What happens in reality – or how to make CoupledTransportation Reproducible

1. **Comparison with zero**
2. **Comparison between doubles**
3. **SafetyHelper**
4. **Caching?**
5. **UrbanMsc**
   - **Fixed geometrical values/dimensions?**
   - **Other safety issues**

Alex Howard, CERN

Safety Implementation - queries

Geant4 Users Workshop Hebden Bridge 19th September 2007

Comparison with zero

- SteppingManger contains the following logic:
  - Max(double – double, 0.);
  - Why?
- SteppingManager2 contains the method safetyProposedToAndByProcess()
  - What does that mean?
- UrbanMscModel checked whether safety was positive – which is a hazard if it's uninitialised or actually zero
- SafetyHelper should be used more extensively?
- kCarTolerance should be used instead of zero?
  - Not used in distance to boundary in MSc – why?
- DistanceToOut returns the real value – always – seems okay provided the boundary definition is applied elsewhere

Comparison between doubles

- G4Navigator has a comparison between two G4ThreeVectors @ line650 – why?
- CLHEP contains the equals operator for G4ThreeVectors by comparing all three components – is this safe?
- CoupledTransportation compares doubles – is it safe to assume they are local and identical?

Double Comparison

```
#ifdef G4VERBOSE
      //  The following checks only make sense if the move is larger
      //  than the tolerance.
      //
      G4ThreeVector OriginalGlobalpoint =
                    fHistory.GetTopTransform().Inverse().
                    TransformPoint(fLastLocatedPointLocal);

      G4double shiftOriginSafSq = (fPreviousSftOrigin-pGlobalpoint).mag2();

      // Check that the starting point of this step is
      // within the isotropic safety sphere of the last point
      // to a accuracy/precision  given by fAccuracyForWarning.
      //   If so give warning.
      //   If it fails by more than fAccuracyForException exit with error.
      //
      if( shiftOriginSafSq >= sqr(fPreviousSafety) )
      {
        G4double shiftOrigin = std::sqrt(shiftOriginSafSq);
        G4double diffShiftSaf = shiftOrigin - fPreviousSafety;

        if( diffShiftSaf > fAccuracyForWarning )
        {
          G4Exception("G4Navigator::ComputeStep()",
                      "UnexpectedPositionShift", JustWarning,
                      "Accuracy ERROR or slightly inaccurate position shift.");
          G4cerr << "     The Step's starting point has moved "
                 << std::sqrt(moveLenSq)/mm << " mm " << G4endl
                 << "     since the last call to a Locate method." << G4endl;
          G4cerr << "     This has resulted in moving "
                 << shiftOrigin/mm << " mm "
                 << " from the last point at which the safety "
                 << "     was calculated " << G4endl;
          G4cerr << "     which is more than the computed safety= "
                 << fPreviousSafety/mm << " mm  at that point." << G4endl;
          G4cerr << "     This difference is "
                 << diffShiftSaf/mm << " mm." << G4endl
                 << "     The tolerated accuracy is "
                 << fAccuracyForException/mm << " mm." << G4endl;

          static G4int warnNow = 0;
          if( ((++warnNow % 100) == 1) )
          {
            G4cerr << "  This problem can be due to either " << G4endl;
            G4cerr << "     - a process that has proposed a displacement"
                   << " larger than the current safety , or" << G4endl;
            G4cerr << "     - inaccuracy in the computation of the safety"
                   << G4endl;
            G4cerr << "  We suggest that you " << G4endl
```

```
  G4int oldcoutPrec= G4cout.precision(8);
  G4int oldcerrPrec= G4cerr.precision(10);
  if( fVerbose > 0 )
  {
    G4cout << "*** G4Navigator::ComputeStep: ***" << G4endl;
    G4cout << "    Volume = " << motherPhysical->GetName()
           << " - Proposed step length = " << pCurrentProposedStepLength
           << G4endl;
    if( fVerbose == 4 )
    {
      G4cout << "    Called with the arguments: " << G4endl
             << "    Globalpoint = " << std::setw(25) << pGlobalpoint
             << G4endl
             << "    Direction   = " << std::setw(25) << pDirection
             << G4endl;
      G4cout << "    ----- Upon entering :" << G4endl;
      PrintState();
    }
  }

  static G4double fAccuracyForWarning   = kCarTolerance,
                  fAccuracyForException = 1000*kCarTolerance;
#endif

  G4ThreeVector newLocalPoint = ComputeLocalPoint(pGlobalpoint);
  if( newLocalPoint != fLastLocatedPointLocal )
  {
    // Check whether the relocation is within safety
    //
    G4ThreeVector oldLocalPoint = fLastLocatedPointLocal;
    G4double moveLenSq = (newLocalPoint-oldLocalPoint).mag2();

    if ( moveLenSq >= kCarTolerance*kCarTolerance )
    {
#ifdef G4VERBOSE
      //  The following checks only make sense if the move is larger
      //  than the tolerance.
      //
      G4ThreeVector OriginalGlobalpoint =
                    fHistory.GetTopTransform().Inverse().
                    TransformPoint(fLastLocatedPointLocal);

      G4double shiftOriginSafSq = (fPreviousSftOrigin-pGlobalpoint).mag2();

      // Check that the starting point of this step is
      // within the isotropic safety sphere of the last point
      // to a accuracy/precision  given by fAccuracyForWarning.
      //   If so give warning.
      //   If it fails by more than fAccuracyForException exit with error.
      //
      if( shiftOriginSafSq >= sqr(fPreviousSafety) )
      {
        G4double shiftOrigin = std::sqrt(shiftOriginSafSq);
        G4double diffShiftSaf = shiftOrigin - fPreviousSafety;
```

SafetyHelper

- Should the onus for correct safety (positive or zero if on a boundary) be moved to the SafetyHelper
- Would ease maintenance and consistency?

Caching

- The Transportations, Navigator, PathFinder and SteppingManagers all cache the safety between steps or proposals from processes
- Is this consistent?
- Are all the (local) caches necessary?
- SafetyHelper?

G4UrbanMscModel (1)

G4UrbanMscModel contains the following examples of fixed (with dimension) Geometrical parameters:

```
+ Help

                                const G4String& nam)
  : G4VEmModel(nam),
    dtrl(m_dtrl),
    lambdalimit(m_lambdalimit),
    facrange(m_facrange),
    facgeom(m_facgeom),
    skin(m_skin),
    steppingAlgorithm(m_stepAlg),
    samplez(m_samplez),
    isInitialized(false)
{
  taubig        = 8.0;
  tausmall      = 1.e-20;
  taulim        = 1.e-6;
  currentTau    = taulim;
  tlimitminfix  = 1.e-6*mm;
  stepmin       = tlimitminfix;
  skindepth     = skin*stepmin;
  smallstep     = 1.e10;
  currentRange  = 0. ;
  frscaling2    = 0.25;
  frscaling1    = 1.-frscaling2;
  tlimit        = 1.e10*mm;
  tlimitmin     = 10.*tlimitminfix;
  nstepmax      = 25.;
  geombig       = 1.e50*mm;
  geommin       = 1.e-3*mm;
  geomlimit     = geombig;
  presafety     = 0.*mm;
  facsafety     = 0.25;
  Zeff          = 1.;
  particle      = 0;
  theManager    = G4LossTableManager::Instance();
  inside        = false;
  insideskin    = false;

--:--  G4UrbanMscModel.cc      (C++ CVS-1.66 Abbrev)--L197--23%----------
```

G4UrbanMscModel (2)

What is small step?
What is tlimit?

```
sp->GetStepStatus();

// standard  version
//
if (steppingAlgorithm == fUseDistanceToBoundary)
  {
    //compute geomlimit and presafety
    GeomLimit(track);

    // is far from boundary
    if(currentRange <= presafety)
      {
        inside = true;
        return tPathLength;
      }

    smallstep += 1.;
    insideskin = false;

    if((stepStatus == fGeomBoundary) || (stepNumber == 1))
      {
        if(stepNumber == 1) smallstep = 1.e10;
        else  smallstep = 1.;

        // facrange scaling in lambda
        // not so strong step restriction above lambdalimit
        G4double facr = facrange;
        if(lambda0 > lambdalimit)
          facr *= frscaling1+frscaling2*lambda0/lambdalimit;

        // constraint from the physics
        if (currentRange > lambda0) tlimit = facr*currentRange;
        else                        tlimit = facr*lambda0;
```

```
      //lower limit for tlimit
      if(tlimit < tlimitmin) tlimit = tlimitmin;

      // constraint from the geometry (if tlimit above is too big)
      G4double tgeom = geombig;

      if((geomlimit < geombig) && (geomlimit > geommin))
        {
          if(stepStatus == fGeomBoundary)
            tgeom = geomlimit/facgeom;
          else
            tgeom = 2.*geomlimit/facgeom;

          if(tlimit > tgeom) tlimit = tgeom;
        }
    }

  //if track starts far from boundaries increase tlimit!
  if(tlimit < facsafety*presafety) tlimit = facsafety*presafety ;

  //  G4cout << "tgeom= " << tgeom << " geomlimit= " << geomlimit
  //     << " tlimit= " << tlimit << " presafety= " << presafety << G4endl;

  // shortcut
  if((tPathLength < tlimit) && (tPathLength < presafety))
    return tPathLength;

  G4double tnow = tlimit;
  // optimization ...
  if(geomlimit < geombig) tnow = max(tlimit,facsafety*geomlimit);

  // step reduction near to boundary
  if(smallstep < skin)
    {
      tnow = stepmin;
```

```
          if(geomlimit > skindepth)
            {
              if(tnow > geomlimit-0.999*skindepth)
                tnow = geomlimit-0.999*skindepth;
            }
          else
            {
              insideskin = true;
              if(tnow > stepmin) tnow = stepmin;
            }
        }

      if(tnow < stepmin) tnow = stepmin;

      if(tPathLength > tnow) tPathLength = tnow ;
    }
    // for 'normal' simulation with or without magnetic field
    //  there no small step/single scattering at boundaries
  else if(steppingAlgorithm == fUseSafety)
    {
      // compute presafety again if presafety <= 0 and no boundary
      // i.e. when it is needed for optimization purposes
      if((stepStatus != fGeomBoundary) && (presafety < tlimitminfix))
        presafety = safetyHelper->ComputeSafety(sp->GetPosition());

      // is far from boundary
      if(currentRange < presafety)
        {
          inside = true;
          return tPathLength;
        }

      if((stepStatus == fGeomBoundary) || (stepNumber == 1))
        {
          // facrange scaling in lambda
```

Conclusions/Discussion

- Why are we comparing doubles?
- Sometimes safety is un-initialised (old transportation/navigation?), which means even comparing with zero is a hazard…
- The SteppingManager max(double-double, 0) causes real reproducibility issues
- G4UrbanMscModel baffles me completely:
  - Fixed geometrical values (with no reference to scalability/geomtry size)
  - Safety compared against internal values – why?
  - AtBoundary definition doesn't appear to be concrete
- Caching of safety vs. SafetyHelper vs. local copies should be consistent… Otherwise it can get carried over from the previous event!
- Verbosity displays a number of safety values which are small but non-zero $\mathcal{O}(10^{-12})$ – why?