Managed by Fermi Research Alliance, LLC for the U.S. Department of Energy Office of Science

Integration of VecGeom into Geant4

Guilherme Lima (Fermilab)
for the VecGeom Group

Geant4 Collaboration Meeting
Fermilab – September 30, 2015

Outline

- Note: VecGeom has already been introduced by Sandro in previous presentation
- In this talk:
  - VecGeom integration
    - what is it?
    - why is it needed?
    - how is it done?
  - Results
    - callgrind on a simple geometry
    - full CMS detector
  - How to use VecGeom shapes with Geant4
  - Current status
  - Summary

VecGeom integration with Geant4

- **What is it?**
  It means that we can run “any Geant4 job” using solids and algorithms from the VecGeom library
  - ...uses VecGeom shapes when available, if not uses the USolids versions then the Geant4 ones
  - no changes are needed in user/application code other than turning on a few compilation switches
  - VecGeom shapes: only scalar algorithms are used – no parallelized SIMD processing of tracks!

VecGeom integration with Geant4

- **Why is it needed?**
  - It allows direct, high-level (physics) comparisons, for validation of solids and their algorithms, and for optimization purposes (Geant4 / USolids / VecGeom)
  - use of well tested Geant4 tools for testing, logging, debugging, validation, visualization and so on
  - make VecGeom shapes immediately and transparently available to any Geant4 applications
  - *expected* to provide performance gains with respect to both USolids and Geant4 shapes – to be verified!
  - promotes widespread adoption, feeding back into further developments and contributions

VecGeom integration with Geant4

- **How is it done?**
  - VecGeom was designed to be *USolids-compatible*
  - Existing USolids interface was used, e.g.

    G4Box → G4UBox → UBox → (VecGeom) SimpleBox

  - USolids shapes were *extended* to either use USolids source code, or to define USolids shapes as inheriting from the VecGeom shapes, depending on whether a pre-processor macro is defined from compilation command lines:

    ```
    g++ […] -DVECGEOM_REPLACES_USOLIDS […]
    ```

  - if that macro is not defined, current USolids shapes and algorithms are used
  - all navigation is still performed and controlled by Geant4
  - boolean operations with solids still managed by Geant4

Example: the integrated UBox

```
#ifndef USOLIDS_UBox
#define USOLIDS_UBox

#ifdef VECGEOM_REPLACE_USOLIDS

//============== here for VecGeom-based implementation
#include "volumes/SpecializedBox.h"
#include "volumes/LogicalVolume.h"
#include "volumes/UnplacedBox.h"
#include "base/Transformation3D.h"

class UBox: public vecgeom::SimpleBox {
  // just forwards UBox to vecgeom::SimpleBox
  using vecgeom::SimpleBox::SimpleBox;

public:
  // add default constructor for tests
  UBox() : vecgeom::SimpleBox(new vecgeom::LogicalVolume(new vecgeom::UnplacedBox()),
                              &vecgeom::Transformation3D::kIdentity, this) {}
};
//============== end of VecGeom-based implementation

#else

//============== here for USolids-based implementation
#include "VUSolid.hh"
class UBox : public VUSolid
{
  /// ... existing UBox declarations ...
};
//============== end of USolids-based implementation

#endif  // VECGEOM_REPLACE_USOLIDS
#endif  // USOLIDS_UBox
```

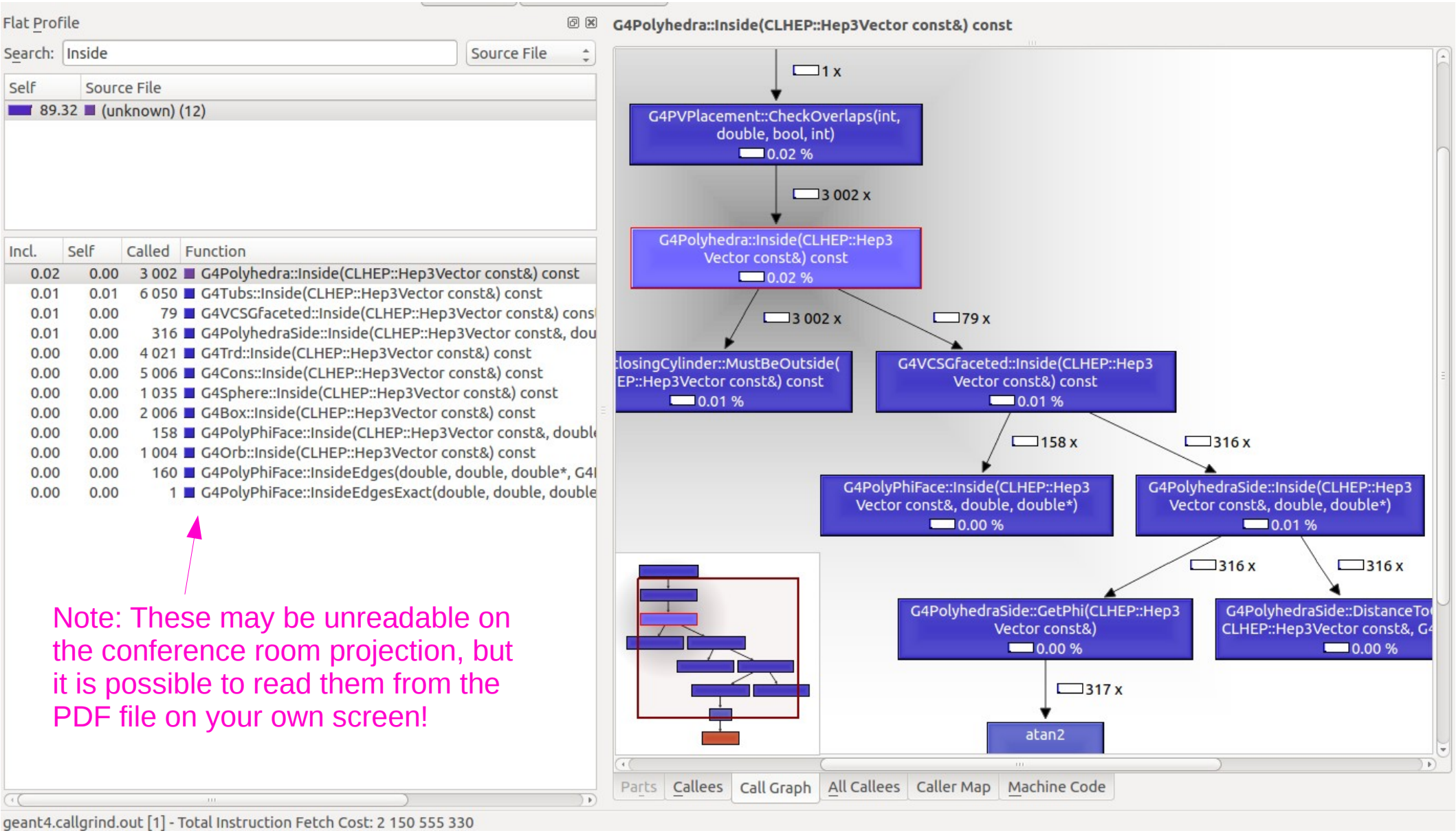
Macro that enables the use of VecGeom shapes

Most shapes only needed this part.

Only two shapes needed any extra declarations.

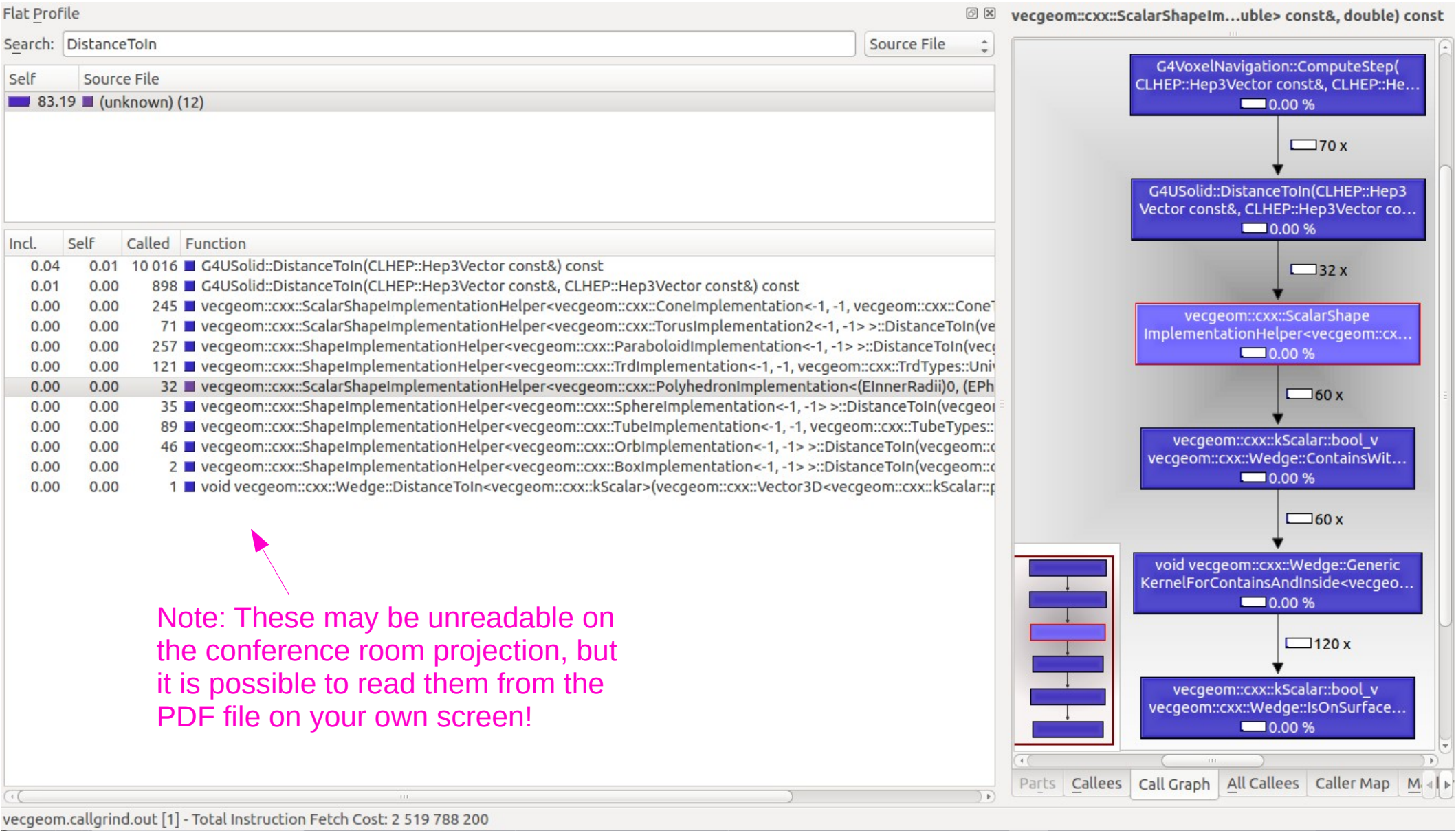
Existing Usolids used when flag is not enabled

CallGrind graphs: using Geant4 shapes



Note: These may be unreadable on the conference room projection, but it is possible to read them from the PDF file on your own screen!

CallGrind graphs: using VecGeom shapes



Note: These may be unreadable on the conference room projection, but it is possible to read them from the PDF file on your own screen!

VecGeom integration how-to

How can I use VecGeom shapes within Geant4?

- Overview
  - Step 1: build VecGeom and Usolids libraries
  - Step 2: build Geant4 libraries linked to both USolids and VecGeom libraries
  - Step 3: build your Geant4 application

Tip: CMake takes care of carrying the configuration over from one step to the next

VecGeom integration how-to

- Step 1: build USolids and VecGeom libraries
  - VecGeom provides the USolids library configured to use VecGeom shapes and algorithms

```
cmake -DBACKEND=Scalar -DGEANT4=OFF -DUSOLIDS=ON -DUSOLIDS_VECGEOM=ON \
    [...other optional VecGeom switches as needed...] \
    -DCMAKE_INSTALL_PREFIX=${VGINSTALLDIR}  ${VGSOURCE}

make install
```

  - Notes:
    - Using “make install” is important to properly install libraries, header files and configuration scripts, to help next steps to find them
    - Only scalar backend can be used by Geant4 (one track at a time)
    - **`-DGEANT4=OFF`** is required to avoid circular dependencies, as **`-DGEANT4=ON`** would introduce Geant4 dependencies in VecGeom
    - If **`-DUSOLIDS_VECGEOM=ON`** is omitted, USolids shapes and algorithms will be used instead of VecGeom ones.

VecGeom integration how-to

- Step 2: build Geant4 libraries
  - Geant4 must be compiled with switches to enable the *external USolids library* provided by VecGeom

```
export Usolids_DIR=${VGINSTALLDIR}/lib/Cmake/Usolids/

cmake -DCMAKE_INSTALL_PREFIX=${G4INSTALL} \
      -DGEANT4_BUILD_CXXSTD=c++11 \
      -DGEANT4_USE_USOLIDS=ON -DGEANT4_USE_SYSTEM_USOLIDS=ON \
      -DGEANT4_INSTALL_DATADIR=${G4INSTALL}/share/Geant4-10.2.0/data \
      -DGEANT4_USE_GDML=ON -DGEANT4_BUILD_MULTITHREADED=OFF \
      ${G4SOURCE}

make install
```

  - Notes:
    - C++11 is required by VecGeom
    - Switch **`-DGEANT4_USE_SYSTEM_USOLIDS=ON`** will force the use of an external USolids library, rather than the one which comes with Geant4
    - Few modifications to Geant4 code were needed, already included in the devel version ref-08, to be released with version 10.2 (December)

VecGeom integration how-to

- Step 3: build your Geant4 application
  - Tell CMake where to find the Geant4 libraries from step 2 and build it

```
cmake -DGeant4_DIR=${G4INSTALL}/lib/Geant4-10.2.0 ${SRCDIR}

make
```

- More complete instructions available in subdirectory *doc* of VecGeom repository

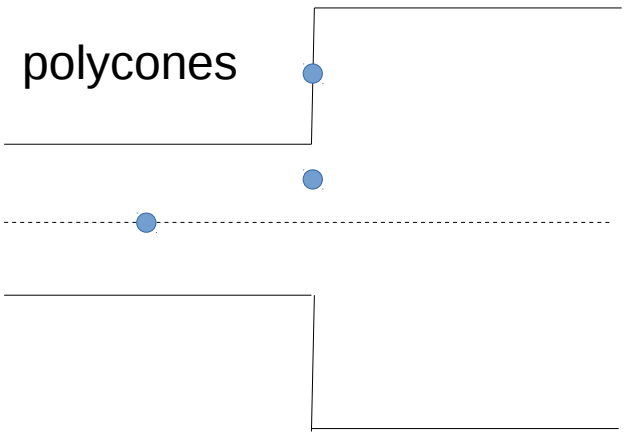
  `https://gitlab.cern.ch/VecGeom/VecGeom`

Testing with a complex detector

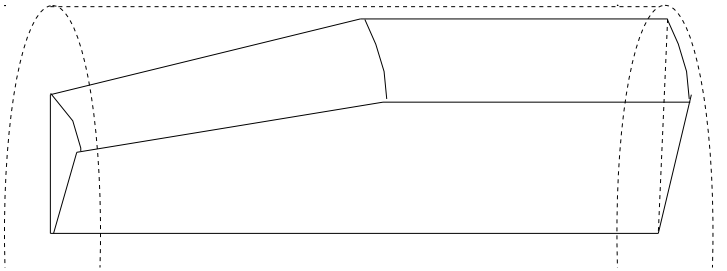
- FullCMS test in geant4 tests repository
- Built agains geant4.10.2.beta
- CMS geometry loaded from a GDML file
  - some error messages due to rounding in GDML data
  - during simulation, hints of overlapping volumes
- Job runs for some time, then crashes
  - debugging is under way, some pathological bugs identified and fixed
- Plan to use it for performance comparison
  - hopefully in a time scale of a couple weeks

Full CMS tests

- Identified some missing features
  - e.g. points on surface from polycone's Inside()
- Some pathological cases tracked and fixed
  - negative Rmin in polycone bounding tube, negative safeties, missing tolerances, ...
- Geant4 tools used extensively, specially navigation checks



polycone's bounding tube



tolerance added to Rmax and
subtracted from Rmin – *if non-zero*

VecGeom integration – Status

- **Current status**
  - first batch of shapes available
  - choice of shapes based on a full CMS model
  - ready: box, trapezoid, tube, cone, polycone, polyhedra, with native *phi*-wedges and *Rmin* where appropriate
    - tested under FullCMS conditions
  - almost ready: orb, sphere, trd, torus, paraboloid
    - not fully tested yet, since not used for FullCMS
    - trd and torus are part of a newer cms2015 model
  - visualization is not available yet
    - CreatePolygon() required for each shape
  - FullCMS tests under way, expected soon

Summary

- **It is possible to use VecGeom shapes from Geant4**
  - integration code is very new
    - it will go through review before being released, but the concept has been proved
    - once validated, VecGeom shapes will replace corresponding USolids shapes
  - nice leverage of Geant4 testing and validation tools
  - no changes needed in user application code
  - final stages of testing on a full CMS model
  - shapes ready: box, trapezoid, tube, cone, polycone, polyhedra, with native *phi*-wedges and *Rmin* where appropriate
    - partially tested under FullCMS conditions
  - shapes *almost ready*: orb, sphere, trd, torus, paraboloid
    - not fully tested yet, since not used for FullCMS