Minimising Precision Problems in GEANT 4 Geometry

P. Kent

February 1995
Revised 10 April 1995

1 Introduction

This document describes precision problems in GEANT 3's geometric primitives, how they are (usually) overcome, and problems resulting from the current implementation.

Based on these problems, some aims - requirements - for GEANT 4 are presented, together with a possible scheme for satisfying them.

I emphasise that this document should serve as a basis for discussion, and is not a full design document.

2 GEANT 3 Primitives

GEANT currently contains 16 primitives, each providing essentially five services at tracking time to the tracking routines (ex. GTNEXT).

Inside(Point) Returns true if the point is inside the primitive, or false if outside.

DistanceToEntry(Point) Calculates the shortest distance ('safety') from a point outside the primitive, to any boundary of the primitive. This can be, and usually is, an underestimate of the true minimum distance.

DistanceToExit(Point) Calculates the shortest distance ('safety') from a point inside the primitive, to any boundary of the primitive. This can be, and usually is, an underestimate of the true minimum distance.

DistanceToEntry(Point,Direction) Calculates the distance from a point outside the primitive, to the primitive's boundary, along a specified direction vector.

DistanceToExit(Point,Direction) Calculates the distance from a point inside the primitive, to the primitive's boundary, along a specified direction vector.

Each primitive has additions services such as calculating normal vectors and scope, but these are not essential to 'pure' tracking.

2.1 Pure Tracking

With no physics interactions, pure tracking consists of the 3 following steps:

1. Identify the solid[1] that the particle is inside.
2. Calculate the intersection distance to volumes inside of the current solid, and to the current solid's boundary (exiting distance).
3. Calculate the minimum step, and move the particle.

This scheme is enhanced by the use of the safety calculations described above, and optimisation schemes such as virtual division, but the underlying process remains the same.

2.2 Problems

Several problems result from the above stepping algorithm:

The intersection functions do not always guarantee that the distance returned is the exact distance - they can be underestimates - and as a consequence it is necessary to recalculate within which solid a particle is in after stepping, whereas this could be determined logically.

To avoid precision problems at boundaries, a small delta is applied to the step length, to ensure that a particle will enter a new solid. This is done for all geometrically limited steps in the simulation. Figure 1 illustrates this.

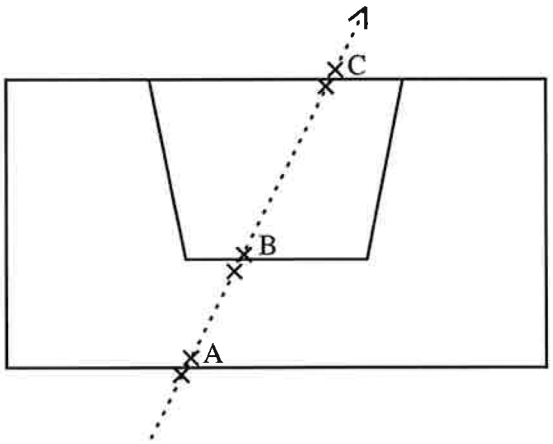

Figure 1: Tracking with pushes. Each push must be large enough to step into the next primitive.

The push mechanism has two main problems:

1. If boundaries of adjacent solids are not coincident, due to rotation and positioning errors, particles can be forced to make one or more small 'epsil' steps to cross both boundaries, if the 'push' is not large enough. Epsil steps result in a large performance loss.

[1] Solid - A geometrical primitive positioned in 3-space

2. The 'push' can affect physics. A push can result in a particle entering a solid, instead of reflecting at the boundary (Figure 2). Normally, this is infrequent enough to be only a minor error.

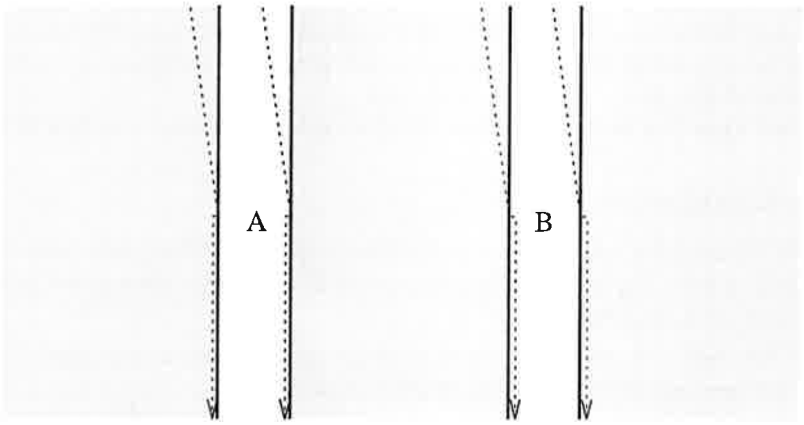

Figure 2: Calorimeter Tracking. The particles should reflect (case A) , but due to the push can continue (case B) into the next solid.

An additional precision problem can occur when a particle appears to be inside a solid, but due to different algorithms/projections used in the distance computation routine, it is possible for a particle to be 'outside' - as seen by the distance computation routine.

This can have effects on the distance computation ranging from negligible, to catastrophic, returning a very large value instead of only an epsil magnitude step. The hexahedral primitives of GEANT 3 are most robust in this respect, and the second order primitives least robust.

3 Aims for GEANT 4

Aims for any revised geometry should include:

1. To be accurate and efficient.
2. To have a well defined specification for the 5 main run-time services - for example, how a DistanceToEntry function should react to a point that is already inside.
3. To minimise the need to re-evaluate a particle's location during tracking. Logical methods should be used to determine a particle's location as much as possible.

Additionally, any requirements made for primitive distance computation must be satisfiable by, for example, a NURB-based primitive from a CAD system, or by surface based GISMO-style primitives.

Improving the robustness of the geometry will help, eliminate the possibility of 'stuck' particles and the simulation looping endlessly.

4 Tolerant Geometry

This section details a proposal avoid precision problems, without the inherent dangers of the 'push'.

Boundaries are effectively given a 'thickness', moving the responsibility for managing precision problems from the tracking to the primitives themselves. Hence the primitives can make use of knowledge of the symmetry (etc.) without compromising safety.

Some examples or how tracking could proceed are given in section 4.2.

4.1 Changed Services

Inside(Point) All boundaries are considered to have a thickness, extending to either side (figure 3). For efficiency, phi and theta boundaries could be described by a delta angle.

The thickness must be greater that the precision error resulting from the intersection/distance computation functions.

Instead of returning a boolean result, the function returns whether a point is definitely inside, outside, or on the surface of the primitive.

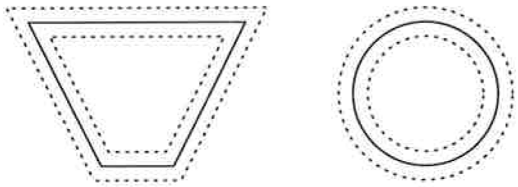

Figure 3: Tolerant boundaries.

DistanceToEntry(Point) Remains as defined in section 2, and is also guaranteed to return 0 for a point that is already inside.

DistanceToExit(Point) Remains as defined in section 2, and is also guaranteed to return 0 for a point that is already outside.

DistanceToEntry(Point,Direction) Calculates the distance from a point outside the primitive, to the primitive's boundary, along a specified direction vector.

The distance to the boundary is guaranteed to be the exact distance. When there are no intersections infinity is returned. Intersections with boundaries less than the tolerance from the point must be ignored if the direction is away from the boundary. This allows for the point to be 'inside' an exact boundary, and the next entry distance calculated (figure 4). By similar arguments, the first root with second order surfaces may need to be discarded.

DistanceToExit(Point,Direction) Calculates the distance from a point inside the primitive, to the primitive's boundary, along a specified direction vector.

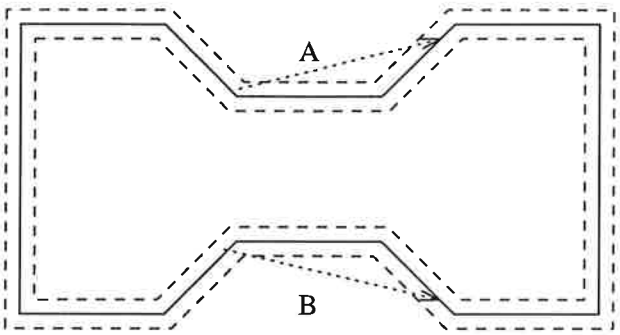

Figure 4: Calculating entry distance. The functions must calculate the distance correctly when the point is outside the exact boundaries (case A) or inside the exact boundaries but within the tolerance (case B).

The distance is guaranteed to be the exact exiting distance.

If the shortest distance to a boundary is less than the tolerance, it is ignored. This allows for the point to be within a tolerant boundary, and to leave immediately.

As an aid to the tracking, the normal of the surface at the point of exit can be returned, when that surface can be considered as 'convex' - when it would not be possible to reenter the primitive without a change of momentum to a direction away from the outwards normal.

This enables the primitive to be ignored at the next tracking step, or, in the case of reflection at a boundary, to immediately know that the particle is going back through the primitive. An example is given in section 4.2.

Both of the distance computations that take a direction could be specified to return 0 when completely inside or outside (ie. not within a tolerant boundary). Although this would further increase the robustness of the geometry, this would increase the complexity of these functions further and, depending on the tracking may not actually be necessary.

4.2 Examples

Some examples of how particle tracking could be carried out are given below. Although a tree like, mother-daughter topology is assumed, a flat structure could equally be used.

4.2.1 Simple Tracking

Figure 5 shows a track entering a solid, stepping through several daughter solids, and finally exiting the original mother.

On stepping to point 1, it is know that the particle is at least inside A. By recursively checking daughters, it is found that the particle has actually entered B via a coincident boundary.

B has no daughters, so DistanceToExit for B is found, and the particle steps to 2. On leaving the solid, other daughter of the new mother are checked in case

the particle immediately enters one. In this case, the particle is on the surface of B and C, but since it is known that the particle is leaving B, the particle must be inside C on the next step.

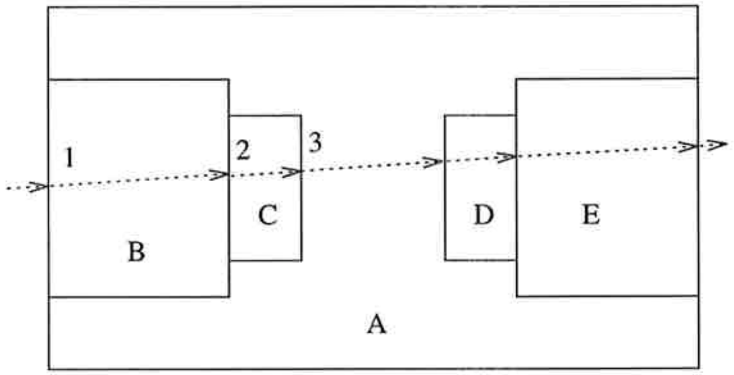

Figure 5: Simple Tracking.

4.2.2 Reflection on a Boundary

If DistanceToExit returns a surface normal, reflections at boundaries can be detected trivially, eliminating the need to search other solids. This is a case that is handled poorly by the current GEANT.

If the particle's momentum is away from the direction of the outwards pointing surface normal, then the solid is not left (figure 6, case A).

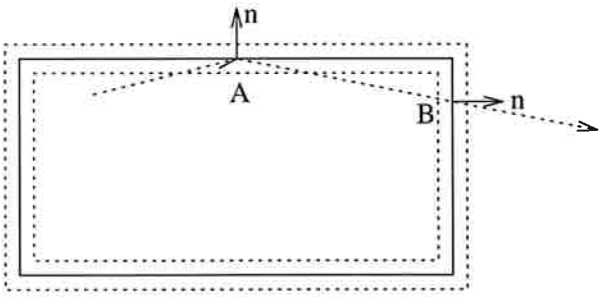

Figure 6: Boundary Reflections

Conversely, when particles leave a solid, the solid can be excluded from further checks if the momentum has a component in the direction of the outwards normal (figure 6, case B).

This checks can apply to some surfaces of convex primitives - for example, when leaving by the outer radius of a GEANT 3 CONS primitive, the surface normal can be used. Phi planes can be considered convex, when $\delta\phi \leq 180$, together with the outer radii of TUBE(S), CONE(S) and SPHEres, in addition to all surfaces of all the hexahedral primitives.

The performance benefit of utilising exiting surface normals can be easily compared with the performance of a simpler scheme by simply not calculating the normals.

4.3 Evaluation and Further Work

The 'tolerant geometry' outlined above, has the potential to satisfy all of the aims listed above (section 3), provided that any ambiguous cases can be resolved by the tracking, by making use of the 'safety' distances.

The tolerant geometry should minimise the number of 'epsil' steps taken in cases where many surfaces are not completely coincident, due to positioning errors.

Additionally, four points must be examined:

Feasibility It must be determined whether the functions described can be written safely for more general primitives, such as NURBS, and any ambiguous cases checked if they cannot be dealt with satisfactorily.

Efficiency The exiting normal technique must be checked to see whether it is a win, on average. In most cases, calculation of a surface normal is almost free - the trigonometry and calculations required for surfaces normals are similar to those required for the intersection calculations. However, if the particle is moved to a different destination due to multiple-scattering, a magnetic field, or the step is not limited by geometry, the returned normal is meaningless.

Validation A means of ensuring that the tolerance is large enough must be found.

Cost and Complexity The scheme would add to the complexity of the distance computation functions, making them harder to debug and maintain. This is not a valid reason on its own however, because of the savings in reducing the number of steps taken by each particle, or in re-evaluating the location of the particles during tracking.

5 Conclusions

A scheme to minimise precision problems in geometrical primitives was presented, and compared against GEANT 3's implementation. Although the scheme appears promising, further design work must be undertaken. In the context of the GEANT 4 project, this work remains ahead of its time, so further work must wait until the geometric sub-system is better defined, and its interface with the general tracking sub-system established.