
{
  G4ThreeVector bmin, bmax;
  BoundingLimits(bmin,bmax); // get bounding box
  G4BoundingEnvelope bbox(bmin,bmax); // constuct bounding envelope
  return bbox.CalculateExtent(pAxis,pVoxelLimit,pTransform,pMin,pMax);
}
```

- A more advanced example with building bounding polygons:

```
{
  // Quick check of the bounding box
  G4ThreeVector bmin, bmax;
  BoundingLimits(bmin,bmax);
  if (bbox.BoundingBoxVsVoxelLimits(pAxis,pVoxelLimit,pTransform,pMin,pMax))
  {
    return (pMin < pMax) ? true : false;
  }

  // Build bounding polygons
  std::vector<const G4ThreeVectorList*> polygons;
     ...

  // Calculate extent
  G4BoundingEnvelope benv(bmin,bmax,polygons);
  return benv.CalculateExtent(pAxis,pVoxelLimit,pTransform,pMin,pMax);
}
```

Impact to voxelisation

`CalculateExtent()` method has been re-implemented for all relevant classes

- it includes 15 wrappers, 10 CGS solids, 16 specific solids as well as reflected, scaled and Boolean constructs
- improved voxel structure: reduced memory footprint and faster optimization time
- improved tracking performance (~2% measured in pure ray-tracing “geantino” tests)

Geant4 10.02.p03
CMS setup:
Total memory consumed for geometry optimization: 22209 kByte
Total CPU time elapsed for geometry optimization: 4.6 seconds

ATLAS setup:
Total memory consumed for geometry optimization: 290665 kByte
Total CPU time elapsed for geometry optimization: 110 seconds

Geant4 10.03.p02
CMS setup:
Total memory consumed for geometry optimization: 16418 kByte (5.8 MByte less – 16% less)
Total CPU time elapsed for geometry optimization: 3.5 seconds (1.1 s faster)

ATLAS setup:
Total memory consumed for geometry optimization: 272640 kByte (18 MByte less – 6% less)
Total CPU time elapsed for geometry optimization: 89 seconds (21 s faster)

Other impact

- Consistent calculation of extent in the Geant4 solids and corresponding wrappers has allowed construction of identical voxel structures when using Geant4 and Geant4+VecGeom
- It also helped to improve calculation of the bounding boxes in the VecGeom solids
- Having identical voxel structures, we can do more accurate comparison of Geant4 and VecGeom performances
- Code for the Geant4 solids has been noticeably reduced, for example, the method CreateRotatedVertices() have been eliminated from all classes

Revision of Geant4 solids

Motivation

The revision has been inspired by the following factors:

- Most of the Geant4 solids were implemented a long time ago. We now have better hardware and software:
  - better compilers with more stable and advanced optimizers
  - better implementation of math functions, for example, std::min(a,b), std::max(a,b) are not anymore templates but processor instructions

- Several places where code could be improved, for example:
  - G4Trd was slower than G4Trap
  - use of std:atan2(y,x) in solids with a cut in Phi
  - G4Polycone and G4Polyhedra not optimized

- Particularities in the shapes used in the geometry setup of the CERN experiments
  - CMS: >90% of G4Trap shapes have rectangular YZ cross section
  - ALICE & ATLAS: 100% of G4ExtrudedSolid shapes are right prisms

Example: “std::max” instead of “if”

```
EInside G4Box::Inside(const G4ThreeVector& p)
const {
   EInside in = kOutside ;
   G4ThreeVector q(std::abs(p.x()), std::abs(p.y()),
std::abs(p.z()));
   if ( q.x() <= (fDx - delta) ) {
      if (q.y() <= (fDy - delta) ) {
         if ( q.z() <= (fDz - delta) ) {
            in = kInside ;
          } else if ( q.z() <= (fDz + delta) ) {
             in = kSurface ;
         }
       } else if ( q.y() <= (fDy + delta) ) {
          if ( q.z() <= (fDz + delta) ) {
             in = kSurface ;
          }
       }
    } else if ( q.x() <= (fDx + delta) ) {
       if ( q.y() <= (fDy + delta) ) {
          if ( q.z() <= (fDz + delta) ) {
             in = kSurface ;
          }
       }
   }
   return in ;
} // possible number of comparisons 2 - 4
```

```
VUSolid::EnumInside UBox::Inside(const
UVector3& aPoint) const
{
  double dz = std::abs(aPoint.z()) - fDz;
  if (dz > delta) return EnumInside::eOutside;

  double dx = std::abs(aPoint.x()) - fDx;
  if (dx > delta) return EnumInside::eOutside;

  double dy = std::abs(aPoint.y()) - fDy;
  if (dy > delta) return EnumInside::eOutside;

  if (dx > -delta ||
      dy > -delta ||
      dz > -delta) return EnumInside::eSurface;
  return EnumInside::eInside;
} // possible number of comparisons 1 - 6
```

```
EInside G4Box::Inside(const G4ThreeVector& p)
const {
   G4double dist = std::max(std::max(
                        std::abs(p.x())-fDx,
                        std::abs(p.y())-fDy),
                        std::abs(p.z())-fDz);
   if (dist > delta) return kOutside;
   return (dist > -delta) ? kSurface : kInside;
} // possible number of comparisons 3 - 4
```

Speed-up up to 3 times!

Example: use of temporary variables

```
G4double invx = (v.x() == 0) ? DBL_MAX : -1./v.x();
G4double tx1 = (p.x() + fDx)*invx;
G4double tx2 = (p.x() - fDx)*invx;
G4double tmin = std::min(tx1,tx2);
G4double tmax = std::max(tx1,tx2);

G4double invy = (v.y() == 0) ? DBL_MAX : -1./v.y();
G4double ty1 = (p.y() + fDy)*invy;
G4double ty2 = (p.y() - fDy)*invy;
tmin = std::max(tmin,std::min(ty1,ty2));
tmax = std::min(tmax,std::max(ty1,ty2));

G4double invz = (v.z() == 0) ? DBL_MAX : -1./v.z();
G4double tz1 = (p.z() + fDz)*invz;
G4double tz2 = (p.z() - fDz)*invz;
tmin = std::max(tmin,std::min(tz1,tz2));
tmax = std::min(tmax,std::max(tz1,tz2));
```

```
G4double invx = (v.x() == 0) ? DBL_MAX : -1./v.x();
G4double tx1 = (p.x() + fDx)*invx;
G4double tx2 = (p.x() - fDx)*invx;
G4double txmin = std::min(tx1,tx2);
G4double txmax = std::max(tx1,tx2);

G4double invy = (v.y() == 0) ? DBL_MAX : -1./v.y();
G4double ty1 = (p.y() + fDy)*invy;
G4double ty2 = (p.y() - fDy)*invy;
G4double tymin = std::max(txmin,std::min(ty1,ty2));
G4double tymax = std::min(txmax,std::max(ty1,ty2));

G4double invz = (v.z() == 0) ? DBL_MAX : -1./v.z();
G4double tz1 = (p.z() + fDz)*invz;
G4double tz2 = (p.z() - fDz)*invz;
G4double tmin = std::max(tymin,std::min(tz1,tz2));
G4double tmax = std::min(tymax,std::max(tz1,tz2));
```

Speed-up by 2 times!

What has been done

- All Geant4 solids have the following six methods which are used for tracking:
  - `Inside(p)`, detects position of a point, can be “inside”, “on surface”, “outside”
  - `SurfaceNormal(p)`, computes normal at a point of the surface
  - `DistanceToIn(p)`, estimates distance to the surface from outside (will be referenced as Safety To In)
  - `DistanceToOut(p)`, estimates distance to the surface from inside (will be referenced as Safety To Out)
  - `DistanceToIn(p,v)`, computes exact distance to the surface along “v” from outside
  - `DistanceToOut(p,v)`, computes exact distance to the surface along “v” from inside

- Several benchmarks have been implemented to measure performance of main methods and control modification of the code, you can find them in `source/geometry/benchmarks`:
  - `benchBoxTrdParaTrap.cc`
  - `benchTubs.cc`, benchmark for all possible topologies of G4Tubs
  - `benchPolyhedrons.cc`, benchmark for hexagonal prism defined as G4ExtrudedSoilid, G4TesselatedSolid and G4Polyhedra
  - `benchQuadrics.cc`, benchmark for shapes limited by second order surfaces (G4Orb, G4Ellipsoid, G4EllipticalTube, G4EllipticalCone, G4Paraboloid and G4Hype)
- A collection of general purpose utilities for geometry related calculations has been implemented and grouped in G4GeomTools class (~25 utilities are available at present time)

- G4Box, G4Trd, G4Para, G4Trap, G4Orb – revision has been completed
- G4Tubs, G4EllipticalCone – most of the methods have been revised, except `DistanceToIn(p,v)` and `DistanceToOut(p,v)`
- G4ExtrudedSolid – specialized code has been added for convex right prism

“Trapezoid like” solids

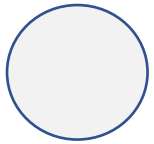
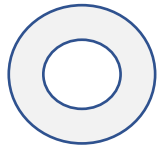
This and next two slides contain result of the benchmarks mentioned on the previous slide.
Measurements were done for 100 million calls on MacBook Pro, 2.7 GHz Intel Core i5

	G4Box		**G4Trd**		**G4Para**		**G4Trap**	
	old	new	old	new	old	new	old	new
Inside								Type 1 / Type2 / General
kInside	0.39 s	0.39 s	0.58 s	0.51 s	0.48 s	0.59 s	1.06 s	0.47 / 0.61 /0.71 s
kSurface	1.09 s	0.39 s	1.51 s	0.51 s	1.27 s	0.60 s	1.37 s	0.47 / 0.61 /0.70 s
kOutside	0.77 s	0.32 s	0.89 s	0.40 s	0.78 s	0.50 s	0.88 s	0.40 / 0.56 /0.60 s
Safety to In	0.35 s	0.31 s	2.13 s	0.42 s	0.94 s	0.50 s	0.81 s	0.39 / 0.58 / 0.60 s
Safety to Out	0.52 s	0.34 s	1.16 s	0.45 s	0.91 s	0.52 s	0.84 s	0.42 / 0.58 / 0.63 s
Distance To In	2.96 s	1.15 s	3.21 s	2.07 s	2.26 s	2.15 s	4.03 s	2.48 s
Distance To Out	3.61 s	1.79 s	4.53 s	2.59 s	4.06 s	2.75 s	4.49 s	3.06 s
Surface Normal	1.62 s	0.60 s	3.22 s	1.59 s	2.27 s	1.71 s	1.98 s	1.79 s

Type 1 – rectangular YZ cross-section + symmetrical X-sides
Type 2 – just rectangular YZ cross-section

G4Tubs: elimination of std::atan2(y,x)

Below is the result of a benchmark test for most popular G4Tubs topologies: “cylinder”, “hollow cylinder” and “sector of a tube”. Major contribution to the performance improvement of “sector of a tube” gave elimination of std::atan2(y,x)

	 old	new	 old	new	 old	new
Inside						
kInside	0.53 s	0.63 s	0.53 s	0.66 s	3.98 s	0.77 s
kSurface	1.02 s	0.63 s	1.18 s	0.66 s	4.42 s	0.78 s
kOutside	0.95 s	0.47 s	1.10 s	0.49 s	1.76 s	0.61 s
Safety to In	0.66 s	0.50 s	0.68 s	0.52 s	3.17 s	0.74 s
Safety to Out	0.50 s	0.51 s	0.54 s	0.53 s	1.62 s	0.66 s
Distance To In	1.99 s	*	2.59 s	*	3.57 s	*
Distance To Out	2.61 s		2.90 s		8.01 s	
Surface Normal	1.67 s	1.13 s	2.14 s	1.52 s	7.46 s	1.95 s

* G4Tubs::DistanceToIn(p,v)/DistanceToOut(p,v) has not been revised yet

G4ExtrudedSolid: convex right prism

Practically 100% of G4ExtrudedSolid shapes used by the LHC experiments are *right prisms*.
By request of CMS a specialized implementation for *convex right prism* have been added into G4ExtrudedSolid.
Below is the result of the benchmark for a hexagonal prism defined as G4Polyhedra and G4ExtrudedSolid
(see `geometry/benchmarks/benchPolyhedrons.cc`)

	G4Polyhedra	**G4ExtrudedSolid** quasi right prism	**G4ExtrudedSolid** true right prism
Inside			
kInside	18.98 s	10.32 s	1.17 s
kSurface	13.91 s	7.82 s	1.16 s
kOutside	0.74 -10.32 s	1.90 - 5.08 s	1.08 s
Safety to In	19.16 s	2.06 s	0.95 s
Safety to Out	18.59 s	160.50 s	0.95 s
Distance To In	11.99 s	24.77 s	3.21 s
Distance To Out	14.51 s	82.93 s	3.71 s
Surface Normal	19.77 s	38.64 s	2.63 s

“Geantino” tracking in real geometries

”Geantino” is an imaginary particle which always moves along straight lines without interaction with material. The result of the “geantino” tracking test is reproducible from one version to another and is very convenient for control of modifications in the Geant4 geometry code.

CMS 2016 geometry, 100K ”geantino” events

G4 10.02	G4 10.03	G4 10.04-dev	G4 10.04-dev + VecGeom
108 s (106%)	105 s (103%)	102 s (100%)	95 s (93%)

“Almost full” ATLAS geometry, 100K “geantino” events

G4 10.02	G4 10.03	G4 10.04-dev	G4 10.04-dev * + VecGeom
346 s (105%)	340 s (103%)	330 s (100%)	472 s (143%)

* Too big difference, unexpected slow down, need investigation

Conclusion and Plans

- Current results give confidence that it is possible to achieve noticeable speed-up by revising the Geant4 shapes code and therefore provide immediate benefit to users and experiments simulations.

- Future plans:
  - Complete revision of G4Tubs, adopt its code for G4CutTubs
  - Add specialized implementation for *concave right prism* into G4ExtrudedSolid
  - Revise G4Polycone and G4Polyhedra
  - Feed back ideas/improvements to corresponding VecGeom shapes algorithms where possible
  - Keep using Geant4 geometry as reference for validation