Fermilab

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

VecGeom – Vectorized Geometry

Guilherme Lima
for the GeantV Group

US ASCR-HEP Meeting
Fermilab, January 30, 2015

Presentation Outline

- Motivations
  - Need for performance optimization
  - Accelerating options
  - HEP detector simulations (Geant4)
- Geometry in HEP simulations
  - Requirements and challenges
  - Implementation choices
- Status and outlook
  - Shapes implemented
  - Preliminary performance
  - Summary and outlook

Improving performance – common options

- Multi-threading
  - Already used in Geant4.10-MT
  - Not covered in this talk
- New architectures and co-processors
  - GPGPUs or Intel Xeon-Phi, require specific software layers (Cuda, OpenCL, OpenMP, MPI, ...)
  - Specialized cores, used for the *intense* kernels
- SIMD-vector instructions (SSE, AVX, AV-512,...)
  - Explicit vectorization using libraries or intrinsics
  - Compiler autovectorization, promoted by smart structuring of data and algorithms

All are orthogonal paths → multiplicative gains!

Geometry in HEP simulations

- *Detector description*
  - an hierarchycal, multi-level structure of 'mother' and 'daughter' shapes
    - allow for the replication of common composite elements
  - Class concepts for separate responsibilities:
    - geometrical properties: shapes, dimensions
    - geometrical algorithms: containment, distances, volumes, normal vectors, Extent, etc.
    - relative positioning, coordinate transformations, materials

- Navigation

  Given track parameters: position (x,y,z) and direction (dx,dy,dz), predict particle trajectories and intersections with any geometrical boundaries.

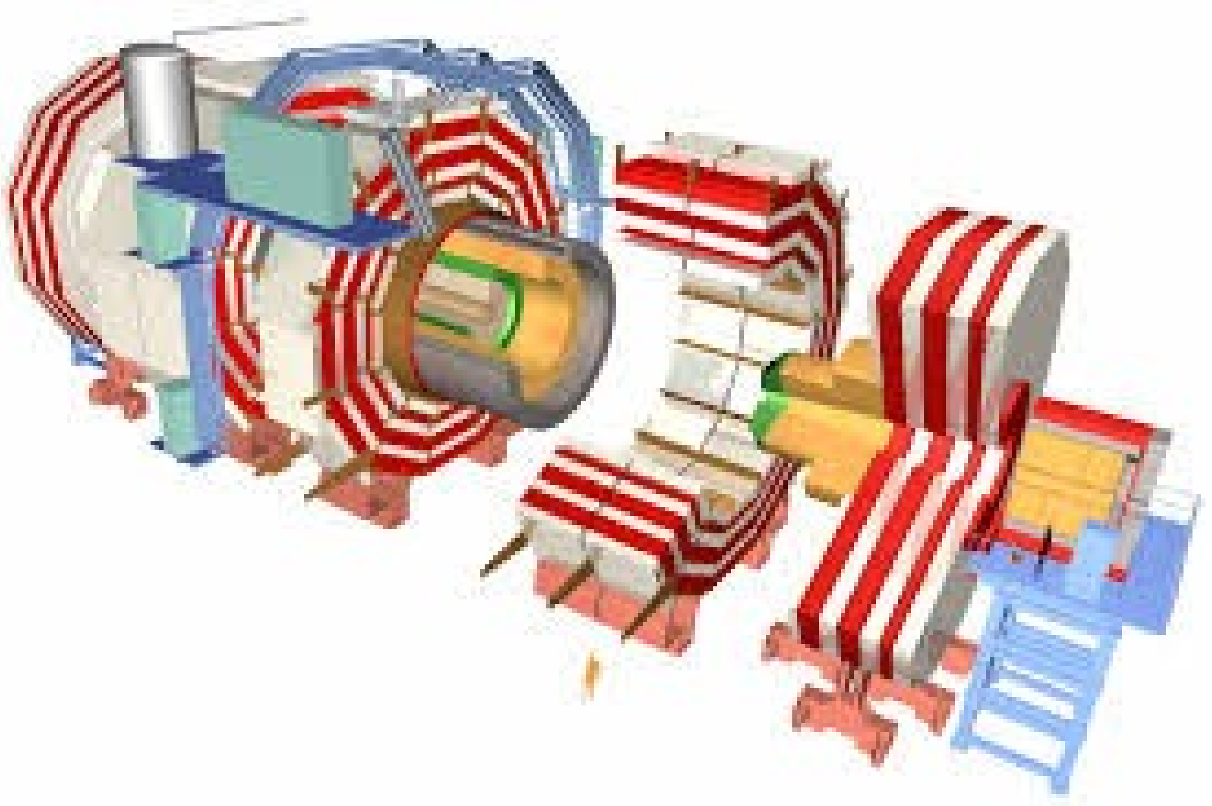
  External managers will take care of interactions with physics processes (including magnetic fields) and updates to track properties and positioning, display, etc.

Geometry in HEP simulations

Detector description

An hierarchycal, multi-level structure of 'mother' and 'daughter' shapes, allows for easy replication of common composite elements.

Our simplified version of the CMS detector contains about 4,000 elements in a 15-level hierarchy.



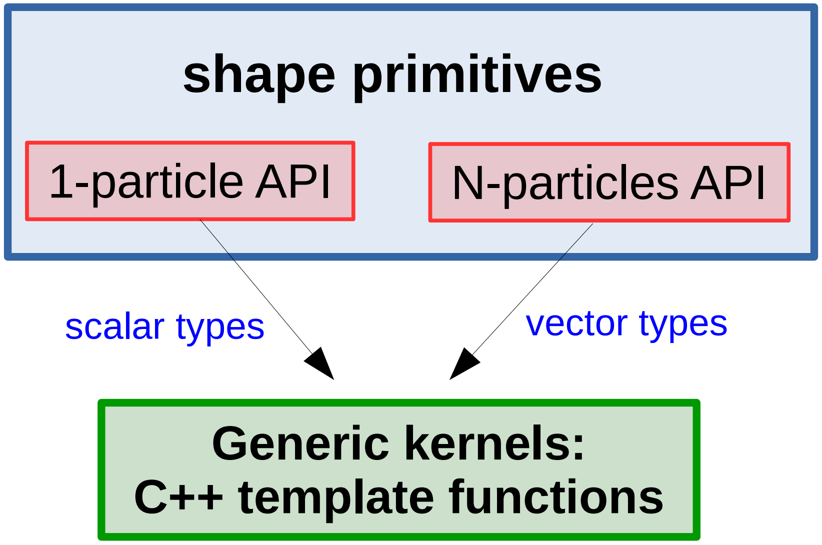
VecGeom – requirements and challenges

=> VecGeom: a high-performance HEP geometry system

- Multi-purpose:
  - originally developed to be a turn-key replacement for HEP simulation applications (Geant4, Root, USolids)
  - could also be useful for reconstruction and other applications
- Focus on new hardware architectures
  - uses SIMD vectors whenever possible, but falls back to scalar calculations if needed → vectorization, a distinct feature of VecGeom
- Platform independent
  - CPUs, co-processors, GPGPUs, …future… → use of generic data types, tuned by architecture-specific traits during compilation
- Low maintenance with minimal code duplication
  - Use of new features of latest C++ standards → generic source code, with templated functions to produce fast, platform-independent kernels

Implementation choices

- Make use of recent trends to speedup simulations
  - New SIMD architectures with larger registers of 128, 256 or up to 512 bits $\rightarrow$ massively parallel computing (use of vector libraries, for instance the Vc library by Matthias Kretz)
  - Challenge: re-write millions of lines of Geant4 code, while keeping it future-proofed and backward compatible $\rightarrow$ code duplication would lead to a maintenance nightmare...
  - Idea: generic templated kernels, with carefully designed data structures to maximize data locality and optimize data access and data transfers to co-processors and GPUs (more details later)
  - Avoid use of branching, to maximize synchronization among multiple threads
  - Let's see how these are done, in more details...



Avoiding code duplication

- Support of multiple platforms usually means multiple versions of source code
- What are the differences between the two versions of code shown on the right?
- → Primarily: types and their operators, function attributes (__device__), also some higher level functions, e.g. conditional assignment
- Avoid code duplication by abstracting away differences into common types or overloaded functions defined in trait structures.

cuda

```
template <int N>
__device__
double Planes<N>::DistanceToOut(
    double const (&plane)[4][N],
  Vector3D<double> const &point,
  Vector3D<double> const &direction) {

  double bestDistance = kInfinity;
  for (int i = 0; i < N; ++i) {
    double distance;
    distance = -(plane[0][i]*point[0] + plane[1][i]*point[1] +
                 plane[2][i]*point[2] + plane[3][i]);
    distance /= (plane[0][i]*direction[0] + plane[1][i]*direction[1] +
                 plane[2][i]*direction[2]);
    bestDistance = (distance < bestDistance) ? distance : bestDistanc
  }
  return bestDistance;
}
```

Vc

```
template <int N>
Vc::double_v Planes<N>::DistanceToOut(
    double const (&plane)[4][N],
  Vector3D<Vc::double_v> const &point,
  Vector3D<Vc::double_v> const &direction) {

  Vc::double_v bestDistance = kInfinity;
  for (int i = 0; i < N; ++i) {
    Vc::double_v distance;
    distance = -(plane[0][i]*point[0] + plane[1][i]*point[1] +
                 plane[2][i]*point[2] + plane[3][i]);
    distance /= (plane[0][i]*direction[0] + plane[1][i]*direction[1]
                 plane[2][i]*direction[2]);
    bestDistance(distance < bestDistance) = distance;
  }
  return bestDistance;
}
```

Using traits to avoid code duplication

- Intensive kernels are developed in a generic way, using only trait-defined types and functions.
- Architecture-specific traits are created as needed, to associate generic types and functions with their arch-specific types.
- Appropriate backends are requested by #define'ing their macros needed at compilation, e.g.

  -DVECGEOM_VC or
  -DVECGEOM_CUDA

backend/cuda/Backend.h

```
namespace vecgeom {
#ifdef VECGEOM_NVCC
inline
#endif
namespace cuda {

struct kCuda {
  typedef int       int_v;
  typedef Precision precision_v;
  typedef bool      bool_v;
  typedef Inside_t  inside_v;
  const static bool early_returns = false;
  static constexpr precision_v kOne = 1.0;
  static constexpr precision_v kZero = 0.0;
  const static bool v_kTrue = true;
```

backend/vc/Backend.h

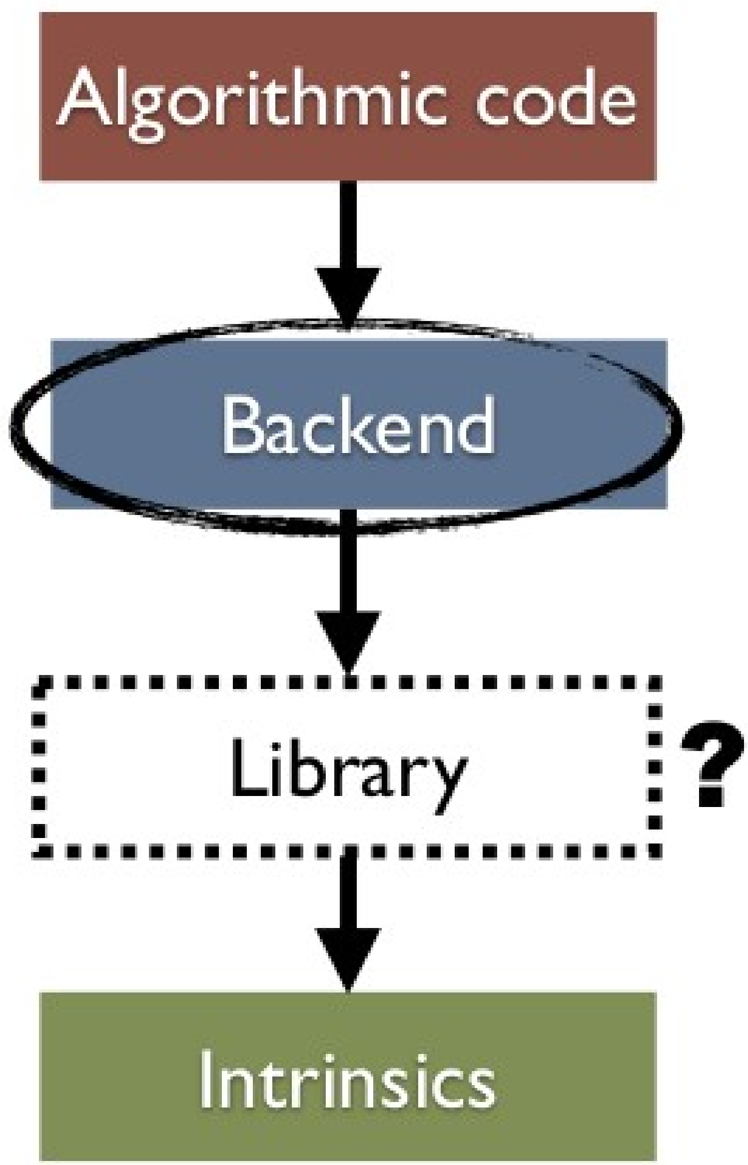
```
#include <Vc/Vc>

namespace vecgeom {
inline namespace VECGEOM_IMPL_NAMESPACE {

struct kVc {
  typedef Vc::int_v                       int_v;
  typedef Vc::Vector<Precision>           precision_v;
  typedef Vc::Vector<Precision>::Mask     bool_v;
  typedef Vc::Vector<int>                 inside_v;
  constexpr static bool early_returns = false;
  const static precision_v kOne;
  const static precision_v kZero;
```

Explicit vectorization

- Explicit SIMD vectorization can be implemented directly using intrinsics, but a vectorization library already brings many utilities pre-defined, like common math operators and functions.
- VecGeom currently works with Vc library, by Mathias Kretz, but other libraries can be easily plugged in (Agner Fog's VCL, Intel's VML, Cilk Plus, ...).
  A new backend is maybe all that is needed.



A generic kernel

```
template <int N>
template <class Backend>
VECGEOM_CUDA_HEADER_BOTH
typename Backend::Float_t Planes<N>::DistanceToOutKernel(
    double const (&plane)[4][N],
    Vector3D<typename Backend::Float_t> const &point,
    Vector3D<typename Backend::Float_t> const &direction) {

  typedef typename Backend::Float_t Float_t;
  typedef typename Backend::bool_v Bool_t;

  Float_t bestDistance = kInfinity;
  Float_t distance[N];
  Bool_t valid[N];
  for (int i = 0; i < N; ++i) {
    distance[i] = -(plane[0][i]*point[0] + plane[1][i]*point[1] +
                    plane[2][i]*point[2] + plane[3][i]);
    distance[i] /= (plane[0][i]*direction[0] + plane[1][i]*direction[1] +
                    plane[2][i]*direction[2]);
    valid[i] = distance[i] >= 0;
  }
  for (int i = 0; i < N; ++i) {
    MaskedAssign(valid[i] && distance[i] < bestDistance, distance[i],
                 &bestDistance);
  }
  return bestDistance;
}
```

The Backend, as discussed

Arithmetics just works!

MaskedAssign( ) is an optimized if( ) replacement

Shapes needed for CMS detector – Nov/2014 status

Shape	algorithms ready	GPU tested	unit-tests available	stress tests	Usolids-compatible	Root importer
Box	●	✔	✔	✔	✔	✔
Tube	●	✔	✘	✘	✘	✔
Cone	●	✔	✔	✔	✔	✔
Trapezoid	●	✔	✔	✔	✔	✔
Torus	◕	✔	✘	✘	✔	✔
Polyhedra	●	✘	✘	✘	✘	✘
Polycone	◔	✘	✘	✘	✘	✘
Composite shapes	◕	✔	✘	✘	✘	✔

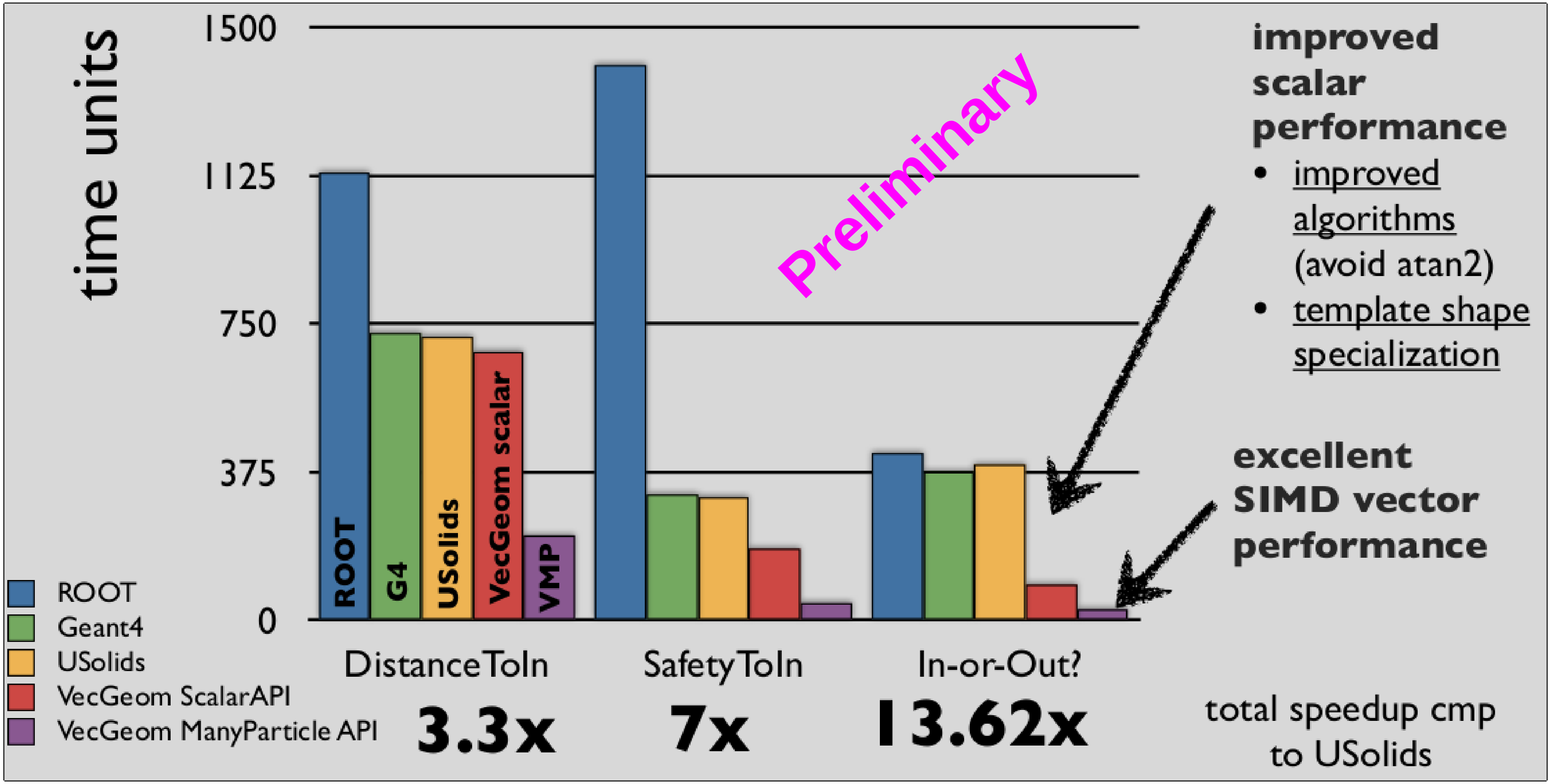
Shapes needed for CMS detector – Jan/2015 status

Shape	algorithms ready	GPU tested	unit-tests available	stress tests	Usolids-compatible	Root importer
Box	●	✔	✔	✔	✔	✔
Tube	●	✔	✔	✔	✔	✔
Cone	●	✔	✔	✔	✔	✔
Trapezoid	●	✔	✔	✔	✔	✔
Torus	●	✔	✔	✔	✔	✔
Polyhedra	●	✔	✔	✔	✔	✔
Polycone	●	✔	✔	✔	✔	✔
Composite shapes	●	✔	✔	✔	✔	✔

Preliminary performance

Our benchmarking tests can compare processing times for Geant4, Root and Usolids. Results shown below are based on ~ideal conditions, illustrating significant improvements due to the use of SIMD vectorization, but also a few other improvements.

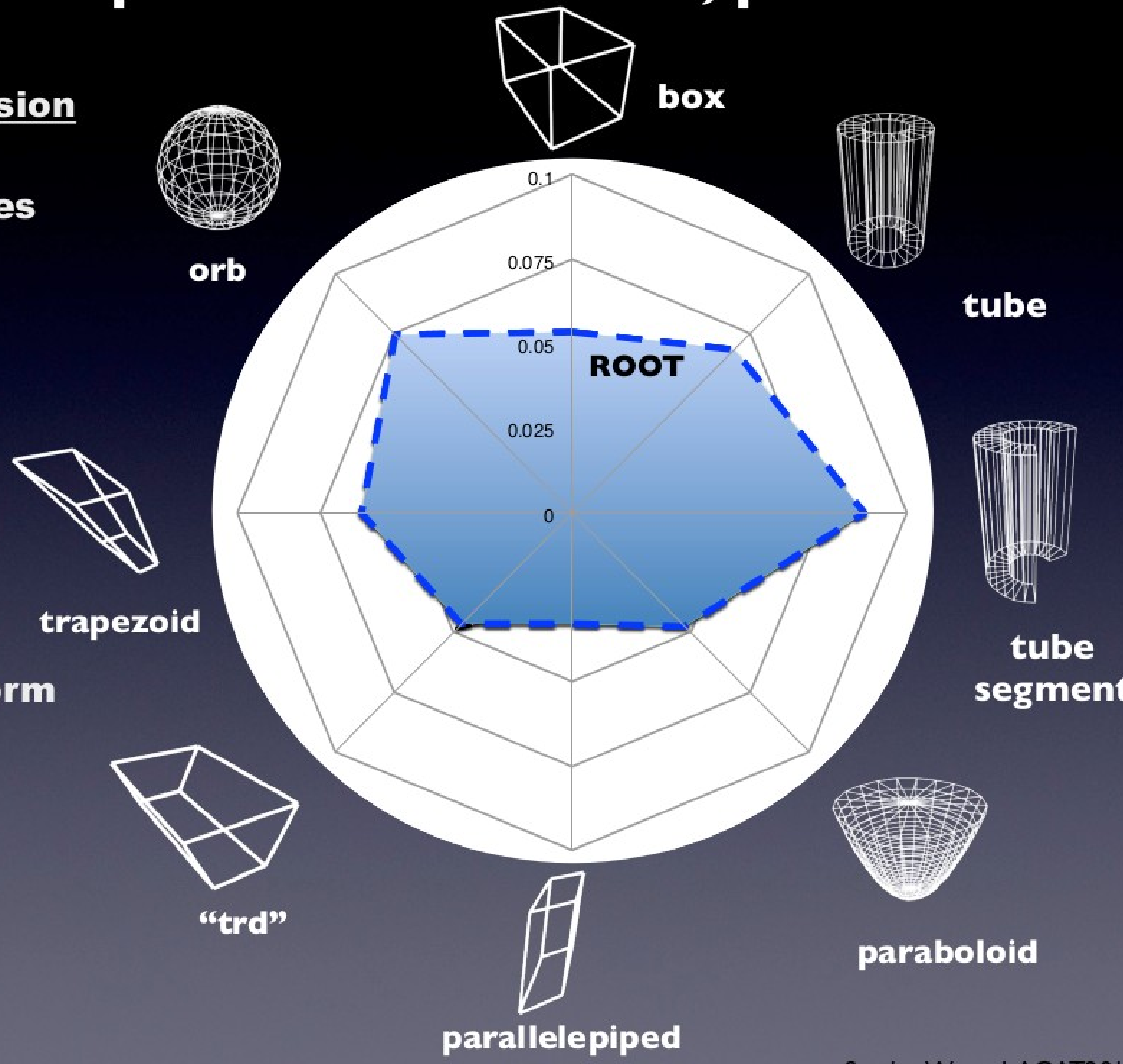
As an example: **tube shape**



Solid/shape implementation status; performance

timings for <u>collision detection</u> for various primitives

timing points form a polygon per library

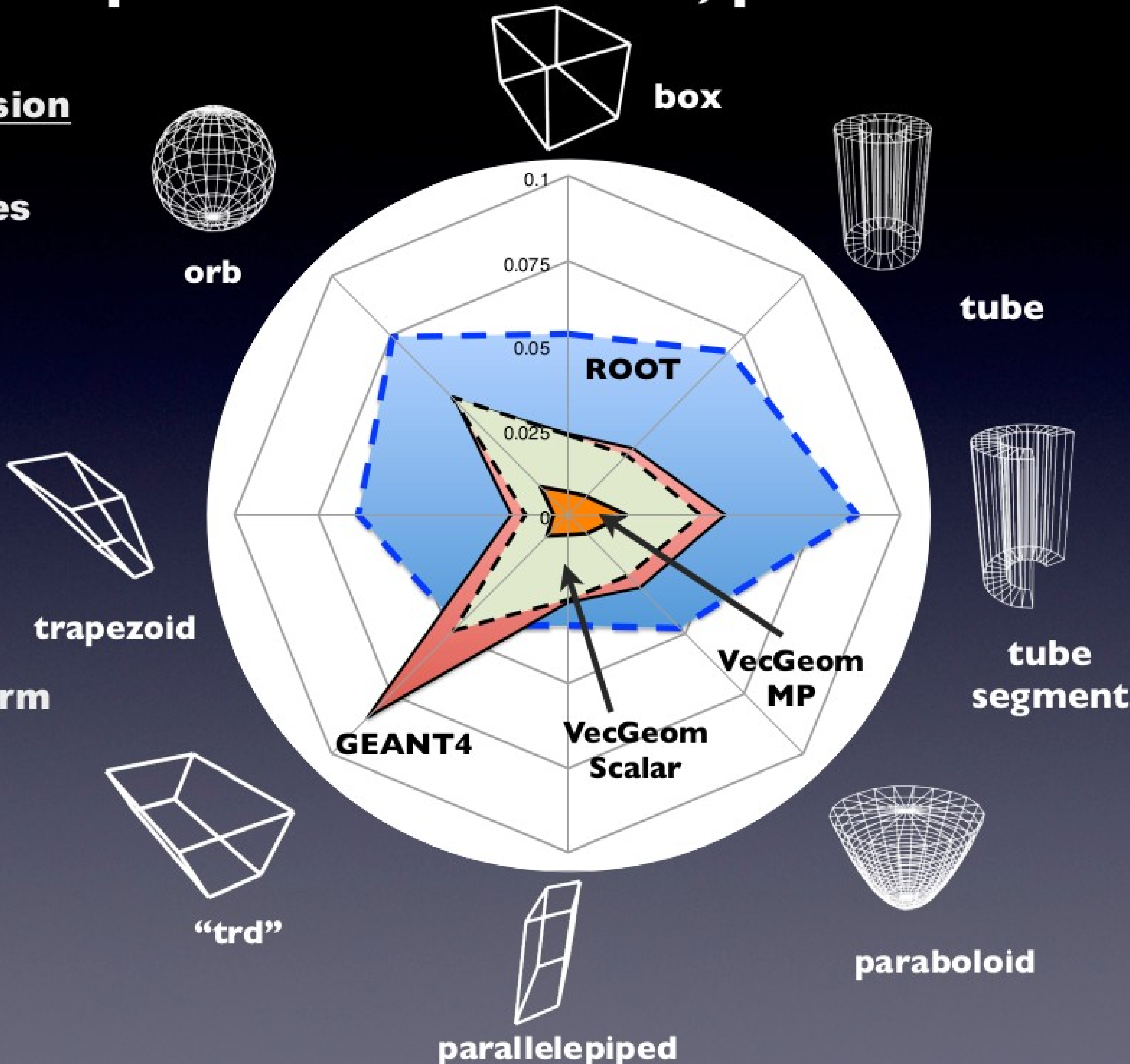


Solid/shape implementation status; performance

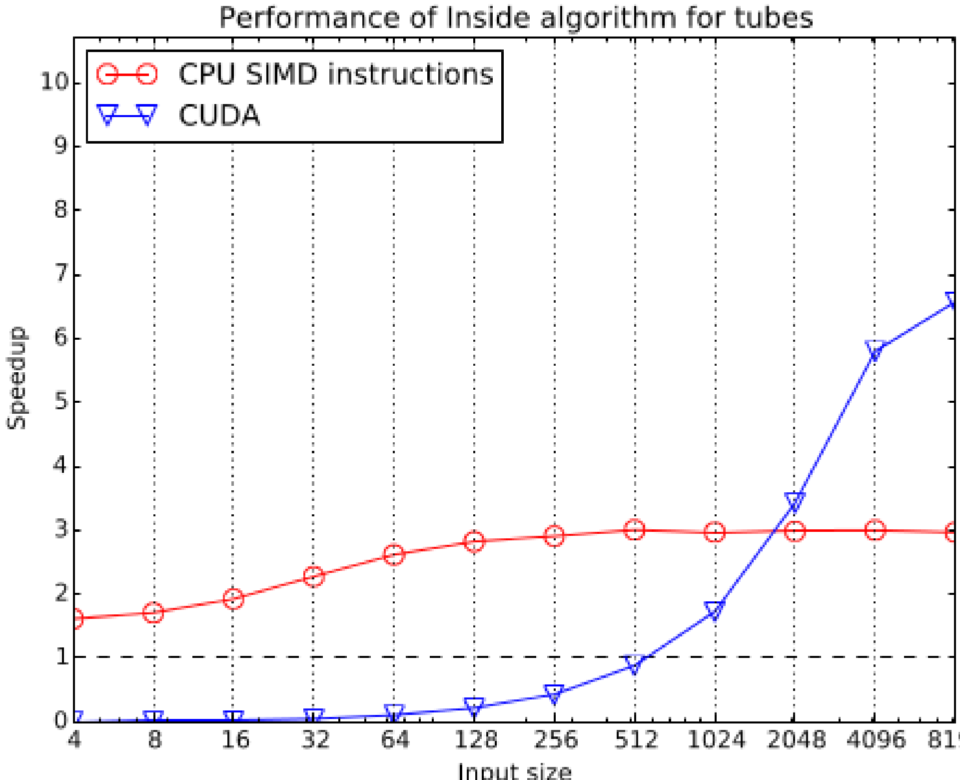
timings for <u>collision detection</u> for various primitives

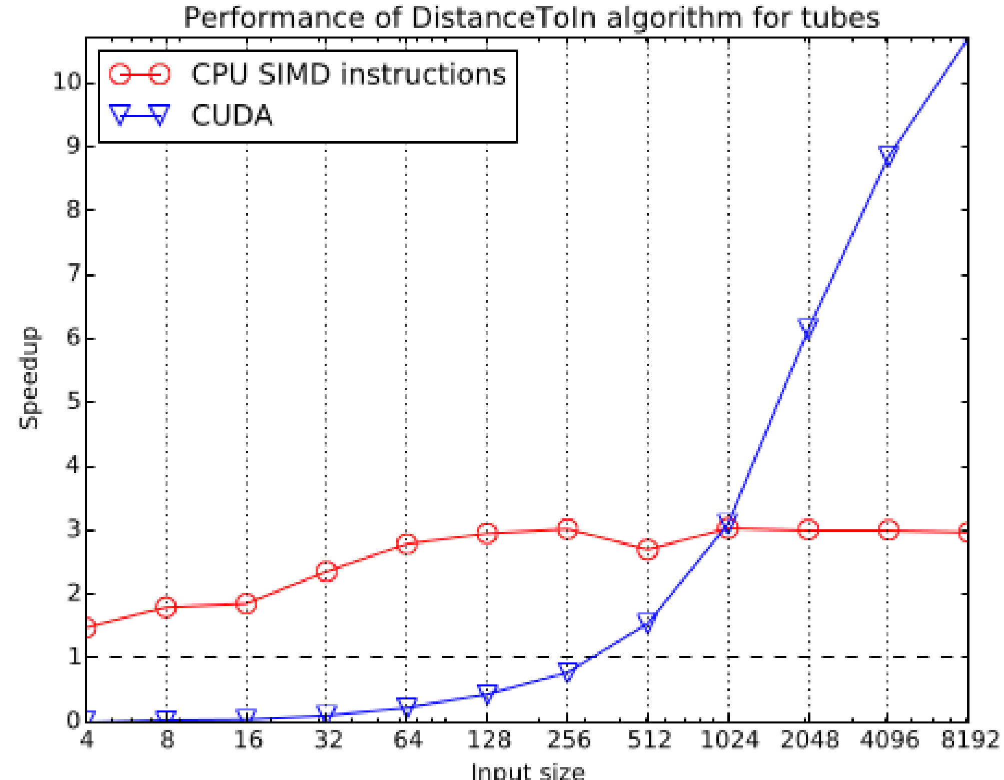
timing points form a polygon per library

smaller area = better library performance



Preliminary tests with GPUs





~10 (lower) FLOPS vs. ~50 (higher) FLOPS

* GPU comparisons is very preliminary, the normalization is not reliable yet
* SIMD vectorization provides excellent improvement, but saturates around ~3x speed-up
* GPUs require relatively large baskets to overtake the overhead due to data transfer, but it is still improving at large basket sizes (# tracks processed in parallel)
→ huge speed-ups are possible in special circumstances.

Summary & Outlook

- VecGeom is a detector geometry library prototype which demonstrates the concept of using a generic programming approach to implement fast, vectorized algorithms in multiple architectures, while keeping code duplication under control
- Current VecGeom algorithms show significant speed-ups with respect to existing implementations (Root, Geant4), due to the use of SIMD vectorizations (SSE, AVX)
- Much larger speed-ups may be obtained, in particular circumstances, using GPU-based systems. Use of hybrid systems?
- A simplified version of the CMS detector has been successfully used for small scale tests: a total of ~3,000 tracks from a handful ttbar events have been navigated through the geometry (no magnetic field at yet)
- The promising performance results shown, were obtained for a few shapes which have been through a first step of optimization after the vectorization. We are ready for a next, more thorough round of optimizations, to be extended to all CMS and other shapes.
- We are in the verge of a new paradigm in the HEP detector simulations. GeantV + VecGeom are the testbed for the R&D which will take us there.
- A lot more work is still needed, specially for the vectorization of physics processes – see next talk!

Acknowledgements...

- To the people involved on the VecGeom part of the GeantV project:
  - CERN: J.Apostolakis, G.Bitzes, G.Cosmo, J.de Fine Licht, A.Gheata, H.Kim, T.Nikitina, O.Shadura, S.Wenzel
  - Fermilab: P.Canal, G.Lima
  - BARC (India): A.Bhattacharyya, R.Sehgal
  - Univ. of Catania (Italy): M.Bandieramonte

References

- GeantV: http://geant.cern.ch
- Geant4: http://www.geant4.org
- Root: http://root.cern.ch
- Usolids: http://aidasoft.web.cern.ch/USolids
- CMS detector: http://cms.web.cern.ch
- Intel Xeon Phi: https://software.intel.com/en-us/mic-developer
- NVidia Cuda: https://developer.nvidia.com/cuda-zone
- Vc library: http://code.compeng.uni-frankfurt.de/projects/vc
- Agner fog VCL: http://www.agner.org/optimize/vectorclass.pdf
- intel Cilk: https://www.cilkplus.org
- OpenMP: http://www.openmp.org