Subject: Re: [Possible bug] Geometry - Point
Posted by rylek on Thu, 14 Dec 2006 22:01:34 GMT
View Forum Message <> Reply to Message

Hello artekk!

I am afraid that your proposal, although sound in motivation, cannot stand a closer scrutiny. One of its major flaws has been pointed above by Mirek - the EPSILON is highly application- and context-dependent and cannot be pre-set to a constant value to suit every possible situation. Another argument against it looks somewhat abstract, but has serious practical implications: the above definition of equality is not transitive, i.e. generally (a == b && b == c) doesn't imply a == c. This is very important because it can introduce unexpected bugs into various algorithms. The inexact nature of the above proposed 'similarity relation' also causes trouble in hashing (e.g. in the Index / Map class family) which is based on the axiom that for each two values a, b, such that a == b, it must also hold that GetHashValue(a) == GetHashValue(b). When you define equality via similarity, I believe there is no way to define the hash value algorithm (other than a fixed constant independent on the argument) not breaking that axiom.

To make the long story short: I believe there is good reason in standard C/C++ for the == operator on floats and doubles to match only exactly equal values (and I think it is only natural for Pointf to build on this behaviour as in analytical geometry points and vectors are usually seen simply as ordered n-tuples of real numbers).

It is true, of course, that, when working with real numbers (and of course also with their n-tuples), funny things sometimes happen due to rounding and inexact nature of various arithmetical and transcendental operations. However, the nature of these deviations from math-theoretical behaviour (i.e. the difference between results of finite-precision and infinite-precision calculations) is far too complex to be "swept under the rug" using such a simple modification of the equality operator. I myself write real-number algorithms quite often and I strongly believe that the construction of a robust algorithm involves thorough theoretical analysis of the introduction and propagation of noise in the algorithm, static and/or dynamic estimation of error accumulation and a clear distinction where deviations are tolerable and the acceptable magnitude of these deviations (for instance, you usually need altogether different measure for linear and angular deviations).

Regards

Tomas