

Geometric Tools Engine Update History

Last modified: February 20, 2023

Contents

1	Updates to Version 6.5	2
2	Updates to Version 6.4	5
3	Updates to Version 6.3	9
4	Updates to Version 6.2	11
5	Updates to Version 6.1	15
6	Updates to Version 6.0	20
7	Version 6.0	25

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

- Version 6.5 posted December 14, 2022.
- Version 6.4 posted June 8, 2022.
- Version 6.3 posted April 3, 2022.
- Version 6.2 posted March 7, 2022.
- Version 6.1 posted February 7, 2022.
- 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 Updates to Version 6.5

February 20, 2023. Eliminated the Fedora gcc12.{1,2}.1 release-build warnings about uninitialized members in **std.function.h**, probably due to incorrect optimization by the compiler. The warnings occurred in release builds of **PVWUpdater.cpp** where the member default constructors were called in the class constructors followed by a **Set** call to assign the members. I removed the **Set** calls and initialize the members directly in the constructor initializer list.

GTE/Graphics/PVWUpdater.cpp

February 2, 2023. Added the parent window handle to **Window::Parameters** for GTE on Microsoft Windows. That handle is passed to the **CreateWindow** functions.

GTE/Applications/MSW/Window.{h,cpp}
GTE/Applications/MSW/WindowSystem.cpp

Added the **OnCopyData** callback for GTE on Microsoft Windows. This allows passing data from one application to another application.

GTE/Applications/WindowApplication.{h,cpp}

Fixed linter complaint suggesting that **.empty()** be used instead of **.size() == 0**.

GTE/Graphics/CLODMeshCreator.h
GTE/Applications/MSW/WindowSystem.cpp

Added a missing pragma.

GTE/Mathematics/ContTetrahedron3.h

January 24, 2023. Added a new file for fitting a rectangle to a convex quadrilateral that is nearly a rectangle. The algorithm uses a least-squares formulation.

GTE/Mathematics/ApprConvexQuadByRect.h
GTE/GTMathematics.{v16,v17}.{vcxproj,vcxproj.filters}

January 17, 2023. Modified `ETManifoldMesh.h` to conform to commenting standards and choosing `using` instead of `typedef`. Added explicit initialization in the constructors of nested classes. Added braces to default constructor calls. The most important change is that during unit testing of the port of `ETManifoldMesh` from GTE to GTL, I created a small mesh for which `GetBoundaryPolygons` triggered an assertion (unexpected for a correct implementation of the algorithm). The idea of processing triangle strips at boundary vertices that are shared by 4 or more edge is correct. The code works for the strict definition of manifold mesh, where bow-tie configurations are not allowed. However, GTE has relaxed the constraint and allows bow-ties. For such meshes, the implementation of `GetBoundaryPolygons` is flawed and can lead to failure. Moreover, the approach I used in the implementation is too complicated. I re-implemented the algorithm to be simpler.

GTE/Mathematics/ETManifoldMesh.h

Modified `VETManifoldMesh.h` to conform to commenting standards and explicitly initializing nested class members in their constructors.

GTE/Mathematics/VETManifoldMesh.h

Modified `TSManifoldMesh.h` to conform to commenting standards and choosing `using` instead of `typedef`. Added explicit initialization in the constructors of nested classes. I renamed the `T` member to `S` in the nested class `Triangle`. The other mesh classes use `T` to stand for *Triangle*. Using `T` for *Tetrahedron* makes the code difficult to read at times. Switching to `S` (for *Simplex*) helps with readability. Finally, I added the keyword `virtual` to `Insert` and `Remove` to support a new class `VTSManifoldMesh` that adds vertex manipulation to the base class `TSManifoldMesh`.

GTE/Mathematics/TSManifoldMesh.h
GTE/Mathematics/Delaunay3.h
GTE/GTMathematics.{v16,v17}.{vcxproj,vcxproj.filters}
GTE/Mathematics/VTSManifoldMesh.h

The static 2-manifold class has a member function `GetBoundaryPolygons`, just as the dynamic class `ETManifoldMesh` does. This function has the same flawed implementation mentioned previously. It is now re-implemented using the same approach as in `ETManifoldMesh`.

GTE/Mathematics/StaticVETManifoldMesh2.h

Added a static 3-manifold class for tetrahedra meshes. This is significantly faster than the dynamic classes `TSManifoldMesh` and `VTSManifoldMesh`.

GTE/Mathematics/StaticVTSManifoldMesh3.h

Added newer static manifold mesh classes, [MeshStaticManifold2](#) and [MeshStaticManifold3](#), that perform better than [StaticVETManifoldMesh2](#) and [StaticVTSManifoldMesh3](#). The newer classes are also multithreaded. The older static classes will be deprecated for GTL in favor of the newer static classes.

GTE/Mathematics/MeshStaticManifold2.h

GTE/Mathematics/MeshStaticManifold3.h

Important Note. [ETManifoldMesh](#), [VETManifoldMesh](#) and [StaticVETManifoldMesh2](#) store adjacency pointers and indices according to the following design: Let the triangle have ordered vertices $(V[0], V[1], V[2])$. The triangle adjacent to edge $(V[0], V[1])$ is $A[0]$. The triangle adjacent to edge $(V[1], V[2])$ is $A[1]$ and the triangle adjacent to edge $(V[2], V[0])$ is $A[2]$. The new class [MeshStaticManifold2](#) uses a different design: The triangle adjacent to edge $(V[1], V[2])$ is $A[0]$, the triangle adjacent to edge $(V[2], V[0])$ is $A[1]$ and the triangle adjacent to edge $(V[0], V[1])$ is $A[2]$. That is, $A[i]$ is the triangle adjacent to the edge opposite vertex $V[i]$.

The design for the new 2-manifold triangle meshes is consistent with the design of the 3-manifold tetrahedra meshes [TSManifoldMesh](#) and [VTSManifoldMesh](#). Let the tetrahedron have ordered vertices $(V[0], V[1], V[2], V[3])$. The triangle faces are counterclockwise ordered when viewed from outside the tetrahedron, $T[0] = (V[1], V[2], V[3])$, $T[1] = (V[0], V[3], V[2])$, $T[2] = (V[0], V[1], V[3])$ and $T[3] = (V[0], V[2], V[1])$. Each face is opposite the vertex that is not listed in the face. The tetrahedron adjacent to face $T[i]$ is $A[i]$. Unfortunately, I had not noticed previously that the primitive class [Tetrahedron3](#) assigns the faces in the opposite order: $T[0] = (V[0], V[2], V[1])$, $T[1] = (V[0], V[1], V[3])$, $T[2] = (V[0], V[3], V[2])$ and $T[3] = (V[1], V[2], V[3])$.

The edge and face design in GTL will be consistent and uses the designs of [MeshStaticManifold2](#) and [MeshStaticManifold3](#).

The profiling results are listed next for the 2-manifold classes, running on an Intel i9-10900 CPU. The test triangle mesh has 1048576 vertices and 2097117 triangles.

class	numThreads	milliseconds
VETManifoldMesh	1	8471
StaticVETManifoldMesh2	1	1129
	2	904
	4	817
	8	761
	16	759
MeshStaticManifold2	1	816
	2	629
	4	526
	8	542
	16	551

The profiling results are listed next for the 3-manifold classes, running on an Intel i9-10900 CPU. The test tetrahedron mesh has 262144 vertices and 1765604 tetrahedra.

class	numThreads	milliseconds
VTSManifoldMesh	1	10706
StaticVTSManifoldMesh3	1	2035
	2	1512
	4	1256
	8	1160
	16	1092
MeshStaticManifold3	1	2202
	2	691
	4	507
	8	391
	16	362

January 11, 2023. The [Error](#) function and the comment in the file about the error formula were incorrect. They appear to have been copied-and-pasted from the [ApprOrthogonalLine3.h](#) implementation. The error code was introduced in the port from WM5 to GTE1. Thanks to Scott Johnson for informing me about the mistakes.

[GTE/Mathematics/ApprOrthogonalPlane3.h](#)

January 4, 2023. The posted file [StaticVETManifoldMesh2.h](#) was out of sync with the development version and was missing the initialization of the members of [mAdjacents](#) to *invalid* (-1). I also modified some comments and added some [const](#) modifiers where necessary.

[GTE/Mathematics/StaticVETManifoldMesh2.h](#)

January 1, 2023. New file that provides an additional speed-up to [VETManifoldMeshNR](#) for constructing static vertex-edge-triangle manifold meshes from collections of triangles.

[GTE/GTMathematics.{v16,v17}.vcxproj*](#)
[GTE/Mathematics/StaticVETManifoldMesh2.h](#)

December 24, 2022. Feature request for polyline/polygon offset. The code and a sample application has been added.

[GTE/GTMathematics.{v16,v17}.vcxproj*](#)
[GTE/Mathematics/PolylineOffset.h](#)
[GTE/BuildAll*.sln](#)
[GTE/Samples/Geometrics/CMakeLists.txt](#)
[GTE/Samples/Geometrics/PolylineOffset/*](#)

2 Updates to Version 6.4

December 14, 2022. The following are modifications based on complaints from ClangCL for v143 MSVS tools regarding unused variables. The unused variables were actually used in the code (for watching during debugging), but the final values were never consumed by the functions.

GTE/Graphics/Shader.cpp ([handle](#))
GTE/Graphics/GL45/GLSLShader.cpp ([blockIndex](#))
GTE/Samples/Intersection/IntersectConvexPolyhedra/ConvexPolyhedron.h ([numZero](#))
GTE/Samples/Graphics/MultipleRenderTargets/MultipleRenderTargetsWindow3.cpp ([hr](#))
GTE/Samples/Physics/RoughPlaneParticle1/RoughPlaneParticle1Window2.h ([mSize](#))
GTE/Samples/Physics/WrigglingSnake/WrigglingSnakeWindow3.cpp ([t](#))

Added [override](#) to [DrawScreenOverlay](#) to resolve ClangCL complaint.

GTE/Samples/Geometrics/MinimumAreaBox2D/MinimumAreaBox2DWindow2.h

Added missing file to projects.

GTE/Samples/Intersection/IntersectConvexPolyhedra/IntersectConvexPolyhedra{DX11, GL45}.vcxproj.filters

December 12, 2022. Added new code and sample application for 2D find-intersection queries between a linear component (line, ray, segment) and a segment mesh.

GTE/GTMathematics.*.*
GTE/Mathematics/IntrLine2SegmentMesh2.h
GTE/Mathematics/IntrRay2SegmentMesh2.h
GTE/Mathematics/IntrSegment2SegmentMesh2.h
GTE/BuildAll*.sln
GTE/Samples/Intersection/CMakeLists.txt
GTE/Samples/Intersection/IntersectRayPolygon/*

Fixed a comment.

GTE/Mathematics/IntrLine2Segment2.h

December 1, 2022. Added new 2D distance queries for point-circle and point-arc. Also fixed some missing header file references in the GTMathematics.v16 project.

GTE/Mathematics/DistPoint2Arc2.h
GTE/Mathematics/DistPoint2Circle2.h
GTE/GTMathematics.{v16,v17}.vcxproj
GTE/GTMathematics.{v16,v17}.vcxproj.filters

November 27, 2022. The [CopyBackBuffer](#) call mapped [mBackBufferStaging](#) but failed to unmap it after the copy.

GTE/Graphics/DX11/DX11Engine.cpp

November 11, 2022. Rolling back the changes from the 4 November 2002 update. The code is faster but the rounding errors cause misclassification of the number of points of intersection. Welcome to the trade-off

of floating-point versus rational arithmetic. At some point I need to implement rational arithmetic classes using the GNU Multiprecision Library to take advantage of the better performance for arbitrary precision multiplication. At a later date, I can implement a similar speed-up for [BSNumber](#) and [BSRational](#).

[GTE/Mathematics/IntrEllipse2Ellipse2.h](#)

November 4, 2022. When using [BSRational<UIntegerAP32>](#) for template type [T](#), the number of words used in the [SolveQuartic](#) root finder can be extremely large—on the order of 100000. This leads to poor performance. The root finder has numerical rounding errors whether [T](#) is a floating-point type or a rational type, so to avoid the poor performance, the rational-valued quartic coefficients are converted to [double](#) and then reconverted to [T](#). This introduces a small amount of rounding error, but performance is significantly better.

[GTE/Mathematics/IntrEllipse2Ellipse2.h](#)

November 2, 2022. Added a fuzzy containment that tests whether a point is on a circular arc. The non-fuzzy containment requires the point to be on the circle of the arc (or nearly on the circle within floating-point rounding errors).

[GTE/Mathematics/Arc2.h](#)

October 17, 2022. Back-ported [QuadricSurface](#) from GTL to GTE, both versions unit tested.

[GTE/Mathematics/QuadricSurface.h](#)

September 23, 2022. Fixed a bug in [ComputeSqrDistance](#) on line 247 (incorrect sign on s). Thanks to Tom Foster for reporting the bug.

[GTE/Mathematics/IntrCanonicalBox3Cylinder3.h](#)

August 25, 2022. Reimplemented the TCB spline code based on the rewrite of the PDF document *Kochanek–Bartels Cubic Splines (TCB Splines)*. I removed the tangent adjustments suggested by Kochanek and Bartels (based on discrete variables), replacing them with adjustments based on continuous variables. I added support for specifying the boundary tangents explicitly rather than duplicating the endpoints of the keyframes and computing the boundary tangents internally. Finally, I added a parameter (λ) per key frame position to control the speed at those positions and ensure continuity of speed.

[GTE/Mathematics/TCBSplineCurve.h](#)

August 23, 2022. Feature request for code to compute the minimum-area aligned box that contains a circular arc.

[GTE/Mathematics/ContAlignedBox2Arc2.h](#)

August 16, 2022. Reset some [std::vector](#) arrays for re-use of the [operator\(\)](#) function. Added [mConvexHull3](#) member and accessor (from Leopard20 at github). Modified the file version.

GTE/Mathematics/MinimumVolumeBox3.h

July 13, 2022. The managed library project compiles with the directory structure of GTE, but if folders are moved elsewhere, the managed library needed `$(GTE_PATH)` for the header search path. `GTE_PATH` is an environment variable that stores the path to the top-level folder `GeometricTools/GTE` (see the installation and release notes).

GTE/Samples/CSharpCppManaged/ManagedLibrary/ManagedLibrary.*,vcxproj

July 12, 2022. Removed the file `ManagedObject.h`, consolidating the contents with the managed files `MinimumVolumeBox.{h,cpp}`. Updated the copyright date in the assembly file. Modified the comments in the program file.

GTE/Samples/CSharpCppManaged/CSharpCppManaged*.sln
GTE/Samples/CSharpCppManaged/ManagedLibrary/ManagedLibrary.*
GTE/Samples/CSharpCppManaged/ManagedLibrary/MinimumVolumeBox.{h,cpp}
GTE/Samples/CSharpCppManaged/ManagedLibrary/ManagedObject.h
GTE/Samples/CSharpCppManaged/CSharpApplication/Properties/AssemblyInfo.cs
GTE/Samples/CSharpCppManaged/CSharpApplication/Program.cs

July 4, 2022. The `Window2` class was implemented not to allow window resizing. I removed this constraint and added comments in the header file about the responsibilities of an application class derived from `Window2`. The 2D fluid simulation sample application was modified to illustrate the resizing.

GTE/Applications/Window2.{h,cpp}
GTE/Samples/Physics/Fluids2D/Fluids2DMain.cpp
GTE/Samples/Physics/Fluids2D/Fluids2DWindow2.{h,cpp}

Modified comments that referred to the old file names `GteSOMEFILE.h`, replacing them by `SOMEFILE.h`.

GTE/Applications/Window2.cpp
GTE/Graphics/DX11/DX11InputLayout.h
GTE/Mathematics/ApprCone3.h
GTE/Mathematics/ApprTorus3.h
GTE/Mathematics/BSNumber.h
GTE/Mathematics/BSplineReduction.h
GTE/Mathematics/Cone.h
GTE/Mathematics/Math.h
GTE/Mathematics/MinimumAreaCircle2.h
GTE/Mathematics/MinimumVolumeSphere3.h
GTE/Mathematics/Quaternion.h
GTE/Mathematics/SlerpEstimate.h

June 29, 2022. The `IsOnSurface` function is a copy-and-paste error from the implicit curve file. I modified the name to `IsOnCurve`. I also set the output `curvature` on early exit in order not to leave output uninitialized.

GTE/Mathematics/ImplicitCurve2.h

Greatly simplified the implementation based on a revision of the document [PrincipleCurvature.pdf](#).

[GTE/Mathematics/ImplicitSurface3.h](#)

June 26, 2022. Added a new class [VETManifoldMeshNR](#) which represents a vertex-edge-triangle manifold mesh for which triangles are provided as a single batch and NO REMOVE operations are going to be performed on the mesh. [VETManifoldMeshNR](#) significantly outperforms [VETManifoldMesh](#). [VETManifoldMesh](#) is general purpose, allowing triangle insertions and triangle removals. The underlying C++ container classes lead to significant memory allocation/deallocation costs and are also expensive for find operations. It turns out that the design of [VETManifoldMeshNR](#) automatically gives you vertex adjacency information, so there is no [ETManifoldMeshNR](#) implementation. It is a requirement that the input triangles form a manifold mesh with consistently ordered triangles. In most applications, this requirement is already satisfied.

[GTE/Mathematics/VETManifoldMeshNR.h](#)

The [VETManifoldMeshNR](#) class is designed to minimize the allocations. For example, using this class for a collection of 1131652 positions and 2242293 triangles, the CPU times on an Intel (R) Core (TM) i9-10900 are as follows.

	ETManifoldMesh	VETManifoldMeshNR
graph creation	2234 milliseconds	118 milliseconds
connected component labeling	2169 milliseconds	40 milliseconds
get boundary polygons	232 milliseconds	25 milliseconds

Added comments to indicate that [VETManifoldMeshNR](#) might be a better choice for your application if it meets the preconditions.

[GTE/Mathematics/ETManifoldMesh.h](#)

[GTE/Mathematics/VETManifoldMesh.h](#)

3 Updates to Version 6.3

June 8, 2022. Added new files that contain an efficient implementation of natural cubic splines and natural quintic splines. The PDF documentation has been revised accordingly, <https://www.geometrictools.com/Documentation/NaturalSplines.pdf>. The class [NaturalSplineCurve](#) will be deprecated in favor of the new classes.

[GTE/Mathematics/NaturalCubicSpline.h](#)

[GTE/Mathematics/NaturalQuinticSpline.h](#)

Fixed gcc compiler warnings (signed/unsigned integer comparisons).

[GTE/Mathematics/NaturalSplineCurve.h](#)

Fixed gcc errors about the keyword `template` required before the dependent typenames involving the `Degree*` function calls. The errors occurred with gcc 12.1.1 on Fedora 36 but not with gcc 9.3.0 on Ubuntu or with Microsoft compilers.

GTE/Mathematics/ASinEstimate.h
GTE/Mathematics/ExpEstimate.h
GTE/Mathematics/LogEstimate.h

Fixed an error in the deprecated file for `UniqueVerticesTriangles`; the `<array>` header is not explicitly included. This occurred with gcc 12.1.1 on Fedora 36 even though my instantiation tool did not include explicit instantiation for that class. The error did not occur with gcc 9.3.0 on Ubuntu 20.04.1 LTS or Microsoft compilers when running the tool.

GTE/Mathematics/UniqueVerticesTriangles.h

Fixed bugs in a block of code that is not exercised by the `ShaderReflection` sample application. This problem was caught by gcc 12.1.1 on Fedora 36.

GTE/Graphics/GL45/GLSLReflection.cpp

Modified code to avoid a potentially uninitialized variable warning from gcc 12.1.1 on Fedora 36.

GTE/Samples/Geometrics/IncrementalDelaunay2/IncrementalDelaunay2Window2.cpp
GTE/Samples/Intersection/IntersectTriangles2D/IntersectTriangles2DWindow2.cpp
GTE/Samples/Mathematics/Interpolation2DWindow3.cpp

Fedora 36 with gcc 12.1.1 complains about potential index-out-of-range because `Evaluate` references all possible elements, but the compiler is unaware that the logic of `Evaluate` is designed to assign only those elements that are valid. For now, I am passing in an array of 3-tuples knowing that only one will be filled in. Added arrays of N -tuples to avoid the warnings.

GTE/Mathematics/ParametricCurve.h
GTE/Mathematics/ParametricSurface.h
GTE/Samples/Mathematics/BSplineCurveReduction/BSplineCurveReductionWindow3.cpp

June 4, 2022. The constructor for free or closed splines allocated more than enough elements for `mCoefficients` and the `GetNumPoints()` function returned the number of points based on the size of `mCoefficients`. The constructor for clamped splines allocated exactly the correct number of elements for `mCoefficients`, but then `GetNumPoints()` returned the incorrect number of points. This caused too few points to be processed in clamped splines. This occurred when the optimization was added to minimize memory allocations and deallocations for free splines. The optimization code has been reworked and the same memory optimization was added for closed and clamped splines. The unit tests pass for all 3 flavors of splines.

GTE/Mathematics/NaturalSplineCurve.h

May 30, 2022. The `IsPowerOfTwo(int32_t)` function called itself (infinite recursion). Modified the static cast to use `uint32_t` as intended.

GTE/Mathematics/BitHacks.h

May 16, 2022. In the GTL development, fixed a bug in the `Inverse` function for `Transform<T>` objects. The unit tests were incomplete in that they tested the inverse directly using queries to the internal matrices of `Transform<T>` rather than verifying the correctness of the output of `Inverse`. The corrections were backported to GTE.

GTE/Mathematics/Transform.h

The `RectangleManager` code was copied, pasted and modified to produce the `BoxManager` code. The comments were part of this and the `BoxManager` code referred to rectangles rather than boxes. Also, non-const references in both classes were modified to const references.

GTE/Mathematics/BoxManager.h
GTE/Mathematics/RectangleManager.h

4 Updates to Version 6.2

April 3, 2022. By user request, on Microsoft Windows I added virtual `OnWindowsMessage` to `WindowApplication` and called by `WindowSystem::WindowProcedure`. This allows the application to use a third-party UI package, processing windows messages intended for that package. A return value of `true` tells the window procedure not to pass the message to the application window.

GTE/Applications/WindowApplication.{h,cpp}
GTE/Applications/MSW/WindowSystem.cpp

April 2, 2022. Ported the Wild Magic 5 graphics sample `ShadowMaps` to GTE6. This is the last of the Wild Magic code I had planned on porting.

GTE/BuildAll*.
GTE/Samples/Graphics/CMakeLists.txt
GTE/Samples/Graphics/ShadowMaps/*
GTE/Samples/Data/Stone.png

March 29, 2022. Ported the Wild Magic 5 `MeshSmoother` class to GTE6.

GTE/Mathematics/MeshSmoother.h

March 28, 2022. I finally trapped the problem with Windows OpenGL (WGL) where the first-drawn frame is not correct. I added a swap-buffers call immediately after creating the `WGLEngine` object (in the application layer), which eliminated the problem.

GTE/Applications/MSW/WindowSystem.cpp

Ported the Wild Magic 5 skinning sample to GTE6.

```
GTE/BuildAll*.*
GTE/Samples/Graphics/CMakeLists.txt
GTE/Samples/Graphics/Skinning/*
```

Added new constructors to avoid the repeated pattern of default construction of `MeshFactory` followed by the `SetVertexFormat` call.

```
GTE/Graphics/MeshFactory.{h,cpp}
```

Ported the line-torus find-intersection query from Wild Magic5 to GTE6 (at user request). Added a sample application to show that the code is working correctly. Internal unit tests have also been added.

```
GTE/BuildAll*.*
GTE/GTMathematics*.*
GTE/Mathematics/IntrLine3Torus3.h
GTE/Samples/Intersection/CMakeLists.txt
GTE/Samples/Intersection/IntersectLine3Torus3/*
```

March 25, 2022. The current 2D segment-segment queries use the find-intersection query for two lines, but the segments are converted to center-direction-extent form. The two-point representation has endpoints p_0 and p_1 and the parameterization is $p_0 + t(p_1 - p_0)$ for $t \in [0, 1]$. The center-direction-extent form is $c + sd$ for $|s| \leq e$. The center is $c = (p_0 + p_1)/2$, the direction is $d = (p_1 - p_0)/|p_1 - p_0|$ and the extent is $e = |p_1 - p_0|/2$. The computation of d involves a normalization which generally cannot be computed exactly with rational arithmetic. The computation of e is also not exact. The segment parameters returned from the queries are relative to s , not to t . The old queries are `operator()` functions and still exist. To allow for exact rational computation, I added new queries named `Exact` which uses the two-point form for segments. [In GTL development, I have been eliminating the use of the segment center-direction-extent form.]

```
GTE/Mathematics/IntrSegment2Segment2.h
```

The code had a call `Normalize(diff)` that is not necessary because the sign tests dependent on `diff` are (theoretically) independent of the length of `diff`. Removing the normalize call now allows the code to produce theoretically correct results when type `T` is a rational type.

```
GTE/Mathematics/IntrLine2Line2.h
```

Replace parenthesized casts by `static_casts`.

```
GTE/Mathematics/IntrLine2Line2.h
GTE/Mathematics/IntrLine2Ray2.h
GTE/Mathematics/IntrLine2Segment2.h
GTE/Mathematics/IntrRay2Ray2.h
GTE/Mathematics/IntrRay2Segment2.h
GTE/Mathematics/IntrSegment2Segment2.h
```

March 23, 2022. The [SetRow](#) function was missing a return statement.

[GTE/Mathematics/GMatrix.h](#)

March 22, 2022. Added abstract base classes for implicit curves in 2D, class [ImplicitCurve2](#), and implicit surface in 3D, class [ImplicitSurface3](#). The latter is a port of [ImplicitSurface](#) from Wild Magic 5. These classes support computation of coordinate frames and curvature information. For parametric curves and surfaces, see [FrenetFrame](#) and [DarbouxFrame](#).

[GTE/GTMathematics.*](#)
[GTE/Mathematics/ImplicitCurve2.h](#)
[GTE/Mathematics/ImplicitSurface3.h](#)

Ported the Wild Magic 5 graphics sample [VolumeFog](#) to GTE6.

[GTE/BuildAll*](#)
[GTE/GTGraphics.*.*](#)
[GTE/Graphics/GTGraphics.h](#)
[GTE/Graphics/VolumeFogEffect.{h,cpp}](#)
[GTE/Samples/Graphics/CMakeLists.txt](#)
[GTE/Samples/Graphics/VolumeFog/*](#)
[GTE/Data/BlueSky.png](#)
[GTE/Data/Shader/VolumeFogEffect.{vs,ps}.{hlsl,glsl}](#)

Fixed the minor number in file versions. Looks like versioning is the bane of the month.

[GTE/Graphics/BoundTree.h](#)
[GTE/Graphics/CollisionGroup.h](#)
[GTE/Graphics/CollisionMesh.{h,cpp}](#)
[GTE/Graphics/CollisionRecord.h](#)
[GTE/Graphics/PlanarShadowEffect.{h,cpp}](#)
[GTE/Mathematics/ApproxCone3EllipseAndPoints.h](#)
[GTE/Samples/Mathematics/FitConeByEllipseAndPoints/FitConeByEllipseAndPoints*.{h,cpp}](#)
[GTE/Samples/Physics/CollisionsBoundTree/CollisionsBoundTree*.{h,cpp}](#)
[GTE/Samples/Physics/CollisionsMovingSpheres/CollisionsMovingSpheres*.{h,cpp}](#)
[GTE/Samples/Physics/CollisionsMovingSphereTriangle/CollisionsMovingSphereTriangle*.{h,cpp}](#)

March 21, 2022. Ported the Wild Magic 5 sample [CollisionsMovingSphereTriangle](#) to GTE6.

[GTE/BuildAll*](#)
[GTE/Samples/Physics/CMakeLists.txt](#)
[GTE/Samples/Physics/CollisionsMovingSphereTriangle/*](#)

March 20, 2022. Ported the Wild Magic 5 sample [CollisionsMovingSpheres](#) to GTE6. The [CollisionsBoundTree](#) sample should have been in the Physics folder.

```

GTE/BuildAll*
GTE/Samples/SceneGraphs/CMakeLists.txt
GTE/Samples/Physics/CMakeLists.txt
GTE/Samples/Physics/CollisionsMovingSpheres/*
GTE/Samples/SceneGraphs/CollisionsBoundTree/*
GTE/Samples/Physics/CollisionsBoundTree/*

```

Ported the Wild Magic 5 collision detection code to GTE6: [BoundTree](#), [CollisionGroup](#) and [CollisionRecord](#). Added a wrapper class, [CollisionMesh](#), that provides the interface to a [Visual](#)-based triangle mesh to be used at a template parameter for the collision detection code. Added test-intersection and find-intersection queries for moving triangles. Ported the WM5 sample [CollisionsBoundTree](#) to illustrate use of the collision detection.

```

GTE/GTGraphics.{v16,v16}.vcxproj
GTE/GTGraphics.{v16,v16}.vcxproj.filters
GTE/Graphics/GTGraphics.h
GTE/Graphics/CMakeLists.txt
GTE/Mathematics/IntrTriangle3Triangle3.h
GTE/Samples/CMakeLists.txt
GTE/Graphics/BoundTree.h
GTE/Graphics/CollisionGroup.h
GTE/Graphics/CollisionRecord.h
GTE/Graphics/CollisionMesh.{h,cpp}
GTE/Samples/SceneGraphs/CollisionsBoundTree/*

```

I forgot to update the minor version from 1 to 2 when I posted GTE6.2. Time to automate this step.

```

GTE/CMakeLists.txt
GTE/Samples/CMakeLists.txt
GTE/Samples/Distance/CMakeLists.txt
GTE/Samples/Geometrics/CMakeLists.txt
GTE/Samples/Graphics/CMakeLists.txt
GTE/Samples/Imagics/CMakeLists.txt
GTE/Samples/Intersection/CMakeLists.txt
GTE/Samples/Mathematics/CMakeLists.txt
GTE/Samples/Physics/CMakeLists.txt
GTE/Samples/SceneGraphs/CMakeLists.txt

```

March 18, 2022. Fixed a bug in [ContainsPoint](#). Modified the internal unit tests to include the test that exposes the bug.

```

GTE/Mathematics/IntrTriangle3Triangle3.h

```

Removed the MSVS 2015 and MSVS 2017 versions of the C#/managed/C++ projects because I no longer support these compilers. The MSVS 2019 projects still exist. I added projects for MSVS 2022. Fixed the path problem in the MSVS 2019 version (modified environment variable from [GTE4.PATH](#) to [GTE.PATH](#)).

```

GTE/Samples/CSharpCppManaged/CSharpCppManaged.{v14,v15}.sln
GTE/Samples/CSharpCppManaged/CSharpApplication/CSharpApplication.{v14,v15}.csproj

```

GTE/Samples/CSharpCppManaged/CppLibrary/CppLibrary.{v14,v15}.*
 GTE/Samples/CSharpCppManaged/ManagedLibrary/ManagedLibrary.{v14,v15}.*
 GTE/Samples/CSharpCppManaged/CppLibrary/CppLibrary.v16.vcxproj
 GTE/Samples/CSharpCppManaged/CSharpCppManaged.v16.sln
 GTE/Samples/CSharpCppManaged/CSharpApplication/CSharpApplication.v16.csproj
 GTE/Samples/CSharpCppManaged/CppLibrary/CppLibrary.v16.*
 GTE/Samples/CSharpCppManaged/ManagedLibrary/ManagedLibrary.v16.*

March 13, 2022. The `Math.h` file generated compiler errors when I used one of the inline non-templated functions (needed `<stdint>`). The `UniqueVerticesTriangles.h` file generated compiler errors when I tried to use it (needed `<set>`). This file is deprecated, but until it is removed it needs to compile. I updated my internal instantiation tool to handle these files appropriately.

GTE/Mathematics/Math.h
 GTE/UniqueVerticesTriangles.h

5 Updates to Version 6.1

March 7, 2022. The planar reflection constructor had out-of-order initialization of members, and gcc on Linux complained about this. The Microsoft compilers (2019/2022) did not complain. I also removed a block of code in the constructor (unexposed in a conditional compilation block) that was moved to a class member function.

GTE/Graphics/PlanarReflectionEffect.cpp

The change to the interface for `PlanarReflectionEffect` required changes to the bouncing ball sample application.

GTE/Samples/Physics/BouncingBall/BouncingBallWindow3.{h,cpp}

The `MinimumWidthPoints2` class is templated with class `T`. It contained a line `using RotatingCalipers = typename RotatingCalipers<T>` which Microsoft compilers allow. The gcc compilers complained about the `typename`, so I removed it. I removed two unused variables. There was a sign mismatch in a loop counter.

GTE/Mathematics/MinimumWidthPoints2.h

Removed unused lines of code (old iterator code that was replaced by range-based iteration) and fixed a sign mismatch in a loop. Removed the declaration for `static constexpr maxFloat = std::numeric_limits<float>::max();` and now use the `max()` value itself in the code. I must be missing something about declaring class-member `constexpr` values; the loader for gcc complained that `maxFloat` is undefined. The Microsoft compilers (2019/2022) never complain about such a declaration.

GTE/Graphics/CLODMeshCreator.h

An unreferenced variable for the camera position needed to be accessed in two lines of code rather than re-lookup the variable.

GTE/Samples/Physics/DLODNodes/DLODNodesWindow3.cpp

The gcc compiler complained about the compound Boolean expression in the `GTE_ASSERT` statement in `SwitchNode::SetActiveChild`. It wanted parentheses around the `and` expression.

GTE/Graphics/SwitchNode.cpp

The CMake list of source files needed to be updated for `PlanarShadowEffect.cpp`, `SwitchNode.cpp`, `CLODMesh.cpp` and `DLODNode.cpp`.

GTE/Graphics/CMakeLists.txt

March 6, 2022. I wrote a GTE6 sample application for planar shadows that is similar to the one for Wild Magic 5. I revised the planar reflections code so that the two effects have essentially the same design.

GTE/BuildAll*.sln
GTE/Graphics/PlanarShadowEffect.{h,cpp}
GTE/Graphics.h
GTE/Samples/Graphics/PlanarShadows/PlanarShadows*.vcxproj
GTE/Samples/Graphics/PlanarShadows/PlanarShadows*.vcxproj.filter
GTE/Samples/Graphics/PlanarShadows/PlanarShadowsMain.cpp
GTE/Samples/Graphics/PlanarShadows/PlanarShadowsWindow3.{h,cpp}
GTE/Samples/Graphics/PlanarShadows/PlanarShadows.code-workspace
GTE/Samples/Graphics/PlanarShadows/CMakeLists.txt
GTE/Samples/Graphics/PlanarShadows/CMakeSample.sh
GTE/Samples/Graphics/PlanarShadows/cmake-variants.json
GTE/Samples/Graphics/PlanarShadows/.vscode/launch.json
GTE/Samples/Graphics/PlanarShadows/.vscode/settings.json
GTE/Samples/Graphics/CMakeLists.txt
GTE/Graphics/PlanarReflectionEffect.{h,cpp}
GTE/Samples/Graphics/PlanarReflections/PlanarReflectionsMain.cpp
GTE/Samples/Graphics/PlanarReflections/PlanarReflectionsWindow3.{h,cpp}

March 4, 2022. I wrote a GTE6 sample application for planar reflections that is similar to the one for Wild Magic 5. The `PlanarReflectionEffect` drawing function had a bug. It needed to set the back-face stencil parameters in addition to the front-face stencil parameters. Without the back-face parameters, the reflection caster was drawn on both sides of the reflection plane.

GTE/BuildAll*.sln
GTE/Samples/Graphics/PlanarReflections/PlanarReflections*.vcxproj
GTE/Samples/Graphics/PlanarReflections/PlanarReflections*.vcxproj.filter
GTE/Samples/Graphics/PlanarReflections/PlanarReflectionsMain.cpp
GTE/Samples/Graphics/PlanarReflections/PlanarReflectionsWindow3.{h,cpp}
GTE/Samples/Graphics/PlanarReflections/PlanarReflections.code-workspace
GTE/Samples/Graphics/PlanarReflections/CMakeLists.txt
GTE/Samples/Graphics/PlanarReflections/CMakeSample.sh


```
GTE/Samples/Graphics/PlanarReflections/cmake-variants.json
GTE/Samples/Graphics/PlanarReflections/.vscode/launch.json
GTE/Samples/Graphics/PlanarReflections/.vscode/settings.json
GTE/Samples/Graphics/CMakeLists.txt
GTE/PlanarReflectionEffect.{h,cpp}
```

After looking at the requirements for the feature request, I modified the return value of `MinimumWidthPoints2::operator()` to be `OrientedBox2<T>` to be consistent with the minimum-area box code. The sample application now draws the minimum-area box and the minimum-width box for comparison.

```
GTE/Mathematics/MinimumWidthPoints2.h
GTE/Samples/Geometrics/MinimumAreaBox2DWindow2.{h,cpp}
```

March 3, 2022. Added an implementation of the rotating calipers algorithm, class `RotatingCalipers`. The goal was to refactor `MinimumAreaBox2` to pull out the rotating calipers. However, this does not work because the minimum-area algorithm uses a pair of rotating calipers that work together. Added a new class `MinimumWidthPoints2` that uses the rotating calipers algorithm to compute the width of a 2D point set. For now I have modified the `MinimumAreaBox2D` sample application to show also the results from computing the width of a point set. I also cleaned up the comments to use current formatting rules. And I added functions to allow passing `std::vector` inputs rather than raw pointers.

```
GTE/Mathematics/RotatingCalipers.h
GTE/Mathematics/MinimumWidthPoints2.h
GTE/Mathematics/MinimumAreaBox2.h
GTE/Mathematics/GTMathematics*.vcxproj
GTE/Mathematics/GTMathematics*.vcxproj.filters
GTE/Samples/Geometrics/MinimumAreaBox2DWindow2.{h,cpp}
```

The `ConvexHull2` class was supposed to have switched from `FPInterval` to `SWInterval`, but I apparently overlooked this. The `FPInterval` class uses floating-point hardware operations to support interval arithmetic; however, the GCC compiler ignores the changes to the floating-point environment (it does not support modifying the rounding mode of the FPU). The `SWInterval` class implements interval arithmetic and rounds using software operations.

```
GTE/Mathematics/ConvexHull2.h
```

Two class members were not initialized by the constructor.

```
GTE/Graphics/DLODNode.cpp
```

February 25, 2022. Feature request to fit a cone to known elliptical cross sections and some additional points (`ApprCone3EllipseAndPoints`). If the cross sections are provided as point samples approximately on the ellipses, the new code also includes a class to extract points for each ellipse because the points might be stored in a file in some unknown order (`ApprCone3ExtractEllipses`). 3D ellipses are then fit to each subset of points in order to obtain the ellipse input to `ApprCone3EllipseAndPoints`. A PDF has been added to the documentation to describe the fitting algorithm, `FitConeToEllipseAndPoints.pdf`.

GTE/BuildAll*.sln
 GTE/Mathematics/GTMathematics*.vcxproj
 GTE/Mathematics/GTMathematics*.vcxproj.filters
 GTE/Mathematics/ApprCone3EllipseAndPoints.h
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/FitConeByEllipseAndPoints*.vcxproj
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/FitConeByEllipseAndPoints*.vcxproj.filter
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/FitConeByEllipseAndPointsMain.cpp
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/FitConeByEllipseAndPointsWindow3.{h,cpp}
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/Data/CircleAndVertex.txt
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/Data/OneCircleOneEllipse.txt
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/Data/TwoEllipses.txt
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/Data/TwoPartialEllipses.txt
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/FitConeByEllipseAndPoints.code-workspace
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/CMakeLists.txt
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/CMakeSample.sh
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/cmake-variants.json
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/.vscode/launch.json
 GTE/Samples/Mathematics/FitConeByEllipseAndPoints/.vscode/settings.json
 GTE/Samples/Mathematics/CMakeLists.txt

February 24, 2022. Ported the WM5 classes [ClodMesh](#), [CreateClodMesh](#) and [CollapseRecord](#) to GTE6 classes [CLODMesh](#), [CLODMeshCreator](#) and [CLODCollapseRecord](#). Ported the WM5 sample [ClodMeshes](#) to the GTE6 sample [CLODMeshes](#).

GTE/BuildAll*.sln
 GTE/Graphics/GTGraphics.h
 GTE/Graphics/GTGraphics*.vcxproj
 GTE/Graphics/GTGraphics*.vcxproj.filters
 GTE/Graphics/CLODMeshCreator.h
 GTE/Graphics/CLODMesh.{h,cpp}
 GTE/Graphics/CLODCollapseRecord.h
 GTE/Samples/SceneGraphs/CLODMeshes/CLODMeshes*.vcxproj
 GTE/Samples/SceneGraphs/CLODMeshes/CLODMeshes*.vcxproj.filter
 GTE/Samples/SceneGraphs/CLODMeshes/CLODMeshesMain.cpp
 GTE/Samples/SceneGraphs/CLODMeshes/CLODMeshesWindow3.{h,cpp}
 GTE/Samples/SceneGraphs/CLODMeshes/Data/FunctionX64Y64R8.png
 GTE/Samples/SceneGraphs/CLODMeshes/CLODMeshes.code-workspace
 GTE/Samples/SceneGraphs/CLODMeshes/CMakeLists.txt
 GTE/Samples/SceneGraphs/CLODMeshes/CMakeSample.sh
 GTE/Samples/SceneGraphs/CLODMeshes/cmake-variants.json
 GTE/Samples/SceneGraphs/CLODMeshes/.vscode/launch.json
 GTE/Samples/SceneGraphs/CLODMeshes/.vscode/settings.json
 GTE/Samples/SceneGraphs/CMakeLists.txt

February 11, 2022. Modified the [GenerateProject](#) tool to generate the Visual Studio Code workspace files for sample applications.

GTE/Tools/GenerateProject/GenerateProject.*

GTE/Tools/GenerateProject/ProjectTemplateVSCode.{h,cpp}

Added missing *.code-workspace files to the recently ported physics samples.

GTE/Samples/Physics/BouncingSpheres/BouncingSpheres.code-workspace
GTE/Samples/Physics/BouncingTetrahedra/BouncingTetrahedra.code-workspace
GTE/Samples/Physics/RoughPlaneFlatBoard/RoughPlaneFlatBoard.code-workspace
GTE/Samples/Physics/RoughPlaneParticle1/RoughPlaneParticle1.code-workspace
GTE/Samples/Physics/RoughPlaneParticle2/RoughPlaneParticle2.code-workspace
GTE/Samples/Physics/RoughPlaneSolidBox/RoughPlaneSolidBox.code-workspace
GTE/Samples/Physics/RoughPlaneThinRod1/RoughPlaneThinRod1.code-workspace
GTE/Samples/Physics/RoughPlaneThinRod2/RoughPlaneThinRod2.code-workspace
GTE/Samples/Physics/WaterDropFormation/WaterDropFormation.code-workspace
GTE/Samples/Physics/WrigglingSnake/WrigglingSnake.code-workspace

Ported the SwitchNode class and SwitchNodes sample application from Wild Magic 5 to Geometric Tools Engine 6.

GTE/BuildAll*.sln
GTE/Graphics/GTGraphics.h
GTE/Graphics/GTGraphics*.vcxproj
GTE/Graphics/GTGraphics*.vcxproj.filters
GTE/Graphics/SwitchNode.{h,cpp}
GTE/Samples/SceneGraphs/SwitchNodes/SwitchNodes*.vcxproj
GTE/Samples/SceneGraphs/SwitchNodes/SwitchNodes*.vcxproj.filter
GTE/Samples/SceneGraphs/SwitchNodes/SwitchNodesMain.cpp
GTE/Samples/SceneGraphs/SwitchNodes/SwitchNodesWindow3.{h,cpp}
GTE/Samples/SceneGraphs/SwitchNodes/SwitchNodes.code-workspace
GTE/Samples/SceneGraphs/SwitchNodes/CMakeLists.txt
GTE/Samples/SceneGraphs/SwitchNodes/CMakeSample.sh
GTE/Samples/SceneGraphs/SwitchNodes/cmake-variants.json
GTE/Samples/SceneGraphs/SwitchNodes/.vscode/launch.json
GTE/Samples/SceneGraphs/SwitchNodes/.vscode/settings.json
GTE/Samples/SceneGraphs/CMakeLists.txt

Ported the DLODNode (dynamic level of detail node) class and DLODNodes sample application from Wild Magic 5 to Geometric Tools Engine 6.

GTE/BuildAll*.sln
GTE/Graphics/GTGraphics.h
GTE/Graphics/GTGraphics*.vcxproj
GTE/Graphics/GTGraphics*.vcxproj.filters
GTE/Graphics/DLODNode.{h,cpp}
GTE/Samples/SceneGraphs/DLODNodes/DLODNodes*.vcxproj
GTE/Samples/SceneGraphs/DLODNodes/DLODNodes*.vcxproj.filter
GTE/Samples/SceneGraphs/DLODNodes/DLODNodesMain.cpp

GTE/Samples/SceneGraphs/DLODNodes/DLODNodesWindow3.{h,cpp}
GTE/Samples/SceneGraphs/DLODNodes/DLODNodes.code-workspace
GTE/Samples/SceneGraphs/DLODNodes/CMakeLists.txt
GTE/Samples/SceneGraphs/DLODNodes/CMakeSample.sh
GTE/Samples/SceneGraphs/DLODNodes/cmake-variants.json
GTE/Samples/SceneGraphs/DLODNodes/.vscode/launch.json
GTE/Samples/SceneGraphs/DLODNodes/.vscode/settings.json
GTE/Samples/SceneGraphs/CMakeLists.txt

6 Updates to Version 6.0

February 7, 2022. Fixed a compiler error on Linux for a mismatch in `size_t` and `int32_t`. Changed the type of `mNumCtrlPoints` and `mDegree` to `size_t`.

GTE/Samples/Physics/WrigglingSnake/WrigglingSnakeWindow3.h

February 6, 2022. Ported the Wild Magic 5 physics sample `BouncingTetrahedra` to GTE. The port of `BouncingSpheres` was modified so that the two samples have the same conceptual framework.

GTE/Samples/Physics/BouncingTetrahedra/BouncingTetrahedra*.vcxproj
GTE/Samples/Physics/BouncingTetrahedra/BouncingTetrahedra*.vcxproj.filter
GTE/Samples/Physics/BouncingTetrahedra/BouncingTetrahedraMain.cpp
GTE/Samples/Physics/BouncingTetrahedra/BouncingTetrahedraWindow3.{h,cpp}
GTE/Samples/Physics/BouncingTetrahedra/PhysicsModule.{h,cpp}
GTE/Samples/Physics/BouncingTetrahedra/RigidPlane.{h,cpp}
GTE/Samples/Physics/BouncingTetrahedra/RigidTetrahedron.{h,cpp}
GTE/Samples/Physics/BouncingTetrahedra/Initial.txt
GTE/Samples/Physics/BouncingSpheres/BouncingSpheres*.vcxproj
GTE/Samples/Physics/BouncingSpheres/BouncingSpheres*.vcxproj.filter
GTE/Samples/Physics/BouncingSpheres/BouncingSpheresMain.cpp
GTE/Samples/Physics/BouncingSpheres/BouncingSpheresWindow3.{h,cpp}
GTE/Samples/Physics/BouncingSpheres/PhysicsModule.{h,cpp}
GTE/Samples/Physics/BouncingSpheres/BouncingSpheresMain.cpp
GTE/Samples/Physics/BouncingSpheres/BouncingSpheresWindow3.{h,cpp}
GTE/Samples/Physics/BouncingSpheres/RigidPlane.{h,cpp}
GTE/Samples/Physics/BouncingSpheres/RigidSphere.{h,cpp}
GTE/Samples/Physics/BouncingSpheres/Initial.txt

I factored out the impulse computations from the physics samples and moved them to `RigidBody`. The code now can be shared by the applications.

GTE/Mathematics/RigidBody.h

The `Tetrahedron3` class has additional framework to support computing normals at features (vertices, edges, faces). The `DistPoint3Tetrahedron3` query was modified because of an interface change in `Tetrahedron3`.

GTE/Mathematics/Tetrahedron3.h
GTE/Mathematics/DistPoint3Tetrahedron3.h

The algorithm I implemented for distance between tetrahedra was not robust because of testing for containment of vertices of one tetrahedron in another. I modified it to test centroids instead.

GTE/Mathematics/DistTetrahedron3Tetrahedron3.h

Added a new file for the test-intersection query between tetrahedra using the method of separating axes. This allowed for better performance in [BouncingTetrahedra](#) compared to using distance between tetrahedra.

GTE/Mathematics/IntrTetrahedron3Tetrahedron3.h

Fixed comments in the file.

GTE/Mathematics/IntrOrientedBox3OrientedBox3.h

February 3, 2022. Added missing projects from the MSVS 2019 build-all solutions. Added new build-all solutions for MSVS 2022.

GTE/BuildAll.v16.sln
GTE/BuildAllDX11.v16.sln
GTE/BuildAllGL45.v16.sln
GTE/BuildAll.v17.sln
GTE/BuildAllDX11.v17.sln
GTE/BuildAllGL45.v17.sln

February 1, 2022. Added new file that implements tetrahedron-tetrahedron query. This is needed for the port of [BouncingTetrahedra](#) from Wild Magic 5 to Geometric Tools Engine.

GTE/Mathematics/DistTetrahedron3Tetrahedron3.h

Modified the interfaces for computing barycentric coordinates. The raw arrays [bary\[\]](#) were replaced by [std::array](#) objects.

GTE/Mathematics/Vector2.h
GTE/Mathematics/Vector3.h
GTE/Mathematics/Delaunay2Mesh.h
GTE/Mathematics/Delaunay3Mesh.h
GTE/Mathematics/DistPoint3Tetrahedron3.h
GTE/Mathematics/IntpQuadraticNonuniform2.h
GTE/Mathematics/PlanarMesh.h
GTE/Samples/Mathematics/Interpolation2D/Interpolation2DWindow3.cpp

Added a function for computing the centroid of a tetrahedron.

GTE/Mathematics/Tetrahedron3.h

Updated a comment to be consistent with other distance-query comments.

GTE/Mathematics/DistTriangle3Triangles.h

Updated project files that reference the newly added mathematics header files.

GTE/Mathematics/GTMathematics.*.{vcxproj,vcxproj.filters}

January 31, 2022. Added new file that implements point-in-tetrahedron query. This is needed for the port of [BouncingTetrahedra](#) from Wild Magic 5 to Geometric Tools Engine.

GTE/Mathematics/ContTetrahedron3.h

January 30, 2022. The point-tetrahedron distance query code failed to compile because of a mismatch in index type. (Reorganization of my internal tools caused the template instantiation tool not to launch when [Tetrahedron3](#) was modified.)

GTE/Mathematics/DistPoint3Tetrahedron3.h

January 26, 2022. I modified the sample application to allow the spheres to spin based on angular momentum, taking into account transfer of momentum during colliding contact. This required a different algorithm for impulsive function construction than what is in “Game Physics, 2nd edition” (Section 6.2.2). The algorithm details are in a new document at my website, [ComputingImpulsiveForces.pdf](#). I also added friction to stop eventually the spheres from sliding and spinning when they are on the floor. Eliminated the texture seam by hacking it to a toroidal texture. Modified the RigidBody interface for simpler presentation of the Runge-Kutta solver steps and to make it clear the dependency between physics parameters and quantities derived from them (to support synchronizing the state).

GTE/Mathematics/RigidBody.h

GTE/Samples/Physics/BouncingSpheres/BouncingSpheresMain.cpp

GTE/Samples/Physics/BouncingSpheres/BouncingSpheresWindow3.{h,cpp}

GTE/Samples/Physics/BouncingSpheres/PhysicsModule.{h,cpp}

GTE/Samples/Physics/BouncingSpheres/Initial.txt

GTE/Samples/Data/BallTextureWrap.png

January 17, 2022. The function [FitIndexed](#) has tests for the components of [mean](#) being finite floating-point numbers. The test for [mean\[2\]](#) was missing.

GTE/ApprOrthogonalPlane3.h

Converted the [GetFaceIndices](#) function to [static](#) because the indices for the triangular faces are the same no matter which tetrahedron. Added also the static function [GetAllFaceIndices](#) to access the entire array of indices as a single object; this is useful for index buffers associated with tetrahedra.

GTE/Tetrahedron3.h

January 16, 2022. I replaced the hard-coded physics time steps by a physics clock that requires the simulation to run at 60 frames per second. I also added comments that the physics implementation is based on Section 6.6 of my Game Physics book (2nd edition).

GTE/Samples/Physics/BouncingSpheres/BouncingSpheresWindow3.{h,cpp}

January 14, 2022. I ported the Wild Magic 5 physics sample BouncingSpheres to GTE 5. The sample illustrates impulsed-based physics for a collection of spheres. I refactored the code so that the physics portion is isolated to the [PhysicsModule.*](#) files. Update the CMake list of projects for physics.

GTE/Samples/Physics/BouncingSpheres/BouncingSpheres*.*
GTE/Samples/Physics/BouncingSpheres/PhysicsModule.{h,cpp}
GTE/Samples/Physics/CMakeLists.txt

The function [SetMass](#) had type [float](#) instead of the template parameter [Real](#).

GTE/Mathematics/RigidBody.h

January 12, 2022. I ported the Wild Magic 5 physics sample WaterDropFormation to GTE 6. Update the CMake list of projects for physics.

GTE/Samples/Physics/WaterDropFormation/WaterDropFormation*.*
GTE/Samples/Physics/WaterDropFormation/RevolutionSurface.{h,cpp}
GTE/Samples/Physics/CMakeLists.txt

Fixed the file versions to show 6.1 instead of 6.0 to be consistent with my versioning rules.

GTE/Samples/Physics/RoughPlaneParticle1/RoughPlaneParticle1*.*
GTE/Samples/Physics/RoughPlaneParticle1/PhysicsModule.{h,cpp}
GTE/Samples/Physics/RoughPlaneParticle2/RoughPlaneParticle2*.*
GTE/Samples/Physics/RoughPlaneParticle2/PhysicsModule.{h,cpp}
GTE/Samples/Physics/RoughPlaneThinRod1/RoughPlaneThinRod1*.*
GTE/Samples/Physics/RoughPlaneThinRod1/PhysicsModule.{h,cpp}
GTE/Samples/Physics/RoughPlaneThinRod2/RoughPlaneThinRod2*.*
GTE/Samples/Physics/RoughPlaneThinRod2/PhysicsModule.{h,cpp}
GTE/Samples/Physics/RoughPlaneFlatBoard/RoughPlaneFlatBoard*.*
GTE/Samples/Physics/RoughPlaneFlatBoard/PhysicsModule.{h,cpp}
GTE/Samples/Physics/RoughPlaneSolidBox/RoughPlaneSolidBox*.*
GTE/Samples/Physics/RoughPlaneSolidBox/PhysicsModule.{h,cpp}
GTE/Samples/Physics/WrigglingSnake/WrigglingSnake*.*
GTE/Samples/Physics/WrigglingSnake/TubeSurface.{h,cpp}

Added hyperlinks to the website PDF that describes some simple friction-based algorithms.

GTE/Samples/Physics/RoughPlaneParticle1/RoughPlaneParticle1.h
GTE/Samples/Physics/RoughPlaneParticle2/RoughPlaneParticle2.h
GTE/Samples/Physics/RoughPlaneThinRod1/RoughPlaneThinRod1.h
GTE/Samples/Physics/RoughPlaneThinRod2/RoughPlaneThinRod2.h
GTE/Samples/Physics/RoughPlaneFlatBoard/RoughPlaneFlatBoard.h
GTE/Samples/Physics/RoughPlaneSolidBox/RoughPlaneSolidBox.h

January 11, 2022. I ported the Wild Magic 5 physics sample WrigglingSnake to GTE 6. Updated the CMake list of projects for physics.

GTE/Samples/Physics/WrigglingSnake/WrigglingSnake*.*
GTE/Samples/Physics/WrigglingSnake/TubeSurface.{h,cpp}
GTE/Data/Snake.png
GTE/Samples/Physics/CMakeLists.txt

January 10, 2022. I ported the Wild Magic 5 physics samples to GTE 6, the ones involving friction.

GTE/Samples/Physics/RoughPlaneParticle1/RoughPlaneParticle1*.*
GTE/Samples/Physics/RoughPlaneParticle1/PhysicsModule.{h,cpp}
GTE/Samples/Physics/RoughPlaneParticle2/RoughPlaneParticle2*.*
GTE/Samples/Physics/RoughPlaneParticle2/PhysicsModule.{h,cpp}
GTE/Samples/Physics/RoughPlaneThinRod1/RoughPlaneThinRod1*.*
GTE/Samples/Physics/RoughPlaneThinRod1/PhysicsModule.{h,cpp}
GTE/Samples/Physics/RoughPlaneThinRod2/RoughPlaneThinRod2*.*
GTE/Samples/Physics/RoughPlaneThinRod2/PhysicsModule.{h,cpp}
GTE/Samples/Physics/RoughPlaneFlatBoard/RoughPlaneFlatBoard*.*
GTE/Samples/Physics/RoughPlaneFlatBoard/PhysicsModule.{h,cpp}
GTE/Samples/Physics/RoughPlaneSolidBox/RoughPlaneSolidBox*.*
GTE/Samples/Physics/RoughPlaneSolidBox/PhysicsModule.{h,cpp}

I removed the post-build copies of the executables to the folder [GeometricTools/GTE/Executable](#). The tools are used infrequently enough that I now run them from their [.Output](#) folders. Naturally, you can manually copy the executables to where you want them.

GTE/Tools/BitmapFontCreator/BitmapFontCreator*.vcxproj
GTE/Tools/GenerateProject/GenerateProject*.vcxproj

January 9, 2022. I added projects and solutions for Microsoft Visual Studio 2022. I removed the Microsoft Visual Studio 2015 projects and solutions because that version of the IDE reached mainstream end date on October 13, 2020. Microsoft Visual Studio 2017 reaches its mainstream end date on April 12, 2022. However, I have also removed the Microsoft Visual Studio 2017 projects and solutions because I do not have enough time to maintain so many versions of the compiler. From now on, I will support two versions of MSVS, the current one (whatever that is) and the previous one.

The Tools folder has various projects that were expanded to include projects and solutions for MSVS 2022. The [GenerateProject](#) tool has been modified to generate only MSVS 2019 and MSVS 2022 projects and solutions.


```

GTE/Tools/BitmapFontCreator/BitmapFontCreator.v17.*
GTE/Tools/BitmapFontCreator/BitmapFontCreator.v16.vcxproj
GTE/Tools/ChangePlatformToolset/ChangePlatformToolset.v17.*
GTE/Tools/ChangePlatformToolset/ChangePlatformToolset.v16.vcxproj
GTE/Tools/FiniteDifferences/FiniteDifferencesDX11.v17.*
GTE/Tools/FiniteDifferences/FiniteDifferencesGL45.v17.*
GTE/Tools/FiniteDifferences/FiniteDifferencesDX11.v16.vcxproj
GTE/Tools/FiniteDifferences/FiniteDifferencesGL45.v16.vcxproj
GTE/Tools/GenerateApproximations/GenerateApproximations.v17.*
GTE/Tools/GenerateApproximations/GenerateApproximations.v16.vcxproj
GTE/Tools/GenerateOpenGLWrapper/GenerateOpenGLWrapper.v17.*
GTE/Tools/GenerateOpenGLWrapper/GenerateOpenGLWrapper.v16.vcxproj
GTE/Tools/GenerateOpenGLWrapper/GenerateOpenGLWrapper.cpp
GTE/Tools/PrecisionCalculator/PrecisionCalculator.v17.*
GTE/Tools/PrecisionCalculator/PrecisionCalculator.v16.vcxproj
GTE/Tools/RotationApproximation/RotationApproximationDX11.v17.*
GTE/Tools/RotationApproximation/RotationApproximationGL45.v17.*
GTE/Tools/RotationApproximation/RotationApproximationDX11.v16.vcxproj
GTE/Tools/RotationApproximation/RotationApproximationGL45.v16.vcxproj

```

The [GenerateProject](#) tool now creates project, solutions, filter files and source files only for MSVS 2019 and MSVS 2022.

```

GTE/Tools/GenerateProject/GenerateProject.v17.*
GTE/Tools/GenerateProject/GenerateProject.v16.{vcxproj,vcxproj.filters}
GTE/Tools/GenerateProject/GenerateProject.cpp
GTE/Tools/GenerateProject/ProjectTemplate.{h,cpp}
GTE/Tools/GenerateProject/ProjectTemplate.v17.{h,cpp}
GTE/Tools/GenerateProject/ProjectTemplate.v16.{h,cpp}
GTE/Tools/GenerateProject/ProjectTemplate.v15.{h,cpp}
GTE/Tools/GenerateProject/ProjectTemplate.v14.{h,cpp}

```

7 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.