Geometric Tools Engine Update History

Last modified: January 6, 2022

Contents

1 Version 6.0 **2**

The version release dates are listed here. Versions released before the current version may be obtained by email request.

- Version 6.0 posted January 3, 2022.

The updated files and related notes are provided for the versions in each of the ensuing sections. Each section has a list of changes that occurred to the version number mentioned in that section. Those changes were rolled up into the zip file that was posted for the next version. Modified files are colored gold, new files are colored green and deleted files are colored red. Source code is colored Violet.

1 Version 6.0

January 3, 2022. The major revision is based on running the code analysis tools for Microsoft Visual Studio 2019 16.11.8 and for ClangCL. The reported issues were addressed with the exception of some incorrect warnings from MSVS 2019. For now, the incorrect warnings are encapsulated by #pragma commands and will be removed in the GTL development track.

The MSVS code analysis tool is not robust. Sometimes warnings occur in both the Output and Error List windows. Sometimes they occur in only one or the other window (but not both). And sometimes they do not occur in either window, but the 3-dot markers show up in source files that are opened after which the warnings show up in the Error List window. As I discover source files where the warnings show up only when the source file is opened, I will fix the issues. I believe most of these will be warnings about potentially uninitialized variables, typically for class objects whose default constructors should be called but MSVS seems to believe the class members are not initialized. This is the case for the Vector classes, but in the code using these classes, the members are set soon after the declaration. I was hoping the tool would figure out that the member initialization was deferred. (If I were to allocate a std::vector of Vector with a very large number of elements, and the code fills them in by loading data from a file, it would be a shame to waste all that time having the constructor initialize the members only to fill them in again from the loaded data.)

Another tool issue occurred with SymmetricEigensolver.h, in the GetEigenvector function. I have comments in that file about why the tool report is incorrect. For a long time I have been able to ignore the warning because code using it compiles fine, even with *treat warnings as errors*. The warnings occurred as part of a regular source-code build. For the first time I ran the code analyzer explicitly from the MSVS IDE menu on a source file that contains only the include of the aforementioned header file. The report now includes more warnings that appear to be based on the same incorrect diagnosis. The original warning was about a potential out-of-range index, and the Error List window allows you to drop-down a list of steps and assumptions to support the diagnosis. The new warning is about the same issue but for some reason displays a list of line numbers that show up in the drop-down list. After these new warnings occurred, I can no longer successfully build code using the eigensolver. I had to add #pragma commands to prevent the warnings. After that, a couple of other files generated similar warnings that had to be disabled using #pragma commands.

I had also spent a lot of time eliminating the warning about preferring scoped enumerations over unscoped ones. The unscoped ones typically defined enumerants that were used as array indices (in the DX11 and GL45 engine code). This is not allowed with scoped enumerations. I replaced the unscoped enumerations with nested struct, each structure containing constant expressions of the form static uint32_t constexpr someName = someValue;. MSVS 2019 and ClangCL provided with MSVS 2019 allowed this modification, but unfortunately

gcc on my Linux boxes did not. Searching stackoverflow, it appears that C++ considers such structs to be *incomplete*. I actually got linker errors about the various constants referenced by the sample applications but not found in the libraries linked to the applications. I restored the unscoped enumerations and disabled the code analysis warning number in the project settings.

A couple of the sample applications use BSRational<UIntegerFP<N>> where N is large. These lead to code analysis warnings about /analyze:stacksize numKBs indicating that numKBs is larger than the maximum stack size and the you should consider moving data from the stack to the heap. The samples have run correctly without stack overflow errors, so it is not clear to me what the problem is. Regardless, I modified the project settings and specified numKBs large enough to avoid the warnings. When I switched to using ClangCL, the compiler complains it cannot find files and lists the name /analyze:stacksize. It appears that ClangCL does not understand this analysis tool option. Unfortunately, with *treat warnings as errors*, those sample applications will not compile. So I removed the setting of numKBs and then disabled the code analysis warning number in the project settings.