Accelerating Navigation in the VecGeom Geometry Library

Sandro Wenzel / CERN
with Yang Zhang / KIT
for the VecGeom developers

CHEP16@San Francisco
12.10.2016
“Offer geometry library with API for vector transport in Geant-V … targeting use of SIMD from ground up”

“Effort to improve speed, algorithms, code, maintenance burden, …” of geometry code for the benefit of Geant4 / TGeo…”

VecGeom = Geometry Primitives (USolids) + Geometry Model / Navigation + Many-Particle API

gitlab.cern.ch/VecGeom/VecGeom
“The VecGeom Project

“Offer geometry library with API for vector transport in Geant-V … targeting use of SIMD from ground up”

“Effort to improve speed, algorithms, code, maintenance burden, …” of geometry code for the benefit of Geant4 / TGeo…”

VecGeom  =  Geometry Primitives (USolids)  
+ Geometry Model / Navigation
+ Many-Particle API

gitlab.cern.ch/VecGeom/VecGeom
“Offer geometry library with API for vector transport in Geant-V … targeting use of SIMD from ground up”

“Effort to improve speed, algorithms, code, maintenance burden, …” of geometry code for the benefit of Geant4 / TGeo…”

VecGeom = Geometry Primitives (USolids) + Geometry Model / Navigation + Many-Particle API

gitlab.cern.ch/VecGeom/VecGeom

Today: Acceleration the navigation module
- Explicit SIMD vectorization
- Navigator code specialization
- Focus on one-particle features as needed by Geant4 !!
Shape-Primitives Status: The ALICE Use-Case

- VecGeom now has all shape-primitives to satisfy needs of most HEP experiments (Xtruded added recently)
- For ALICE simulations (Pb-Pb), demonstrate that VecGeom offers very significant performance gains for the most CPU relevant shape-primitives

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Safety</th>
<th>Dist2In</th>
<th>Dist2Out</th>
<th>Contains</th>
<th>CPU% Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pgon</td>
<td>2.05</td>
<td>2.52</td>
<td>0.18</td>
<td>1.18</td>
<td>5.93</td>
</tr>
<tr>
<td>Xtru</td>
<td>0.56</td>
<td>0.68</td>
<td>0.20</td>
<td>1.81</td>
<td>3.25</td>
</tr>
<tr>
<td>Pcon</td>
<td>1.07</td>
<td>0.32</td>
<td>0.05</td>
<td>0.13</td>
<td>1.57</td>
</tr>
</tbody>
</table>

% of CPU cost of shape primitives (TGeo) in typical ALICE Pb-Pb simulation
VecGeom now has all shape-primitives to satisfy needs of most HEP experiments (Xtruded added recently)

For ALICE simulations (Pb-Pb), demonstrate that VecGeom offers very significant performance gains for the most CPU relevant shape-primitives

- Depending on experiment, a few % in CPU simulation cost gainable by switching to VecGeom primitives (integration effort into G4/TGeo under way)
Geometry primitives provide algorithms for simple ray-shape problems (focus on individual object)
- Geometry primitives provide algorithms for simple ray - shape problems (focus on individual object)

- **Navigation module** provides "multi-object" algorithms:
  - provides next colliding object + distance in a "multi-object" scene
  - provide object after the next boundary crossing
  - simulations spend significant time in navigation module (ALICE ~30% with TGeo, similar in CMS, …)
- Geometry primitives provide algorithms for simple ray - shape problems (focus on individual object)

- **Navigation** module provides “multi-object” algorithms:
  - provides next colliding object + distance in a “multi-object” scene
  - provide object after the next boundary crossing
  - simulations spend significant time in navigation module (ALICE ~30% with TGeo, similar in CMS, …)

- **Goals / Targets:**
  - Implement navigation system in VecGeom
  - Implement acceleration structures for fast candidate rule-out (scaling ~log(N) - see voxel techniques of G4/TGeo)
  - Target explicit SIMD acceleration
Canonical solution for fast hit-detection: tree structures, lookup structures, bounding boxes, …

How to combine this with SIMD paradigm?
Canonical solution for fast hit-detection: tree structures, lookup structures, bounding boxes, …

How to combine this with SIMD paradigm?

Followed idea based on using (aligned) bounding boxes of geometry objects to filter good hit candidates
Canonical solution for fast hit-detection: tree structures, lookup structures, bounding boxes, …

How to combine this with SIMD paradigm?

Followed idea based on using (aligned) bounding boxes of geometry objects to filter good hit candidates

```
ray bounding box intersection test
```

“done in same CPU time“
SIMD Acceleration of „Voxel“ Navigation

- Canonical solution for fast hit-detection: tree structures, lookup structures, bounding boxes, …
- How to combine this with SIMD paradigm?
- Followed idea based on using (aligned) bounding boxes of geometry objects to filter good hit candidates

get **SIMD gain** from treating **group of boxes** in parallel

get **scaling** from **hierarchies** of bounding box groups (forming **regular trees**)

Inspired from e.g.: Shallow bounding volume hierarchies for fast SIMD ray tracing of incoherent rays (DOI: 10.1111/j.1467-8659.2008.01261.x) + CPU ray-tracing libraries: Intel Embree, …
Basic algorithm:
- let $S ==$ elements in SIMD register
- cluster objects into groups of $S$ objects (we use a variation of k-means)
- identify bounding boxes of grouped objects as daughters of a tree node
- iterate this process

Algorithm illustrated here for SSE (= 2 double numbers per register)
Benchmark SIMD-Trees: Local Benchmark

- Test approach on various detector volumes
  - most important complex volumes from ALICE: ALIC + TPC_Drift
  - a complex volume from CMS: MBWheel (~600 daughter volumes)

- Perform local navigation benchmark*: One step + boundary crossing in the given volume for 0.5 million different tracks

<table>
<thead>
<tr>
<th>Volume</th>
<th>Daughters</th>
<th>G4</th>
<th>TGeo</th>
<th>VecGeom (SSE4.2)</th>
<th>VecGeom (AVX2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIC (ALICE)</td>
<td>65</td>
<td>0.74</td>
<td>1.07</td>
<td>0.30</td>
<td>0.23</td>
</tr>
<tr>
<td>TPC_Drift (ALICE)</td>
<td>641</td>
<td>14</td>
<td>2.2</td>
<td>1.2</td>
<td>0.9</td>
</tr>
<tr>
<td>MBWheel (CMS)</td>
<td>~600</td>
<td>0.84</td>
<td>1.09</td>
<td>0.49</td>
<td>0.35</td>
</tr>
</tbody>
</table>

numbers are time in seconds; worst is red; best is blue

- Demonstrating overall speedup >2x in navigation compared to existing solutions
- Demonstrating gain from SIMD vector unit (see change SSE4.2 to AVX2)

*All packages used in standard release mode without particular tuning
Testing VecGeom in “Toy” Simulations

- Evaluate VecGeom (solids + navigation) on complex modules for multiple steps

- “XRayBenchmarker” test:
  - follow geantinos through geometry - pixel by pixel
  - record some information on screen behind object
  - do same with G4 / TGeo

- perfect for validation of navigation algorithms

- good to get a global idea of library performance
Example for ALICE ITSSPD module

- Perfect agreement between G4/TGeo/VecGeom
- Observe generally factors > 2.6x speed improvement against other packages
- Another indication of global performance advantage of VecGeom
“Production” … to current R&D
Further R&D in Navigation Optimization

- VecGeom offers faster navigation compared to G4/TGeo!! … Can we do more? (other than different acceleration structures)?
Further R&D in Navigation Optimization

- VecGeom offers faster navigation compared to G4/TGeo!! ... Can we do more? (other than different acceleration structures)?

- Traditional simulation navigation algorithms are still ...
Further R&D in Navigation Optimization

- VecGeom offers faster navigation compared to G4/TGeo!! … Can we do more? (other than different acceleration structures)?

- Traditional simulation navigation algorithms are still …

- … too generic
  - runtime polymorphic approach
    - in particular: no internal vectorization possible
Further R&D in Navigation Optimization

- VecGeom offers faster navigation compared to G4/TGeo!! … Can we do more? (other than different acceleration structures)?

- Traditional simulation navigation algorithms are still …

- … too generic
  - runtime polymorphic approach
    - in particular: no internal vectorization possible

- … poorly exploiting structural and static information about a scene
  - no usage of boundary touching relations between objects
  - no fast lookup of global-local transformations for placed entities
  - etc…

Example: Static geometry analysis can reveal that object1 only touches object2; tracks leaving 1 never have to be checked against 3 for relocating.
Further R&D in Navigation Optimization

- VecGeom offers faster navigation compared to G4/TGeo!! … Can we do more? (other than different acceleration structures)?

- Traditional simulation navigation algorithms are still …

- … too generic
  - runtime polymorphic approach
  - in particular: no internal vectorization possible

- … poorly exploiting structural and static information about a scene
  - no usage of boundary touching relations between objects
  - no fast lookup of global-local transformations for placed entities
  - etc…

Example: Static geometry analysis can reveal that object1 only touches object2; tracks leaving 1 never have to be checked against 3 for relocating

- HEP detectors are pretty static objects; most things are known at compile time or constant during (long) run-time

- Opportunity to pre-analyse + pre-compute + compile-time optimize

- R&D goal: Exploit these opportunities via volume-specialized navigator algorithms produced via automatic C++ code generation
A prototype service to generate volume-specialized navigator algorithms has been implemented
- considerably reduced virtual functions
- reduce time spent in coordinate transformation (via compile-time lookup structures)
- put static neighbourhood information for fast relocation
A prototype service to generate volume-specialized navigator algorithms has been implemented:

- considerably reduced virtual functions
- reduce time spent in coordinate transformation (via compile-time lookup structures)
- put static neighbourhood information for fast relocation

Can be embedded into a (JIT) workflow of a simulation.

**Diagram:**

- Small Simulation
  - extract important geometry volumes (in terms of number of steps)
- Volume Name
- Geometry Description
- "NavigationSpecializer"
- Generated Navigator class (as C++ code)
- (Continue) Long Simulation
  - (dynamic) hook-in
  - libNavigators.so
- Compile into
Specialized Navigator Improvement: Benchmark examples

- Extracted important (“showering”) volumes (in terms of number of steps) in an ALICE Pb-Pb simulation
- Measure time to perform a “step” in these volumes

<table>
<thead>
<tr>
<th>Volume</th>
<th>G4</th>
<th>TGeo</th>
<th>VecGeom General</th>
<th>VecGeom Specialized</th>
<th>EXTRA SPEEDUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZNST</td>
<td>0.24</td>
<td>0.28</td>
<td>0.10</td>
<td>0.06</td>
<td>1.67</td>
</tr>
<tr>
<td>ZPST</td>
<td>0.25</td>
<td>0.29</td>
<td>0.11</td>
<td>0.06</td>
<td>1.83</td>
</tr>
<tr>
<td>DCML</td>
<td>0.24</td>
<td>0.28</td>
<td>0.12</td>
<td>0.06</td>
<td>2.00</td>
</tr>
<tr>
<td>voRBCuTube</td>
<td>0.16</td>
<td>0.24</td>
<td>0.10</td>
<td>0.06</td>
<td>1.67</td>
</tr>
<tr>
<td>ZNGx</td>
<td>0.09</td>
<td>0.18</td>
<td>0.06</td>
<td>0.03</td>
<td>2.00</td>
</tr>
<tr>
<td>AFaGraphiteCone</td>
<td>0.74</td>
<td>0.36</td>
<td>0.11</td>
<td>0.03</td>
<td>3.67</td>
</tr>
</tbody>
</table>

numbers are time in seconds; worst is red; best is blue

- Navigator specialization delivers extra speedup kick; making gain compared to G4/TGeo even more significant
- In context of Geant-V: This technique reduces the non-vectorizable parts of navigation and makes multi-particle SIMD gains possible or more efficient.
Presented advances in navigation module of VecGeom

Have (first) SIMD enabled navigation acceleration structures in production:

- Shown to generally outperforming existing solutions in Geant4/TGeo

Showed an avenue to further improve navigation performance

- By automatic navigator code specialization (R&D prototype status)
- Lot’s of work to be done…

“VecGeom navigation is more than an interesting alternative to Geant4/TGeo navigation and could be beneficial to simulation frameworks NOW”
Backup section
SIMD Vectorization

- Achieved via explicit SIMD programming using C++ wrapping libraries such as Vc, UMESIMD, …
- Using our abstraction layer “VecCore”

Navigation System

- In contrast to G4/TGeo; VecGeom navigator classes are state-less, facilitating easy use for multi-threading and multi-particle queries
- Navigation state is carried in separate classes which become part of a track - property

(Preliminary) Impact of VecGeom for Geant-V

- VecGeom is used as one of the native navigation system in Geant-V (the other being TGeo)
- VecGeom reduces overall simulation memory consumption over TGeo by factors > 2
- measuring a “simulation speed gain” over TGeo by factors > ~2

Benchmark setup:

- All benchmarks presented here were run with tag “W40-16” of VecGeom
- Benchmark machine: Intel(R)-Core(TM) i7-5930K running CERN CC7
- compiler gcc4.8.5
- Vc 1.2.0 backend with native (=AVX2) instruction set (unless otherwise specified)