High performance geometry
-- ideas for future direction
(or reasons to start from scratch)
--

Sandro Wenzel / CERN-PH-SFT

meeting at Fermilab, 21.1.2013

What is current status?

- activity since spring 2013 focused on studying feasibility of vectorizing (primitive) geometry kernels
- demonstrated for a couple of shapes (box, tube, cone, tubeseg, coneseg) that this is very possible indeed with good performance gains
- this came at the cost of totally rewriting the routines to make them vector friendly
- programming model: Vc, Intel Cilk Plus (array notation)
- performance example on CPU:
  - (simplified) navigation of particles in a logical volume with daughter shapes
    - CHEP13: **max speedup of 3.1**
    - current status: **max speedup > 4 (with techniques discussed further down)**

goals / challenges ahead

- We should now start a systematic effort to produce a "production ready" library
- Goals:
  - provide a library with vectorized interfaces for important geometry kernels
    - vectorization over particles, shapes
  - provide a library with CUDA/OpenCL kernels for important geometry functions
  - (provide vectorized 1-particle functions)
  - achieve best performance
- main challenges ahead (from my point of view):
  - current code does not serve for vectorization or SIMT -- there are just too many branch levels (see for instance tube -> distanceToIn in Usolids)
  - hence, **total code rewrite necessary** (regardless of starting point: ROOT or USOLIDS)
  - **complete revalidation necessary**

challenges continued ... / implications

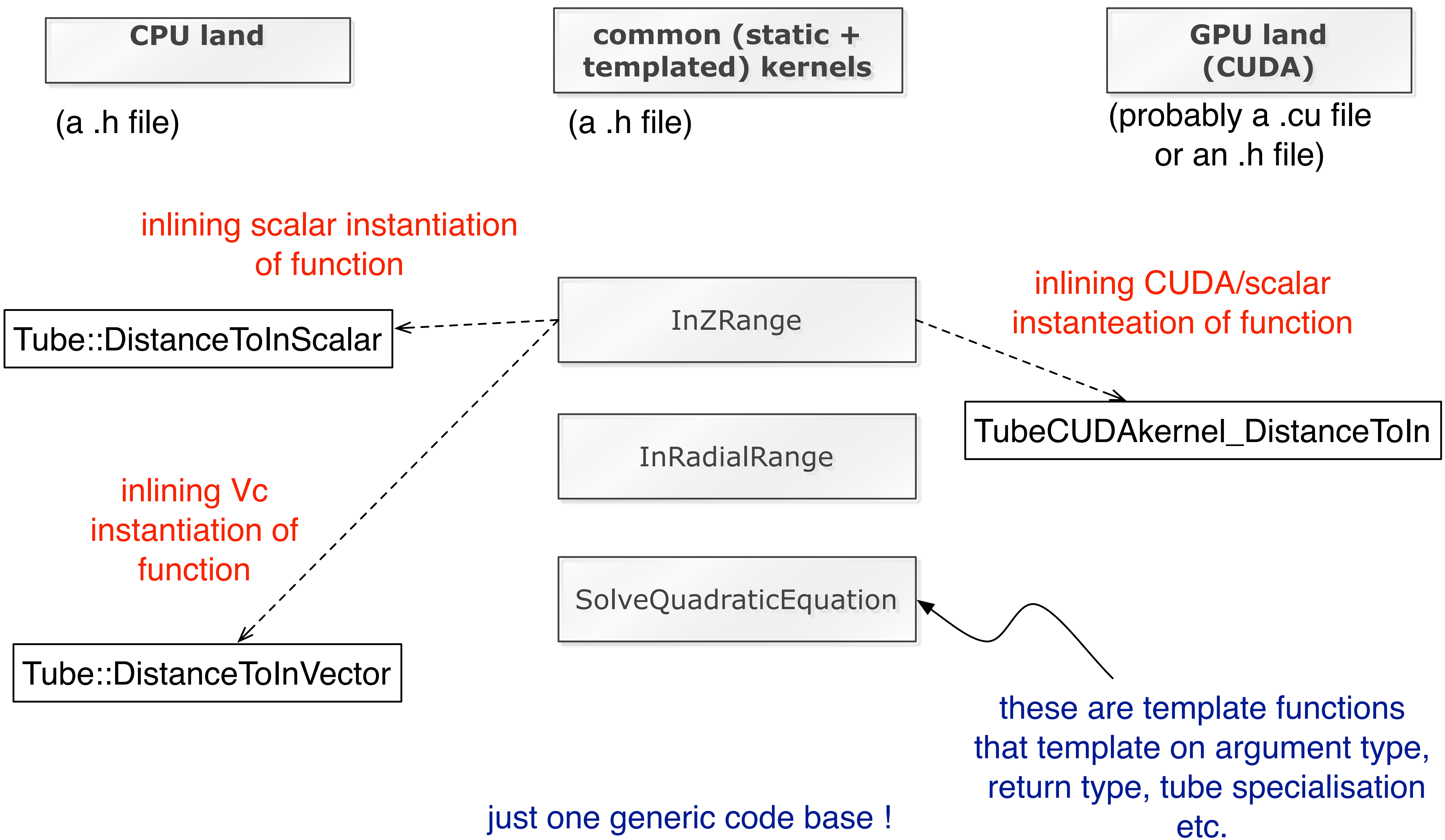
- targeting different backends (vector (Vc, CilkPlus), GPU, scalar) sounds like a lot of code repetition if we continue to code the way it was done in the past
  - will be a nightmare for maintenance and testing
- We should hence (these points are related)
  - write code which is **generic**
    - kernels which work with scalar or vector arguments
  - **reuse code as much as possible without performance loss**
    - example: many kernels for tube / cone / polycone are shared and should be written only once (without function calls)
    - **write code which is composeable of smaller kernels**

my general proposition

- a **templated library** is a good ansatz to solve the challenges presented:
  - you can write generic code easily with template functions
  - you automatically write easily inlinable / reusable code since templates require coding in header files

- a **templated library** is perfect to achieve good performance:
  - template class specialization allows to produce very optimized code for particular shapes / matrices, etc.
    - **example I**: tube example from slides before Christmas
    - **example II**: matrix transform specialization
    - average gain ~20% compared to non-specialized code with runtime branches
    - makes vectorization much more efficient

Sketch of generic code idea



Sandro Wenzel

Very first prototype

- first prototype using these ideas exists
- currently accessible for anyone one github (VecGeom)
  - https://github.com/sawenzel/VecGeom.git
  - asked for repository at CERN
- shapes implemented: box, tube (all variants), cones (all variants), polycone + some navigation methods
- can repeat the benchmark from CHEP13
- contains branch demonstrating generic generation of CUDA, Vc and scalar functions out of same template functions
  - our technical student (Johannes) successfully ran first tests on CUDA and CPU
- should sit down in a working group to look at this code ...

My expectations for this week

- **hearing CUDA ideas and your requirements**
  - **do you need a kernel for every shape primitive or for just for some**
  - **scope of kernels**
  - **virtual function problem**
- study of the prototype and decision of how to proceed
- setup of a common workplan and milestones
- coding conventions
- setup of a plan to integrate this work (step by step)
- setup of a plan to test this work

compatibility etc.

- USolids was started as a unified solid library ....
- ideally the vectorized work should become parts of USolids
- however coding ansatz completely orthogonal to USolids at the moment
- VecGeom could become USolids2.0 / UGeom ???
- We should definitely use the interfaces of USolids to start with