Fermilab

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

Trapezoid optimizations and performance studies

Guilherme Lima (Fermilab)

GeantV Weekly Meeting
March 28, 2017

Trapezoid algorithms

- The trapezoid has 6 planar faces:
- two faces are perpendicular to z-axis at +/-fDz → trivial normals are (0,0,+/-1) like for the box)
- four side planes (non-trivial normals)
- Distances are calculated based on 2 dot products perf face, between face normals and positions or directions
- In order to save dot product calculations, add inside/outside checks and try to return early
- Two trap configurations: PLANESHELL=ON/OFF

Box algorithms are very simple

- Guilherme Amadio has improved box algorithms into just a few lines, thanks to trivial box normals

```
template <typename Real_v>
VECGEOM_FORCE_INLINE
VECGEOM_CUDA_HEADER_BOTH
static Vector3D<Real_v> HalfSize(const UnplacedStruct_t &box)
{
  return Vector3D<Real_v>(box.fDimensions[0], box.fDimensions[1], box.fDimensions[2]);
}

template <typename Real_v, typename Bool_v>
VECGEOM_FORCE_INLINE
VECGEOM_CUDA_HEADER_BOTH
static void Contains(UnplacedStruct_t const &box, Vector3D<Real_v> const &point, Bool_v &inside)
{
  inside = (point.Abs() - HalfSize<Real_v>(box)).Max() < Real_v(0.0);
}

template <typename Real_v, typename Inside_v>
VECGEOM_FORCE_INLINE
VECGEOM_CUDA_HEADER_BOTH
static void Inside(UnplacedStruct_t const &box, Vector3D<Real_v> const &point, Inside_v &inside)
{
  Real_v dist = (point.Abs() - HalfSize<Real_v>(box)).Max();

  inside = vecCore::Blend(dist < Real_v(0.0), Inside_v(kInside), Inside_v(kOutside));
  vecCore__MaskedAssignFunc(inside, Abs(dist) < Real_v(kTolerance), Inside_v(kSurface));
}
```

Box algorithms are very simple

```
template <typename Real_v>
VECGEOM_FORCE_INLINE
VECGEOM_CUDA_HEADER_BOTH
static void DistanceToOut(UnplacedStruct_t const &box, Vector3D<Real_v> const &point,
                          Vector3D<Real_v> const &direction, Real_v const & /* stepMax */, Real_v &distance)
{
  const Vector3D<Real_v> invDir(Real_v(1.0) / NonZero(direction[0]), Real_v(1.0) / NonZero(direction[1]),
                                Real_v(1.0) / NonZero(direction[2]));

  const Real_v distIn = Max((-Sign(invDir[0]) * box.fDimensions[0] - point[0]) * invDir[0],
                            (-Sign(invDir[1]) * box.fDimensions[1] - point[1]) * invDir[1],
                            (-Sign(invDir[2]) * box.fDimensions[2] - point[2]) * invDir[2]);

  const Real_v distOut = Min((Sign(invDir[0]) * box.fDimensions[0] - point[0]) * invDir[0],
                             (Sign(invDir[1]) * box.fDimensions[1] - point[1]) * invDir[1],
                             (Sign(invDir[2]) * box.fDimensions[2] - point[2]) * invDir[2]);

  distance = vecCore::Blend(distIn > distOut || distIn > Real_v(kTolerance), Real_v(-1.0), distOut);
}

template <typename Real_v>
VECGEOM_FORCE_INLINE
VECGEOM_CUDA_HEADER_BOTH
static void SafetyToIn(UnplacedStruct_t const &box, Vector3D<Real_v> const &point, Real_v &safety)
{
  safety = (point.Abs() - HalfSize<Real_v>(box)).Max();
}
```

Can we simplify trapezoid algorithms, inspired on the box ones?

Simpler trapezoid algorithms?

- Think of extensions from box algorithms

```
template <typename Real_v>
VECGEOM_FORCE_INLINE
VECGEOM_CUDA_HEADER_BOTH
static void DistanceToOut(UnplacedStruct_t const &unplaced, Vector3D<Real_v> const &point,
                          Vector3D<Real_v> const &dir, Real_v const &stepMax, Real_v &distance)
{
  (void)stepMax;
  using Bool_v = vecCore::Mask_v<Real_v>;

  // step 0: if point is outside any plane --> return -1, otherwise initialize at Infinity
  Bool_v outside = Abs(point.z()) > MakePlusTolerant<true>(unplaced.fDz);
  distance       = vecCore::Blend(outside, Real_v(-1.0), InfinityLength<Real_v>());
  Bool_v done(outside);
  if (vecCore::EarlyReturnAllowedAt<Real_v>(EARLYRETURNTHR) && vecCore::MaskFull(done)) return;

  //
  // Step 1: find range of distances along dir between Z-planes (smin, smax)
  //

  Real_v distz = (Sign(dir.z()) * unplaced.fDz - point.z()) / NonZero(dir.z());
  vecCore__MaskedAssignFunc(distance, !done && dir.z() != Real_v(0.), distz);

//
// Step 2: find distances for intersections with side planes.
//

#ifndef VECGEOM_PLANESHELL_DISABLE
    Real_v disttoplanes = unplaced.GetPlanes()->DistanceToOut(point, dir);
    vecCore::MaskedAssign(distance, disttoplanes < distance, disttoplanes);

#else
    //=== Here for VECGEOM_PLANESHELL_DISABLE

    // loop over side planes - find pdist,Proj for each side plane
    Real_v pdist[4], proj[4], vdist[4];
```

Still to be dropped?!

This is much simpler than before!

See next page…

rmilab

Simpler trapezoid algorithms?

```
#else
    //=== Here for VECGEOM_PLANESHELL_DISABLE

    // loop over side planes - find pdist,Proj for each side plane
    Real_v pdist[4], proj[4], vdist[4];
    // Real_v dist1(distance);
    // EvaluateTrack<Real_v>(unplaced, point, dir, pdist, proj, vdist);

    TrapSidePlane const *fPlanes = unplaced.GetPlanes();
    for (unsigned int i = 0; i < 4; ++i) {
      // Note: normal vector is pointing outside the volume (convention), therefore
      // pdist>0 if point is outside  and  pdist<0 means inside
      pdist[i] = fPlanes[i].fA * point.x() + fPlanes[i].fB * point.y() + fPlanes[i].fC * point.z() + fPlanes[i].fD;

      // Proj is projection of dir over the normal vector of side plane, hence
      // Proj > 0 if pointing ~same direction as normal and Proj<0 if pointing ~opposite to normal
      proj[i] = fPlanes[i].fA * dir.x() + fPlanes[i].fB * dir.y() + fPlanes[i].fC * dir.z();

      vdist[i] = -pdist[i] / NonZero(proj[i]);
    }

    // early return if point is outside of plane
    // for (unsigned int i = 0; i < 4; ++i) {
    //   done = done || (pdist[i] > MakePlusTolerant<true>(0.));
    // }
    // vecCore::MaskedAssign(dist1, done, Real_v(-1.0));
    // if (vecCore::EarlyReturnAllowedAt<Real_v>(EARLYRETURNTHR) && vecCore::MaskFull(done)) return;

    for (unsigned int i = 0; i < 4; ++i) {
      // if track is pointing towards plane and vdist<distance, then distance=vdist
      // vecCore__MaskedAssignFunc(dist1, !done && proj[i] > 0.0 && vdist[i] < dist1, vdist[i]);
      vecCore__MaskedAssignFunc(distance, pdist[i] > MakePlusTolerant<true>(0.), Real_v(-1.0));
      vecCore__MaskedAssignFunc(distance, proj[i] > 0.0 && -Sign(pdist[i]) * vdist[i] < distance,
                                -Sign(pdist[i]) * vdist[i]);
    }
#endif
  }
```

Considering EvaluateTrack<>() function to replace a whole loop

Block was replaced with 1 line

Performance comparisons

Five commits are compared:

**** 68c48276** – tag v0.3.rc*

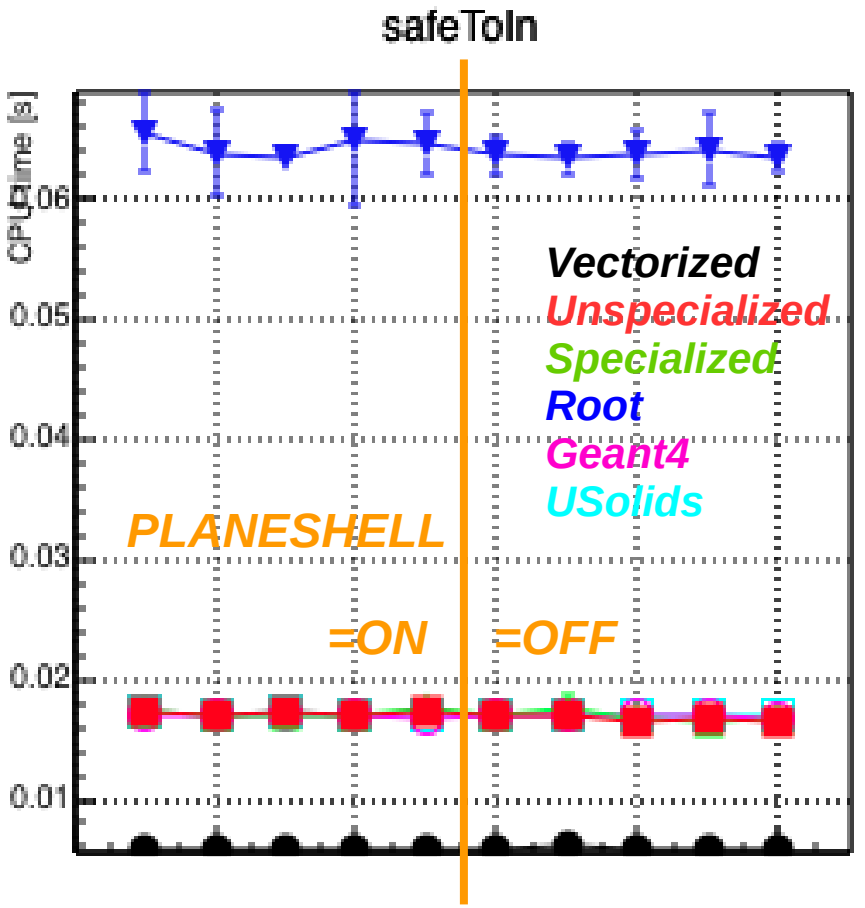
**** 4e5928c1** – tag v00.03.00*

**** 019d29a0** – DistToIn( ) and DistToOut( ) improvements (merge request #384)*

**** b356bad1** – improvs to NormalKernel( ) (merge request #415)*

**** 2c007289** – Improvs to Contains( ), Inside( ) and SafetyToOut( )*

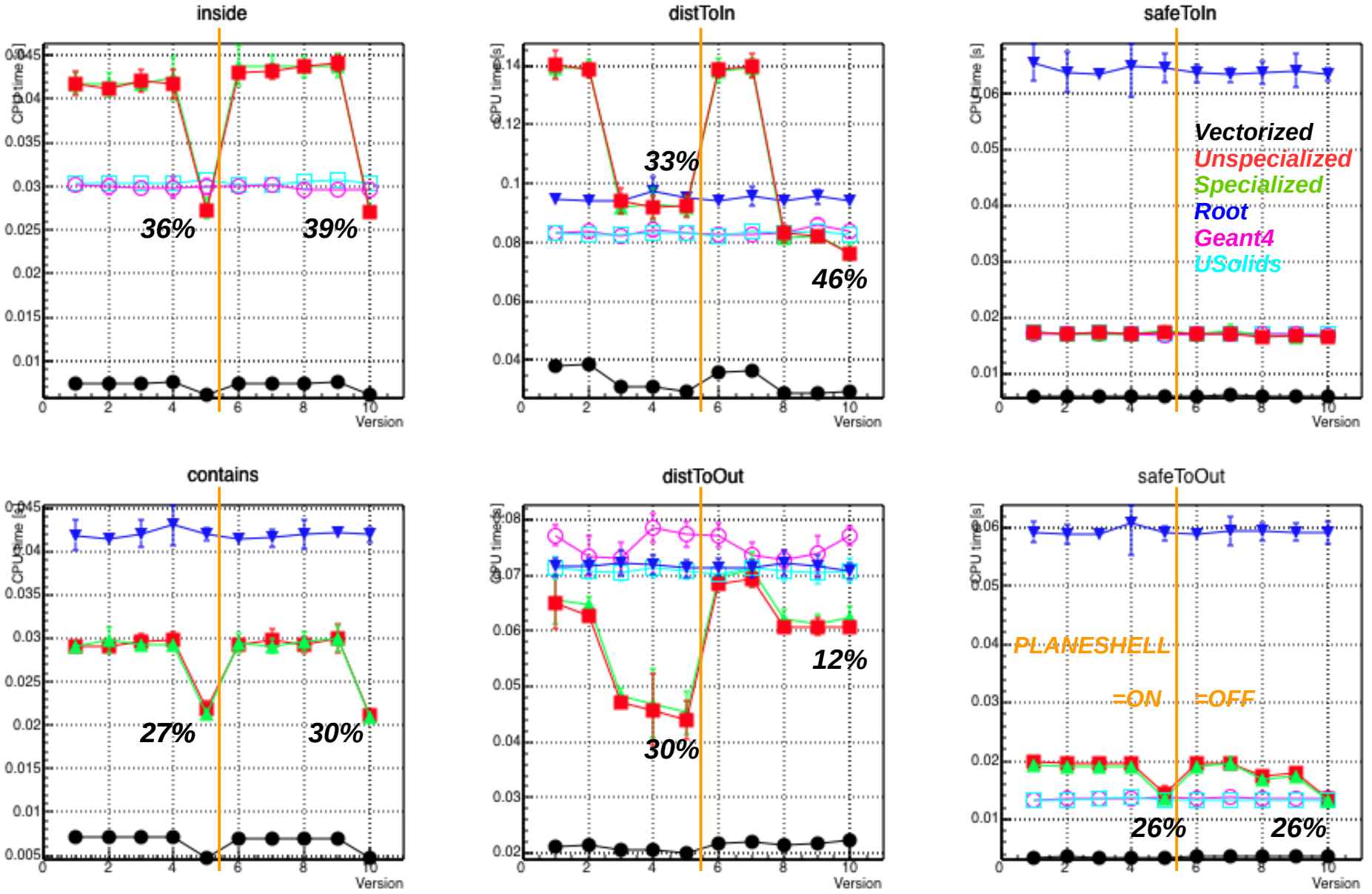
Measurements taken on my MacBook Pro, with most programs disabled.



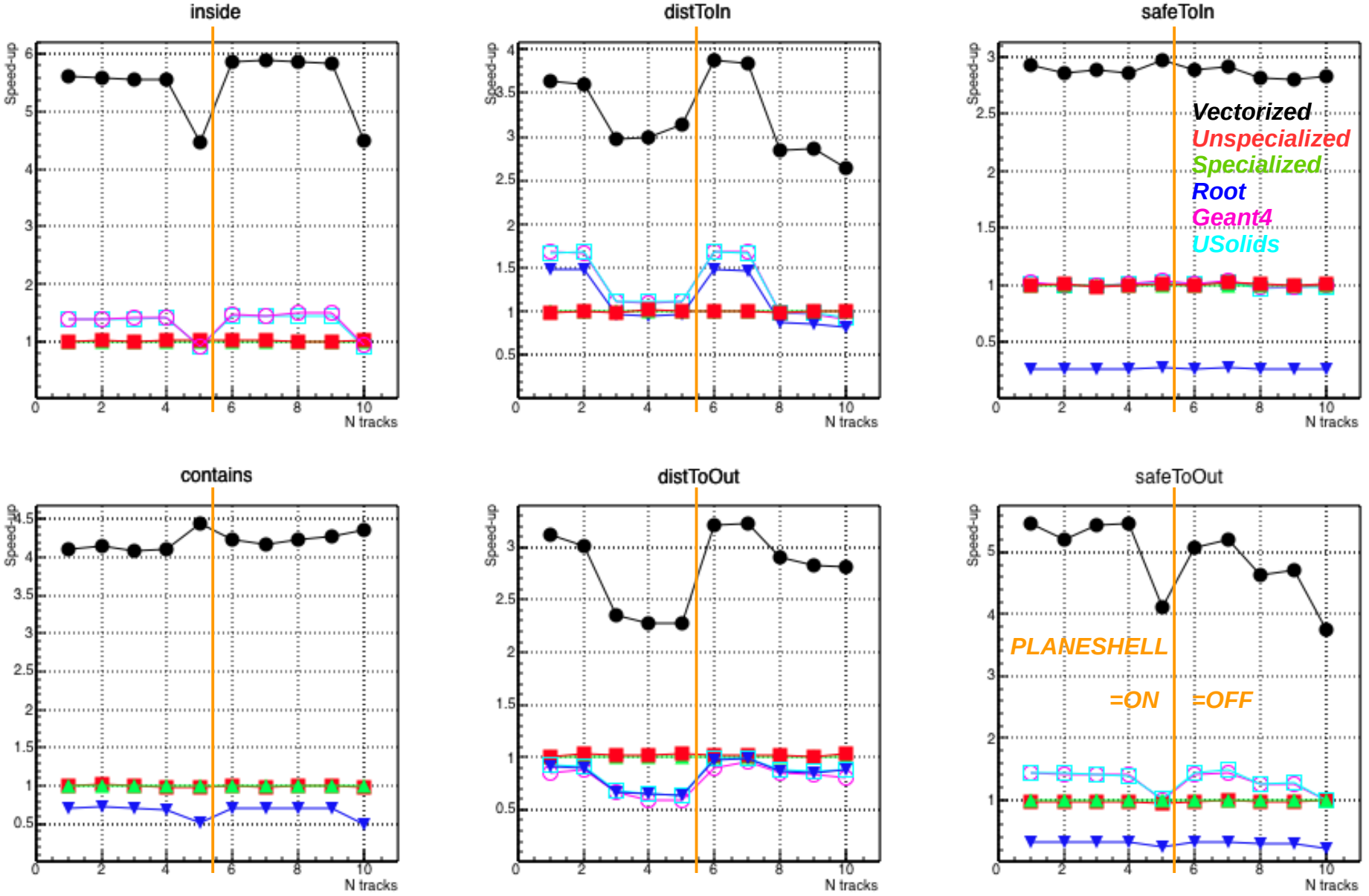
Speedups defined w.r.to Specialized algorithm, e.g. speedup = Vect.time / Spec.time

Each data point define as an average of slowest 20 / 21 data points (largest measurement is discarded in averaging). Error bars represent standard deviations.

Performance comparisons: CPU times



Performance comparisons: speedups



Note that improvements to scalar algorithm brought speedups down to ~4x (expected for AVX)

Performance improvement summary

- Significant performance improvements achieved by simplifying trapezoid algorithms as expanded box algorithms
- Vectorized speedups significantly above expected value (e.g. 4 for AVX) probably mean that scalar algorithms have room for optimization
- Some improvements come from careful use of early returns (see next section!)

Performance effects of early returns

- VecCore::EarlyReturnAllowed() is defined as
  always true for CPUs vs. always false for GPUs
- My previous experience (poorly documented) was that such definition introduces opposing effects on scalar and vectorized algorithms.
- To document this effect, a new templated earlyReturn function is defined (e.g. for T=Real_v):

```
template <typename T>
VECCORE_FORCE_INLINE
VECCORE_ATT_HOST_DEVICE
constexpr Bool_s EarlyReturnAllowedAt(size_t vecSize)
{
#ifdef VECCORE_CUDA_DEVICE_COMPILATION
  return false;
#else
  return vecCore::VectorSize<T>() <= vecSize;
#endif
}

#define EARLYRETURNTHR 1
```

Performance effects of early returns

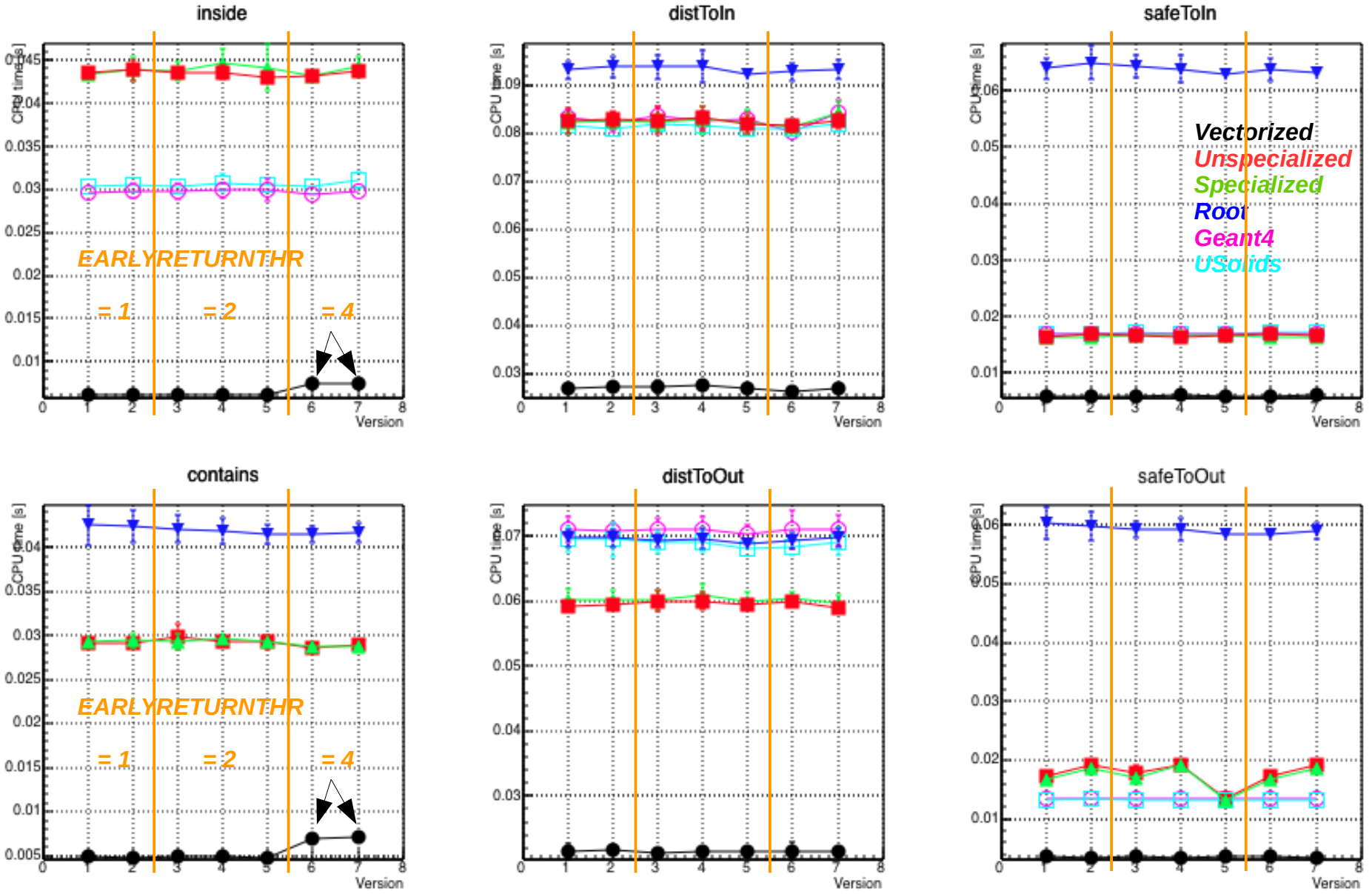
- Look at trapezoid's GenKernelForContainsAndInside():

```
  template <typename Real_v, typename Bool_v, bool ForInside>
  VECGEOM_FORCE_INLINE
  VECGEOM_CUDA_HEADER_BOTH
  static void GenericKernelForContainsAndInside(UnplacedStruct_t const &unplaced, Vector3D<Real_v> const &point,
                                                Bool_v &completelyInside, Bool_v &completelyOutside)
  {
    // z-region
    completelyOutside = Abs(point[2]) > MakePlusTolerant<ForInside>(unplaced.fDz);
    if (vecCore::EarlyReturnAllowedAt<Real_v>(EARLYRETURNTHR) && vecCore::MaskFull(completelyOutside)) {
      return;
    }
    if (ForInside) {
      completelyInside = Abs(point[2]) < MakeMinusTolerant<ForInside>(unplaced.fDz);
    }

#ifndef VECGEOM_PLANESHELL_DISABLE
    unplaced.GetPlanes()->GenericKernelForContainsAndInside<Real_v, ForInside>(point, completelyInside,
                                                                               completelyOutside);
#else
    // here for PLANESHELL=OFF (disabled)
    TrapSidePlane const *fPlanes = unplaced.GetPlanes();
    Real_v dist[4];
    for (unsigned int i = 0; i < 4; ++i) {
      dist[i] = fPlanes[i].fA * point.x() + fPlanes[i].fB * point.y() + fPlanes[i].fC * point.z() + fPlanes[i].fD;
    }

    for (unsigned int i = 0; i < 4; ++i) {
      // is it outside of this side plane?
      completelyOutside = completelyOutside || dist[i] > MakePlusTolerant<ForInside>(0.);
      if (ForInside) {
        completelyInside = completelyInside && dist[i] < MakeMinusTolerant<ForInside>(0.);
      }
      if (vecCore::EarlyReturnAllowedAt<Real_v>(EARLYRETURNTHR) && vecCore::MaskFull(completelyOutside)) return;
    }
#endif

    return;
  }
```

→ *Look at how performance varies with EARLYRETURNTHR value*

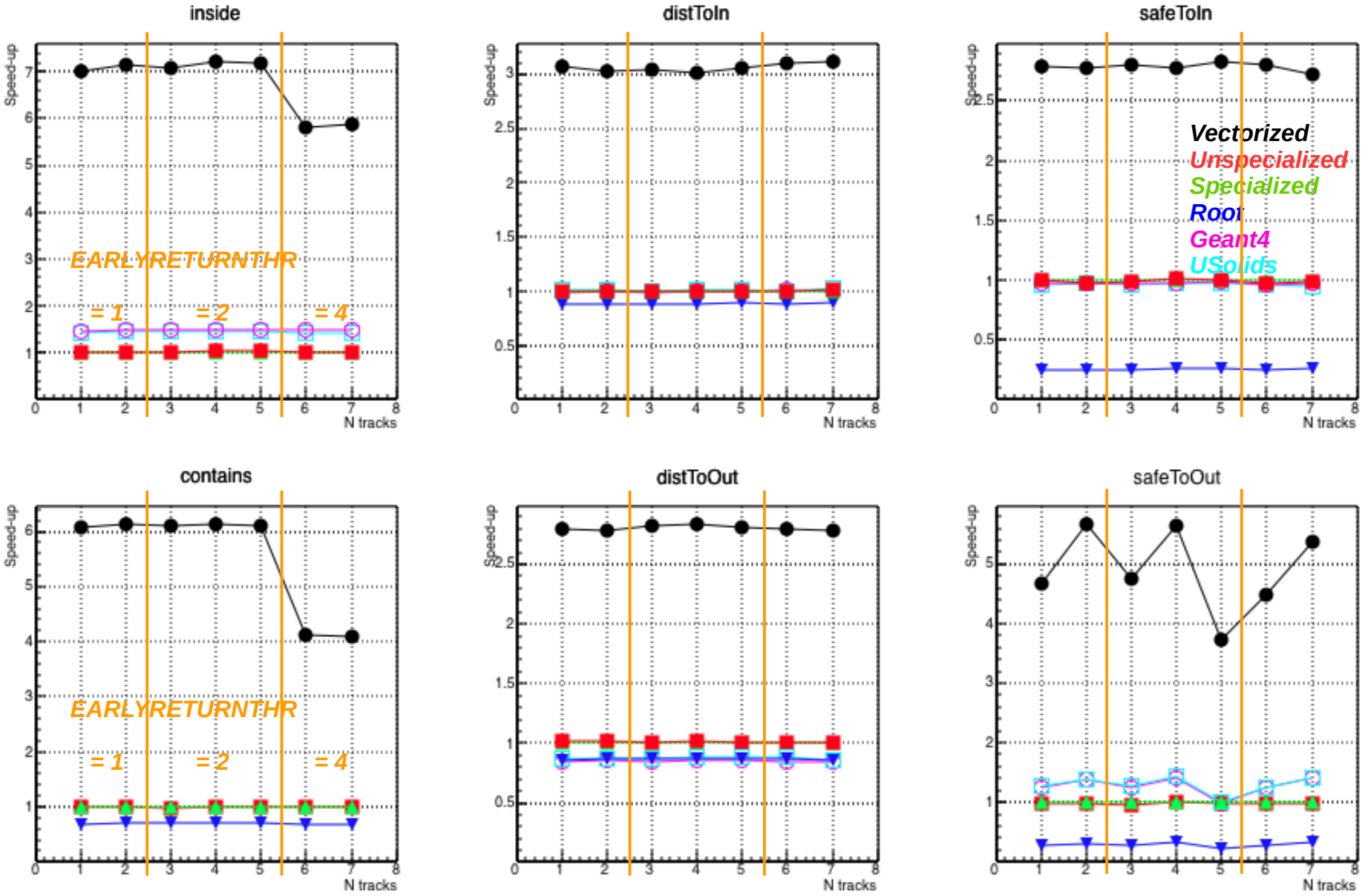
Performance comparisons: CPU times

PLANESHELL=OFF



Performance comparisons: speedups

PLANESHELL=OFF



Suggestions about early returns

- The last two plots show significant overheads from just checking for the conditions for early returns
- Hence the use of early returns should be avoided for very simple functions, like Contains(), Inside() and SafetyToIn,Out() in very simple shapes, as even small overheads produce significant performance degradation.
- Early return checks should be used **very sparingly in vector mode**, unless the checks are true for a significant fractions of the function calls.
- For more complex functions like DistanceToIn,Out(), the overheads can probably be considered small
- The overheads from early return checks in vectorized mode can be minimized (if not completely eliminated) with a smart definition of EarlyReturnAllowed() functions, e.g. always true for scalars, always false for vectors (in CPUs)