# GeometronLib

1.00

Generated by Doxygen 1.8.11

# Contents


**1 GeometronLib 1.00 Alpha Documentation** **1**

**2 Todo List** **3**

**3 Namespace Index** **5**

3.1 Namespace List . . . . . 5

**4 Hierarchical Index** **7**

4.1 Class Hierarchy . . . . . 7

**5 Class Index** **9**

5.1 Class List . . . . . 9

**6 Namespace Documentation** **13**

6.1 Gm::MeshGenerator Namespace Reference . . . . . 13

6.1.1 Detailed Description . . . . . 15

6.1.2 Typedef Documentation . . . . . 15

6.1.2.1 CurveFunction . . . . . 15

6.1.2.2 VertexModifier . . . . . 15

6.2 Gm::MeshModifier Namespace Reference . . . . . 15

6.2.1 Detailed Description . . . . . 16

6.2.2 Function Documentation . . . . . 16

6.2.2.1 ClipMesh(const TriangleMesh &mesh, const Plane &clipPlane, TriangleMesh &front, TriangleMesh &back) . . . . . 16

6.2.2.2 GetDefaultVertexDesc() . . . . . 16

6.2.2.3 InterpolateBarycentric(const VertexDescriptor &vertexDesc, void *outputVertex↩ Buffer, const void *inputVertexBuffer, std::size_t v0, std::size_t v1, std::size_t v2, const Gs::Vector3 &barycentricCoords) . . . . . 16

**7 Class Documentation** **19**

7.1 Gm::AABB< Vec, T > Class Template Reference . . . 19

7.1.1 Detailed Description . . . 20

7.1.2 Constructor & Destructor Documentation . . . 20

7.1.2.1 AABB() . . . 20

7.2 Gm::BezierCurve< P, T > Class Template Reference . . . 20

7.2.1 Detailed Description . . . 20

7.3 Gm::BezierPatch< P, T > Class Template Reference . . . 21

7.3.1 Detailed Description . . . 21

7.3.2 Member Function Documentation . . . 21

7.3.2.1 Evaluate(const T &u, const T &v) const . . . 21

7.3.2.2 GetControlPoint(unsigned int u, unsigned int v) const . . . 22

7.3.2.3 SetControlPoint(unsigned int u, unsigned int v, const P &point) . . . 22

7.4 Gm::MeshGenerator::BezierPatchDescriptor Struct Reference . . . 22

7.4.1 Detailed Description . . . 23

7.4.2 Member Data Documentation . . . 23

7.4.2.1 segments . . . 23

7.5 Gm::BezierTriangle< P, T > Class Template Reference . . . 23

7.5.1 Detailed Description . . . 23

7.5.2 Member Function Documentation . . . 24

7.5.2.1 GetControlPoint(std::size_t i, std::size_t j) const . . . 24

7.5.2.2 SetControlPoint(std::size_t i, std::size_t j, const P &point) . . . 24

7.6 Gm::MeshGenerator::CapsuleDescriptor Struct Reference . . . 24

7.6.1 Detailed Description . . . 25

7.6.2 Member Data Documentation . . . 25

7.6.2.1 ellipsoidSegments . . . 25

7.7 Gm::ClippedPolygon< T > Struct Template Reference . . . 25

7.8 Gm::MeshGenerator::ConeDescriptor Struct Reference . . . 26

7.8.1 Detailed Description . . . 26

7.9 Gm::Spline< P, T >::ControlPoint Struct Reference . . . 26

7.10 Gm::ConvexHullT< T, PlaneEq > Class Template Reference . . . 26
7.10.1 Detailed Description . . . 27
7.10.2 Member Function Documentation . . . 27
7.10.2.1 Normalize() . . . 27
7.10.3 Member Data Documentation . . . 27
7.10.3.1 planes . . . 27
7.11 Gm::MeshGenerator::CuboidDescriptor Struct Reference . . . 27
7.11.1 Detailed Description . . . 28
7.12 Gm::MeshGenerator::CurveDescriptor Struct Reference . . . 28
7.12.1 Detailed Description . . . 28
7.12.2 Member Data Documentation . . . 29
7.12.2.1 segments . . . 29
7.13 Gm::MeshGenerator::CylinderDescriptor Struct Reference . . . 29
7.13.1 Detailed Description . . . 29
7.14 Gm::MeshGenerator::EllipsoidDescriptor Struct Reference . . . 30
7.14.1 Detailed Description . . . 30
7.14.2 Member Data Documentation . . . 30
7.14.2.1 segments . . . 30
7.15 Gm::Playback::EventListener Class Reference . . . 30
7.15.1 Detailed Description . . . 31
7.15.2 Member Function Documentation . . . 31
7.15.2.1 OnNextFrame(Playback &sender) . . . 31
7.15.2.2 OnPause(Playback &sender) . . . 31
7.15.2.3 OnPlay(Playback &sender) . . . 31
7.15.2.4 OnStop(Playback &sender) . . . 32
7.16 Gm::FrustumT< T, PlaneEq > Class Template Reference . . . 32
7.16.1 Detailed Description . . . 32
7.17 Gm::Keyframe< T > Struct Template Reference . . . 32
7.17.1 Detailed Description . . . 33
7.18 Gm::KeyframeSequence Class Reference . . . 33

7.18.1 Detailed Description . . . 34
7.18.2 Member Function Documentation . . . 34
7.18.2.1 BuildKeys(std::vector< PositionKeyframe > positionKeyframes, std::vector< RotationKeyframe > rotationKeyframes, std::vector< ScaleKeyframe > scale↩ Keyframes) . . . 34
7.18.2.2 Interpolate(Gs::Vector3 &position, Gs::Quaternion &rotation, Gs::Vector3 &scale, std::size_t from, std::size_t to, Gs::Real interpolator) . . . 34
7.18.2.3 Interpolate(Gs::AffineMatrix4 &matrix, std::size_t from, std::size_t to, Gs::Real interpolator) . . . 34
7.18.2.4 Interpolate(Gs::AffineMatrix4 &matrix, const Playback &playback) . . . 35
7.19 Gm::Line< T > Class Template Reference . . . 35
7.19.1 Detailed Description . . . 35
7.20 Gm::Line< Gs::Vector2T< T > > Class Template Reference . . . 35
7.20.1 Detailed Description . . . 36
7.21 Gm::Line< Gs::Vector3T< T > > Class Template Reference . . . 36
7.21.1 Detailed Description . . . 37
7.22 Gm::Playback::ListLoop Class Reference . . . 37
7.22.1 Detailed Description . . . 37
7.22.2 Member Function Documentation . . . 38
7.22.2.1 OnNextFrame(Playback &sender) override . . . 38
7.22.2.2 OnStop(Playback &sender) override . . . 38
7.23 Gm::Playback::Loop Class Reference . . . 38
7.23.1 Detailed Description . . . 39
7.23.2 Member Function Documentation . . . 39
7.23.2.1 OnNextFrame(Playback &sender) override . . . 39
7.24 Gm::OBB< Vec, T > Class Template Reference . . . 39
7.24.1 Detailed Description . . . 40
7.25 Gm::Playback::OneShot Class Reference . . . 40
7.25.1 Detailed Description . . . 40
7.25.2 Member Function Documentation . . . 41
7.25.2.1 OnNextFrame(Playback &sender) override . . . 41
7.26 Gm::Playback::PingPongLoop Class Reference . . . 41

7.26.1 Detailed Description . . . 41
7.26.2 Member Function Documentation . . . 42
7.26.2.1 OnNextFrame(Playback &sender) override . . . 42
7.27 Gm::MeshGenerator::PipeDescriptor Struct Reference . . . 42
7.27.1 Detailed Description . . . 43
7.28 Gm::PlaneEquation_NX_eq_D< T > Struct Template Reference . . . 43
7.29 Gm::PlaneEquation_NXD_eq_Zero< T > Struct Template Reference . . . 43
7.30 Gm::PlaneT< T, PlaneEq > Class Template Reference . . . 43
7.30.1 Detailed Description . . . 44
7.30.2 Constructor & Destructor Documentation . . . 45
7.30.2.1 PlaneT(const T &x, const T &y, const T &z, const T &d) . . . 45
7.30.3 Member Function Documentation . . . 45
7.30.3.1 MemberPoint() const . . . 45
7.31 Gm::Playback Class Reference . . . 45
7.31.1 Detailed Description . . . 47
7.31.2 Member Enumeration Documentation . . . 47
7.31.2.1 State . . . 47
7.31.3 Member Function Documentation . . . 47
7.31.3.1 AreFramesChrono() const . . . 47
7.31.3.2 HasEndReached() const . . . 47
7.31.3.3 IsForward() const . . . 48
7.31.3.4 Pause(bool paused=true) . . . 48
7.31.3.5 Play(FrameIndex firstFrameIndex, FrameIndex lastFrameIndex, Gs::Real playbackSpeed, const std::shared_ptr< EventListener > &eventListener=nullptr) 48
7.31.3.6 Play(FrameIndex firstFrameIndex, FrameIndex lastFrameIndex, const std↩ ::shared_ptr< EventListener > &eventListener=nullptr) . . . 49
7.31.3.7 Play(const std::shared_ptr< EventListener > &eventListener=nullptr) . . . 49
7.31.3.8 SetNextFrame(FrameIndex nextFrameIndex) . . . 49
7.31.3.9 SetNextFrame() . . . 49
7.31.3.10 Stop() . . . 50
7.31.3.11 Update(Gs::Real deltaTime) . . . 50

7.32 Gm::UniformSpline< P, T >::Polynomial Struct Reference . . . . . 50
7.32.1 Detailed Description . . . . . 51
7.33 Gm::ProjectionT< T > Class Template Reference . . . . . 51
7.33.1 Detailed Description . . . . . 52
7.33.2 Member Function Documentation . . . . . 52
7.33.2.1 GetFlags() const . . . . . 52
7.33.2.2 GetMatrix(MatrixType &matrix, int flags) const . . . . . 52
7.33.2.3 SetFlags(int flags) . . . . . 53
7.34 Gm::Ray< T > Class Template Reference . . . . . 53
7.34.1 Detailed Description . . . . . 53
7.35 Gm::Skeleton Class Reference . . . . . 54
7.35.1 Member Function Documentation . . . . . 54
7.35.1.1 AddRootJoint(SkeletonJointPtr &&joint) . . . . . 54
7.35.1.2 BuildJointSpace() . . . . . 54
7.35.1.3 CopyFrom(const Skeleton &skeletonModel, MakeSkeletonJointFunction make↩ SkeletonJoint=nullptr) . . . . . 55
7.35.1.4 FillGlobalTransformBuffer(float *buffer, std::size_t bufferSize, bool relative↩ Transform=true) const . . . . . 55
7.35.1.5 FillLocalTransformBuffer(float *buffer, std::size_t bufferSize) const . . . . . 56
7.35.1.6 RebuildPoseTransforms() . . . . . 56
7.35.1.7 RemoveRootJoint(SkeletonJoint &joint) . . . . . 56
7.36 Gm::SkeletonJoint Class Reference . . . . . 57
7.36.1 Member Function Documentation . . . . . 58
7.36.1.1 AddSubJoint(SkeletonJointPtr &&joint) . . . . . 58
7.36.1.2 GlobalTransform(TransformMatrix &matrix) const . . . . . 58
7.36.1.3 GlobalTransform() const . . . . . 58
7.36.1.4 RemoveSubJoint(SkeletonJoint &joint) . . . . . 58
7.36.2 Member Data Documentation . . . . . 59
7.36.2.1 jointSpaceTransform . . . . . 59
7.36.2.2 poseTransform . . . . . 59
7.36.2.3 transform . . . . . 59

7.37 Gm::SphereT< T > Class Template Reference . . . 59
7.37.1 Detailed Description . . . 60
7.38 Gm::MeshGenerator::SpiralDescriptor Struct Reference . . . 60
7.38.1 Detailed Description . . . 61
7.38.2 Member Data Documentation . . . 61
7.38.2.1 mantleSegments . . . 61
7.39 Gm::Spline< P, T > Class Template Reference . . . 61
7.39.1 Detailed Description . . . 61
7.39.2 Member Function Documentation . . . 62
7.39.2.1 AddPoint(const P &point, const T &t) . . . 62
7.40 Gm::MeshGenerator::TorusDescriptor Struct Reference . . . 62
7.40.1 Detailed Description . . . 62
7.40.2 Member Data Documentation . . . 63
7.40.2.1 segments . . . 63
7.41 Gm::MeshGenerator::TorusKnotDescriptor Struct Reference . . . 63
7.41.1 Detailed Description . . . 63
7.41.2 Member Data Documentation . . . 63
7.41.2.1 loops . . . 63
7.41.2.2 segments . . . 64
7.41.2.3 turns . . . 64
7.42 Gm::Transform2T< T > Class Template Reference . . . 64
7.42.1 Detailed Description . . . 65
7.43 Gm::Transform3T< T > Class Template Reference . . . 65
7.43.1 Detailed Description . . . 66
7.44 Gm::Triangle< T > Class Template Reference . . . 66
7.44.1 Detailed Description . . . 66
7.45 Gm::Triangle< Gs::Vector2T< T > > Class Template Reference . . . 67
7.45.1 Detailed Description . . . 67
7.45.2 Member Function Documentation . . . 67
7.45.2.1 BarycentricToCartesian(const Gs::Vector3T< T > &barycentricCoord) const . . . 67

7.45.2.2 BarycentricToCartesian(const Triangle< Gs::Vector3T< T > > &barycentric↩ Triangle) const . . . 68
7.45.2.3 CartesianToBarycentric(const Gs::Vector2T< T > &cartesianCoord) const . . . 68
7.45.2.4 Normal() const . . . 68
7.46 Gm::Triangle< Gs::Vector3T< T > > Class Template Reference . . . 69
7.46.1 Detailed Description . . . 69
7.46.2 Member Function Documentation . . . 69
7.46.2.1 BarycentricToCartesian(const Gs::Vector3T< T > &barycentricCoord) const . . . 69
7.46.2.2 BarycentricToCartesian(const Triangle< Gs::Vector3T< T > > &barycentric↩ Triangle) const . . . 70
7.46.2.3 CartesianToBarycentric(const Gs::Vector3T< T > &cartesianCoord) const . . . 70
7.46.2.4 Normal() const . . . 70
7.47 Gm::TriangleMesh Class Reference . . . 71
7.47.1 Detailed Description . . . 72
7.47.2 Member Function Documentation . . . 72
7.47.2.1 SilhouetteEdges(Gs::Real toleranceAngle=Gs::Real(0)) const . . . 72
7.47.2.2 TriangleNeighbors(std::set< TriangleIndex > triangleIndices, std::size_t search↩ Depth=1, bool edgeBondOnly=false, bool searchViaPosition=false) const . . . 72
7.48 Gm::UniformSpline< P, T > Class Template Reference . . . 73
7.48.1 Detailed Description . . . 73
7.48.2 Member Function Documentation . . . 73
7.48.2.1 Build(const std::vector< P > &points, const T &expansion=T(1)) . . . 73
7.49 Gm::TriangleMesh::Vertex Struct Reference . . . 74
7.49.1 Detailed Description . . . 74
7.50 Gm::MeshModifier::VertexAttributeDescriptor Struct Reference . . . 74
7.50.1 Detailed Description . . . 75
7.51 Gm::MeshModifier::VertexDescriptor Struct Reference . . . 75
7.51.1 Detailed Description . . . 75
7.51.2 Member Data Documentation . . . 76
7.51.2.1 stride . . . 76
7.52 Gm::SkeletonJoint::VertexWeight Struct Reference . . . 76
7.52.1 Detailed Description . . . 76

**Index** **77**

# Chapter 1

# GeometronLib 1.00 Alpha Documentation

The GeometronLib provides basic functionality for 2D and 3D geometrical objects, such as mesh generation, basic collision detection, and respective data structures for lines, rays, spheres etc.

Prerequisites:

- `GaussianLib` header files

Features:

- **AABB** (Axis-Aligned Bounding-Box)
- **OBB** (Oriented Bounding-Box)
- **Line**
- **Ray**
- **Transform2** (3x3 Matrix Manager for 2D Transformations)
- **Transform3** (4x4 Matrix Manager for 3D Transformations)
- **Frustum** (Frustum of Pyramid)
- **Projection** (4x4 Projection Matrix Manager)
- **Sphere**
- **Spline**
- **TriangleMesh**
- **MeshGenerator**
- **BezierCurve**
- **BezierTriangle**

# Chapter 2

# Todo List

**Class Gm::BezierTriangle< P, T >**

This is incomplete!!!

**Class Gm::Spline< P, T >**

!!!OPTIMIZE THIS!!!

# Chapter 3

# Namespace Index

## 3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Gm::MeshGenerator
Namespace with all mesh generation functions . . . . . 13
Gm::MeshModifier
Namespace with all mesh modifier functions . . . . . 15

# Chapter 4

# Hierarchical Index

## 4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:


Gm::AABB< Vec, T > . . . 19
Gm::BezierCurve< P, T > . . . 20
Gm::BezierPatch< P, T > . . . 21
Gm::BezierPatch< Gs::Real > . . . 21
Gm::MeshGenerator::BezierPatchDescriptor . . . 22
Gm::BezierTriangle< P, T > . . . 23
Gm::MeshGenerator::CapsuleDescriptor . . . 24
Gm::ClippedPolygon< T > . . . 25
Gm::MeshGenerator::ConeDescriptor . . . 26
Gm::Spline< P, T >::ControlPoint . . . 26
Gm::ConvexHullT< T, PlaneEq > . . . 26
    Gm::FrustumT< T, PlaneEq > . . . 32
Gm::MeshGenerator::CuboidDescriptor . . . 27
Gm::MeshGenerator::CurveDescriptor . . . 28
Gm::MeshGenerator::CylinderDescriptor . . . 29
Gm::MeshGenerator::EllipsoidDescriptor . . . 30
Gm::Playback::EventListener . . . 30
    Gm::Playback::ListLoop . . . 37
    Gm::Playback::Loop . . . 38
    Gm::Playback::OneShot . . . 40
    Gm::Playback::PingPongLoop . . . 41
Gm::Keyframe< T > . . . 32
Gm::KeyframeSequence . . . 33
Gm::Line< T > . . . 35
Gm::Line< Gs::Vector2T< T > > . . . 35
Gm::Line< Gs::Vector3T< T > > . . . 36
Gm::OBB< Vec, T > . . . 39
Gm::MeshGenerator::PipeDescriptor . . . 42
Gm::PlaneEquation_NX_eq_D< T > . . . 43
Gm::PlaneEquation_NXD_eq_Zero< T > . . . 43
Gm::PlaneT< T, PlaneEq > . . . 43
Gm::Playback . . . 45
Gm::UniformSpline< P, T >::Polynomial . . . 50
Gm::ProjectionT< T > . . . 51
Gm::Ray< T > . . . 53

Gm::Skeleton . . . . . 54
Gm::SkeletonJoint . . . . . 57
Gm::SphereT< T > . . . . . 59
Gm::MeshGenerator::SpiralDescriptor . . . . . 60
Gm::Spline< P, T > . . . . . 61
Gm::MeshGenerator::TorusDescriptor . . . . . 62
Gm::MeshGenerator::TorusKnotDescriptor . . . . . 63
Gm::Transform2T< T > . . . . . 64
Gm::Transform3T< T > . . . . . 65
Gm::Triangle< T > . . . . . 66
Gm::Triangle< Gs::Vector2T< T > > . . . . . 67
Gm::Triangle< Gs::Vector3T< T > > . . . . . 69
Gm::TriangleMesh . . . . . 71
Gm::UniformSpline< P, T > . . . . . 73
Gm::TriangleMesh::Vertex . . . . . 74
Gm::MeshModifier::VertexAttributeDescriptor . . . . . 74
Gm::MeshModifier::VertexDescriptor . . . . . 75
Gm::SkeletonJoint::VertexWeight . . . . . 76

# Chapter 5

# Class Index

## 5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:


Gm::AABB< Vec, T >
: Base AABB (Axis-Aligned Bounding-Box) class . . . 19
Gm::BezierCurve< P, T >
: Curve in BB-Form (Bernstein Bezier) . . . 20
Gm::BezierPatch< P, T >
: Curved patch in BB-Form (Bernstein Bezier) . . . 21
Gm::MeshGenerator::BezierPatchDescriptor
: Descriptor structure for a Bezier patch mesh . . . 22
Gm::BezierTriangle< P, T >
: Curved triangle patch in BB-Form (Bernstein Bezier) . . . 23
Gm::MeshGenerator::CapsuleDescriptor
: Descriptor structure for a capsule mesh (i.e. cylinder with a half-sphere at top and bottom) . . . 24
Gm::ClippedPolygon< T > . . . 25
Gm::MeshGenerator::ConeDescriptor
: Descriptor structure for a cone mesh . . . 26
Gm::Spline< P, T >::ControlPoint . . . 26
Gm::ConvexHullT< T, PlaneEq > . . . 26
Gm::MeshGenerator::CuboidDescriptor
: Descriptor structure for a cuboid (also cube) mesh . . . 27
Gm::MeshGenerator::CurveDescriptor
: Descriptor structure for a curve mesh (as a rope along a given curve function) . . . 28
Gm::MeshGenerator::CylinderDescriptor
: Descriptor structure for a cylinder mesh . . . 29
Gm::MeshGenerator::EllipsoidDescriptor
: Descriptor structure for an ellispoid (also sphere) mesh . . . 30
Gm::Playback::EventListener
: Playback event listener interface . . . 30
Gm::FrustumT< T, PlaneEq >
: Base frustum class . . . 32
Gm::Keyframe< T >
: Keyframe template structure . . . 32
Gm::KeyframeSequence
: Animation keyframe sequence class. This class is used to build the transformations for an animation . . . 33
Gm::Line< T >
: Base line class . . . 35

Gm::Line< Gs::Vector2T< T > >
Specialized line class with 2D vectors . . . 35
Gm::Line< Gs::Vector3T< T > >
Specialized line class with 3D vectors . . . 36
Gm::Playback::ListLoop
List loop playback event listener . . . 37
Gm::Playback::Loop
Loop playback event listener . . . 38
Gm::OBB< Vec, T >
Base OBB (Oriented Bounding-Box) class . . . 39
Gm::Playback::OneShot
One shot playback event listener . . . 40
Gm::Playback::PingPongLoop
Ping-pong loop playback event listener . . . 41
Gm::MeshGenerator::PipeDescriptor
Descriptor structure for a pipe mesh (i.e. cylinder with a hole) . . . 42
Gm::PlaneEquation_NX_eq_D< T > . . . 43
Gm::PlaneEquation_NXD_eq_Zero< T > . . . 43
Gm::PlaneT< T, PlaneEq >
Plane base class with components: 'normal' and 'distance' . . . 43
Gm::Playback
Animation playback class . . . 45
Gm::UniformSpline< P, T >::Polynomial
Polynomial structure with four coefficients . . . 50
Gm::ProjectionT< T >
Projection class . . . 51
Gm::Ray< T >
Ray base class. It's direction must always be normalized! . . . 53
Gm::Skeleton . . . 54
Gm::SkeletonJoint . . . 57
Gm::SphereT< T >
Base sphere class . . . 59
Gm::MeshGenerator::SpiralDescriptor
Descriptor structure for a spiral mesh . . . 60
Gm::Spline< P, T >
Spline base class . . . 61
Gm::MeshGenerator::TorusDescriptor
Descriptor structure for a torus mesh . . . 62
Gm::MeshGenerator::TorusKnotDescriptor
Descriptor structure for a torus-knot mesh (uses the curve generator) . . . 63
Gm::Transform2T< T >
2D transformation class . . . 64
Gm::Transform3T< T >
3D transformation class . . . 65
Gm::Triangle< T >
Triangle base class . . . 66
Gm::Triangle< Gs::Vector2T< T > >
Template specializationn for 2D triangles . . . 67
Gm::Triangle< Gs::Vector3T< T > >
Template specializationn for 3D triangles . . . 69
Gm::TriangleMesh
Triangle mesh base class . . . 71
Gm::UniformSpline< P, T >
Spline class with uniform weights . . . 73
Gm::TriangleMesh::Vertex
Base vertex structure. Contains the members: position, normal, and texCoord . . . 74
Gm::MeshModifier::VertexAttributeDescriptor
Vertex attribute descriptor structure . . . 74

Gm::MeshModifier::VertexDescriptor
Vertex descriptor structure . . . . . 75
Gm::SkeletonJoint::VertexWeight
Vertex-joint weight structure . . . . . 76

# Chapter 6

# Namespace Documentation

## 6.1 Gm::MeshGenerator Namespace Reference

Namespace with all mesh generation functions.

### Classes

- struct BezierPatchDescriptor

  *Descriptor structure for a Bezier patch mesh.*
- struct CapsuleDescriptor

  *Descriptor structure for a capsule mesh (i.e. cylinder with a half-sphere at top and bottom).*
- struct ConeDescriptor

  *Descriptor structure for a cone mesh.*
- struct CuboidDescriptor

  *Descriptor structure for a cuboid (also cube) mesh.*
- struct CurveDescriptor

  *Descriptor structure for a curve mesh (as a rope along a given curve function).*
- struct CylinderDescriptor

  *Descriptor structure for a cylinder mesh.*
- struct EllipsoidDescriptor

  *Descriptor structure for an ellispoid (also sphere) mesh.*
- struct PipeDescriptor

  *Descriptor structure for a pipe mesh (i.e. cylinder with a hole).*
- struct SpiralDescriptor

  *Descriptor structure for a spiral mesh.*
- struct TorusDescriptor

  *Descriptor structure for a torus mesh.*
- struct TorusKnotDescriptor

  *Descriptor structure for a torus-knot mesh (uses the curve generator).*

### Typedefs

- using VertexModifier = std::function< Gs::Real(Gs::Real u, Gs::Real v)>

  *Vertex modifier function interface.*
- using CurveFunction = std::function< Gs::Vector3(Gs::Real t)>

  *Function interface for an arbitrary R -> R^3 transformation.*

## Functions

- void GenerateCuboid (const CuboidDescriptor &desc, TriangleMesh &mesh)

  *Generates a cuboid (also cube) mesh with the specified descriptor and appends the result to the specified output mesh.*
- TriangleMesh GenerateCuboid (const CuboidDescriptor &desc)

  *Generates and returns a new cuboid (also cube) mesh with the specified descriptor.*
- void GenerateEllipsoid (const EllipsoidDescriptor &desc, TriangleMesh &mesh)

  *Generates an ellipsoid (also sphere) mesh with the specified descriptor and appends the result to the specified output mesh.*
- TriangleMesh GenerateEllipsoid (const EllipsoidDescriptor &desc)

  *Generates and returns a new ellipsoid (also sphere) mesh with the specified descriptor.*
- void GenerateCone (const ConeDescriptor &desc, TriangleMesh &mesh)

  *Generates a cone mesh with the specified descriptor and appends the result to the specified output mesh.*
- TriangleMesh GenerateCone (const ConeDescriptor &desc)

  *Generates and returns a new cone mesh with the specified descriptor.*
- void GenerateCylinder (const CylinderDescriptor &desc, TriangleMesh &mesh)

  *Generates a cylinder mesh with the specified descriptor and appends the result to the specified output mesh.*
- TriangleMesh GenerateCylinder (const CylinderDescriptor &desc)

  *Generates and returns a new cylinder mesh with the specified descriptor.*
- void GeneratePipe (const PipeDescriptor &desc, TriangleMesh &mesh)

  *Generates a pipe mesh (i.e. cylinder with a hole) with the specified descriptor and appends the result to the specified output mesh.*
- TriangleMesh GeneratePipe (const PipeDescriptor &desc)

  *Generates and returns a new pipe (i.e. cylinder with a hole) mesh with the specified descriptor.*
- void GenerateCapsule (const CapsuleDescriptor &desc, TriangleMesh &mesh)

  *Generates a capsule mesh (i.e. cylinder with a half-sphere at top and bottom) with the specified descriptor and appends the result to the specified output mesh.*
- TriangleMesh GenerateCapsule (const CapsuleDescriptor &desc)

  *Generates and returns a new capsule mesh (i.e. cylinder with a half-sphere at top and bottom) with the specified descriptor.*
- void GenerateTorus (const TorusDescriptor &desc, TriangleMesh &mesh)

  *Generates a torus mesh with the specified descriptor and appends the result to the specified output mesh.*
- TriangleMesh GenerateTorus (const TorusDescriptor &desc)

  *Generates and returns a new torus mesh with the specified descriptor.*
- void GenerateTorusKnot (const TorusKnotDescriptor &desc, TriangleMesh &mesh)

  *Generates a torus-knot mesh with the specified descriptor and appends the result to the specified output mesh.*
- TriangleMesh GenerateTorusKnot (const TorusKnotDescriptor &desc)

  *Generates and returns a new torus-knot mesh with the specified descriptor.*
- void GenerateSpiral (const SpiralDescriptor &desc, TriangleMesh &mesh)

  *Generates a spiral mesh with the specified descriptor and appends the result to the specified output mesh.*
- TriangleMesh GenerateSpiral (const SpiralDescriptor &desc)

  *Generates and returns a new spiral mesh with the specified descriptor.*
- void GenerateCurve (const CurveDescriptor &desc, TriangleMesh &mesh)

  *Generates a curve mesh (as a rope along a given curve function) with the specified descriptor and appends the result to the specified output mesh.*
- TriangleMesh GenerateCurve (const CurveDescriptor &desc)

  *Generates and returns a new curve mesh (as a rope along a given curve function) with the specified descriptor.*
- void GenerateBezierPatch (const BezierPatchDescriptor &desc, TriangleMesh &mesh)

  *Generates a Bezier patch mesh with the specified descriptor and appends the result to the specified output mesh.*
- TriangleMesh GenerateBezierPatch (const BezierPatchDescriptor &desc)

  *Generates and returns a new Bezier patch mesh with the specified descriptor.*

### 6.1.1 Detailed Description

Namespace with all mesh generation functions.

### 6.1.2 Typedef Documentation

#### 6.1.2.1 using Gm::MeshGenerator::CurveFunction = typedef std::function<Gs::Vector3(Gs::Real t)>

Function interface for an arbitrary R -> R^3 transformation.

**Parameters**

| | | |
|---|---|---|
| in | *t* | Specifies the curve progression. This is in the range [0, 1]. |

**Returns**

3D point which lies on the curve at the position 't'.

**See also**

CurveDescriptor

#### 6.1.2.2 using Gm::MeshGenerator::VertexModifier = typedef std::function<Gs::Real(Gs::Real u, Gs::Real v)>

Vertex modifier function interface.

**Parameters**

| | | |
|---|---|---|
| in | *u* | Specifies the interpolation factor U. This is in the range [0, 1]. |
| in | *v* | Specifies the interpolation factor V. This is in the range [0, 1]. |

**Returns**

Interpolation factor which should be in the range [0, 1].

**Remarks**

This can be used for a couple of mesh generators, to adjust the final vertex position.

**See also**

TorusKnotDescriptor

## 6.2 Gm::MeshModifier Namespace Reference

Namespace with all mesh modifier functions.

### Classes

- struct VertexAttributeDescriptor

  *Vertex attribute descriptor structure.*
- struct VertexDescriptor

  *Vertex descriptor structure.*

### Functions

- const VertexDescriptor & GetDefaultVertexDesc ()

  *Returns the vertex descriptor for the default vertex format.*
- void InterpolateBarycentric (const VertexDescriptor &vertexDesc, void *outputVertexBuffer, const void *inputVertexBuffer, std::size_t v0, std::size_t v1, std::size_t v2, const Gs::Vector3 &barycentricCoords)

  *Makes a barycentric interpolation between the three specified vertices.*
- void ClipMesh (const TriangleMesh &mesh, const Plane &clipPlane, TriangleMesh &front, TriangleMesh &back)

## 6.2.1 Detailed Description

Namespace with all mesh modifier functions.

## 6.2.2 Function Documentation

### 6.2.2.1 void Gm::MeshModifier::ClipMesh ( const TriangleMesh & *mesh*, const Plane & *clipPlane*, TriangleMesh & *front*, TriangleMesh & *back* )

Clips this triangle mesh into a front- and back sided mesh by the specified clipping plane.

**See also**

ClipTriangle

### 6.2.2.2 const VertexDescriptor& Gm::MeshModifier::GetDefaultVertexDesc ( )

Returns the vertex descriptor for the default vertex format.

**See also**

TriangleMesh::Vertex

### 6.2.2.3 void Gm::MeshModifier::InterpolateBarycentric ( const VertexDescriptor & *vertexDesc*, void * *outputVertexBuffer*, const void * *inputVertexBuffer*, std::size_t *v0*, std::size_t *v1*, std::size_t *v2*, const Gs::Vector3 & *barycentricCoords* )

Makes a barycentric interpolation between the three specified vertices.

**Parameters**

| | | |
|---|---|---|
| in | *vertexDesc* | Specifies the vertex descriptor for both output and input vertex buffers. |
| out | *outputVertexBuffer* | Specifies the output vertex buffer where the interpolated vertex is to be stored. |
| in | *inputVertexBuffer* | Specifies the input vertex buffer from where the three vertices are to be read. |
| in | *v0* | Specifies the first vertex index for the triangle to interpolate the barycentric coordinates. |
| in | *v1* | Specifies the second vertex index for the triangle to interpolate the barycentric coordinates. |
| in | *v2* | Specifies the thrid vertex index for the triangle to interpolate the barycentric coordinates. |
| in | *barycentricCoords* | Specifies the barycentric coordinates. The sum of all components must be 1. |

# Chapter 7

# Class Documentation

## 7.1 Gm::AABB< Vec, T > Class Template Reference

Base AABB (Axis-Aligned Bounding-Box) class.

`#include <AABB.h>`

### Public Types

- using **ThisType** = AABB< Vec, T >

### Public Member Functions

- AABB ()
- **AABB** (const Vec< T > &min, const Vec< T > &max)
- void Reset ()

  *Sets the minimum to the highest possible value and the maximum to the lowest possible value.*
- void Reset (const Vec< T > &point)

  *Sets the minimum and maximum to the specified point.*
- void **Insert** (const Vec< T > &point)
- void **Insert** (const ThisType &aabb)
- void **Repair** ()
- Vec< T > **Size** () const
- Vec< T > **Center** () const
- std::vector< Line< Vec< T > > > Edges () const

  *Returns the list of all edges of this AABB.*
- template<typename C >
  AABB< Vec, C > **Cast** () const

### Public Attributes

- Vec< T > **min**
- Vec< T > **max**

### 7.1.1 Detailed Description

**template<template< typename > class Vec, typename T>**
**class Gm::AABB< Vec, T >**

Base AABB (Axis-Aligned Bounding-Box) class.

### 7.1.2 Constructor & Destructor Documentation

#### 7.1.2.1 template<template< typename > class Vec, typename T> Gm::AABB< Vec, T >::AABB ( ) [inline]

Constructs a maximal invald bounding-box, i.e. min has the maximal values possible, and max has the minimal values possible.

The documentation for this class was generated from the following file:

- AABB.h

## 7.2 Gm::BezierCurve< P, T > Class Template Reference

Curve in BB-Form (Bernstein Bezier).

`#include <BezierCurve.h>`

### Public Member Functions

- P **Evaluate** (const T &t) const
- P **operator()** (const T &t) const

### Public Attributes

- std::vector< P > **controlPoints**

### 7.2.1 Detailed Description

**template<typename P, typename T>**
**class Gm::BezierCurve< P, T >**

Curve in BB-Form (Bernstein Bezier).

**Template Parameters**

| | |
|---|---|
| *P* | Specifies the type of the control points. |
| *T* | Specifies the basic data type. This should be float or double. |

The documentation for this class was generated from the following file:

- BezierCurve.h

## 7.3 Gm::BezierPatch< P, T > Class Template Reference

Curved patch in BB-Form (Bernstein Bezier).

`#include <BezierPatch.h>`

### Public Member Functions

- P **operator()** (const T &u, const T &v) const
- P Evaluate (const T &u, const T &v) const

  *Evaluates the bezier patch.*
- void SetControlPoint (unsigned int u, unsigned int v, const P &point)

  *Sets the specified control point.*
- P GetControlPoint (unsigned int u, unsigned int v) const

  *Returns the specified control point.*
- const std::vector< P > & GetControlPoints () const

  *Returns the list of all control points of this bezier patch.*
- void SetOrder (unsigned int order)

  *Sets the order of this bezier patch. By default 0.*
- unsigned int GetOrder () const

  *Returns the order of this bezier triangle.*

### 7.3.1 Detailed Description

**template<typename P, typename T>**
**class Gm::BezierPatch< P, T >**

Curved patch in BB-Form (Bernstein Bezier).

**Template Parameters**

| | |
|---|---|
| *P* | Specifies the type of the control points. |

### 7.3.2 Member Function Documentation

#### 7.3.2.1 template<typename P, typename T> P Gm::BezierPatch< P, T >::Evaluate ( const T & *u*, const T & *v* ) const [inline]

Evaluates the bezier patch.

**Parameters**

| | | |
|---|---|---|
| in | *u* | Specifies the interpolation value in U direction. This should be in the range [0, 1]. |
| in | *v* | Specifies the interpolation value in V direction. This should be in the range [0, 1]. |

#### 7.3.2.2 template<typename P, typename T> P Gm::BezierPatch< P, T >::GetControlPoint ( unsigned int *u*, unsigned int *v* ) const [inline]

Returns the specified control point.

**Parameters**

| | | |
|---|---|---|
| in | *u* | Specifies the index in U direction. Must be in the range [0, GetOrder()]. |
| in | *v* | Specifies the index in V direction. Must be in the range [0, GetOrder()]. |

#### 7.3.2.3 template<typename P, typename T> void Gm::BezierPatch< P, T >::SetControlPoint ( unsigned int *u*, unsigned int *v*, const P & *point* ) [inline]

Sets the specified control point.

**Parameters**

| | | |
|---|---|---|
| in | *u* | Specifies the index in U direction. Must be in the range [0, GetOrder()]. |
| in | *v* | Specifies the index in V direction. Must be in the range [0, GetOrder()]. |
| in | *point* | Specifies the new control point. |

The documentation for this class was generated from the following file:

- BezierPatch.h

## 7.4 Gm::MeshGenerator::BezierPatchDescriptor Struct Reference

Descriptor structure for a Bezier patch mesh.

`#include <MeshGenerator.h>`

### Public Attributes

- BezierPatch3 bezierPatch

  *Bezier patch control points.*
- Gs::Vector2ui segments = Gs::Vector2ui(20, 20)
- bool alternateGrid = false

  *Specifies whether the face grids are to be alternating or uniform. By default false.*
- bool backFacing = false

  *Specifies whether the faces point to the back or to the front (default).*

### 7.4.1 Detailed Description

Descriptor structure for a Bezier patch mesh.

### 7.4.2 Member Data Documentation

#### 7.4.2.1 Gs::Vector2ui Gm::MeshGenerator::BezierPatchDescriptor::segments = Gs::Vector2ui(20, 20)

Segmentation in U (x component), and V (y component) direction. Each component will be clamped to [1, +inf). By default (20, 20).

The documentation for this struct was generated from the following file:

- MeshGenerator.h

## 7.5 Gm::BezierTriangle< P, T > Class Template Reference

Curved triangle patch in BB-Form (Bernstein Bezier).

`#include <BezierTriangle.h>`

### Public Member Functions

- P **operator()** (const T &u, const T &v) const
- P **Evaluate** (const T &s, const T &t, const T &u) const
- void SetControlPoint (std::size_t i, std::size_t j, const P &point)

  *Sets the specified control point.*
- P GetControlPoint (std::size_t i, std::size_t j) const

  *Returns the specified control point.*
- const std::vector< P > & GetControlPoints () const

  *Returns the list of all control points of this bezier triangle.*
- void SetOrder (std::size_t order)

  *Sets the order of this bezier triangle. By default 0.*
- std::size_t GetOrder () const

  *Returns the order of this bezier triangle.*

### 7.5.1 Detailed Description

**template<typename P, typename T>**
**class Gm::BezierTriangle< P, T >**

Curved triangle patch in BB-Form (Bernstein Bezier).

**Template Parameters**

| | |
|---|---|
| *P* | Specifies the type of the control points. |

**Todo** This is incomplete!!!

### 7.5.2 Member Function Documentation

#### 7.5.2.1 template<typename P , typename T > P Gm::BezierTriangle< P, T >::GetControlPoint ( std::size_t *i*, std::size_t *j* ) const [inline]

Returns the specified control point.

**Parameters**

| | | |
|---|---|---|
| in | *i* | Specifies the first index. |
| in | *j* | Specifies the second index. |

**Remarks**

The index parameters must always satisfy the following equation: 0 <= i + j <= GetOrder()

#### 7.5.2.2 template<typename P , typename T > void Gm::BezierTriangle< P, T >::SetControlPoint ( std::size_t *i*, std::size_t *j*, const P & *point* ) [inline]

Sets the specified control point.

**Parameters**

| | | |
|---|---|---|
| in | *i* | Specifies the first index. |
| in | *j* | Specifies the second index. |
| in | *point* | Specifies the new control point. |

**Remarks**

The index parameters must always satisfy the following equation: 0 <= i + j <= GetOrder()

The documentation for this class was generated from the following file:

- BezierTriangle.h

## 7.6 Gm::MeshGenerator::CapsuleDescriptor Struct Reference

Descriptor structure for a capsule mesh (i.e. cylinder with a half-sphere at top and bottom).

`#include <MeshGenerator.h>`

### Public Attributes

- Gs::Vector3 radius = Gs::Vector3(Gs::Real(0.5))

  *Radius of the top- and bottom half-ellipsoids in X, Y, and Z direction. By default (0.5, 0.5, 0.5).*
- Gs::Real height = Gs::Real(1)

  *Capsule height (without top- and bottom half-sphere). By default 1.*
- Gs::Vector2ui mantleSegments = Gs::Vector2ui(20, 1)

  *Segmentation around the cylinder (x component), and height (y component). By default (20, 1).*
- unsigned int ellipsoidSegments = 10
- bool alternateGrid = false

  *Specifies whether the face grids are to be alternating or uniform. By default false.*

#### 7.6.1 Detailed Description

Descriptor structure for a capsule mesh (i.e. cylinder with a half-sphere at top and bottom).

#### 7.6.2 Member Data Documentation

##### 7.6.2.1 unsigned int Gm::MeshGenerator::CapsuleDescriptor::ellipsoidSegments = 10

Segmentation of the top- and bottom half-ellipsoids. Each component will be clamped to [2, +inf). By default 10.

The documentation for this struct was generated from the following file:

- MeshGenerator.h

## 7.7 Gm::ClippedPolygon< T > Struct Template Reference

### Public Member Functions

- void **AddVertex** (const Gs::Vector3T< T > &vertex)

### Public Attributes

- unsigned char count = 0

  *Number of vertices, used for this clipped triangle. This is either 3 or 4.*
- std::array< Gs::Vector3T< T >, 4 > **vertices**

The documentation for this struct was generated from the following file:

- TriangleCollision.h

## 7.8 Gm::MeshGenerator::ConeDescriptor Struct Reference

Descriptor structure for a cone mesh.

`#include <MeshGenerator.h>`

### Public Attributes

- Gs::Vector2 radius = Gs::Vector2(Gs::Real(0.5))

  *Cone radius in U (x component), and V (y component) direction. By default (0.5, 0.5).*
- Gs::Real height = Gs::Real(1)

  *Cone height. By default 1.*
- Gs::Vector2ui mantleSegments = Gs::Vector2ui(20, 1)

  *Segmentation around the cone (x component), and height (y component). By default (20, 1).*
- unsigned int coverSegments = 1

  *Segmentation of the bottom cover. If 0, no bottom cover is generated. By default 1.*
- bool alternateGrid = false

  *Specifies whether the face grids are to be alternating or uniform. By default false.*

### 7.8.1 Detailed Description

Descriptor structure for a cone mesh.

The documentation for this struct was generated from the following file:

- MeshGenerator.h

## 7.9 Gm::Spline< P, T >::ControlPoint Struct Reference

### Public Attributes

- P **point**
- T **interval**

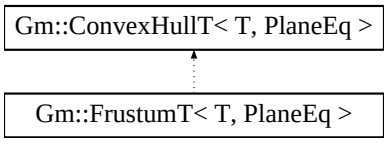
The documentation for this struct was generated from the following file:

- Spline.h

## 7.10 Gm::ConvexHullT< T, PlaneEq > Class Template Reference

`#include <ConvexHull.h>`

Inheritance diagram for Gm::ConvexHullT< T, PlaneEq >:



### Public Member Functions

- **ConvexHullT** (std::size_t planeCount)
- void Normalize ()

  *Normalizes all planes of this convex hull.*
- bool IsPointInside (const Gs::Vector3T< T > &point)

  *Returns true if the specified point is inside the convex hull.*
- bool IsSphereInside (const SphereT< T > &sphere)

  *Returns true if the specified sphere is inside the convex hull (or just intersecting one of its planes).*

### Public Attributes

- std::vector< PlaneT< T, PlaneEq > > planes

### 7.10.1 Detailed Description

**template<typename T, typename PlaneEq = DefaultPlaneEquation<T>>**
**class Gm::ConvexHullT< T, PlaneEq >**

Convex hull base class. Here a convex hull is constructed so that all plane normals point out of the hull.

### 7.10.2 Member Function Documentation

#### 7.10.2.1 template<typename T , typename PlaneEq = DefaultPlaneEquation<T>> void Gm::ConvexHullT< T, PlaneEq >::Normalize ( ) [inline]

Normalizes all planes of this convex hull.

**See also**

PlaneT::Normalize

### 7.10.3 Member Data Documentation

#### 7.10.3.1 template<typename T , typename PlaneEq = DefaultPlaneEquation<T>> std::vector< PlaneT<T, PlaneEq> > Gm::ConvexHullT< T, PlaneEq >::planes

List of all planes which form the convex hull. This must be at least 3 planes to form a valid convex hull.

The documentation for this class was generated from the following file:

- ConvexHull.h

## 7.11 Gm::MeshGenerator::CuboidDescriptor Struct Reference

Descriptor structure for a cuboid (also cube) mesh.

`#include <MeshGenerator.h>`

### Public Attributes

- Gs::Vector3 size = Gs::Vector3(Gs::Real(1.0))

  *Cuboid size. By default (1, 1, 1).*
- Gs::Vector3ui segments = Gs::Vector3ui(1, 1, 1)

  *Cuboid segmentation. Each component will be clamped to [1, +inf). By default (1, 1, 1).*
- bool alternateGrid = false

  *Specifies whether the face grids are to be alternating or uniform. By default false.*

#### 7.11.1 Detailed Description

Descriptor structure for a cuboid (also cube) mesh.

The documentation for this struct was generated from the following file:

- MeshGenerator.h

## 7.12 Gm::MeshGenerator::CurveDescriptor Struct Reference

Descriptor structure for a curve mesh (as a rope along a given curve function).

`#include <MeshGenerator.h>`

### Public Attributes

- CurveFunction curveFunction = nullptr

  *Curve progression function.*
- Gs::Real radius = Gs::Real(0.25)

  *Radius of the tube which forms the curve. By default 0.25.*
- Gs::Vector2ui segments = Gs::Vector2ui(20, 20)

  *Segmentation in U (x component), and V (y component) direction.*
- bool alternateGrid = false

  *Specifies whether the face grids are to be alternating or uniform. By default false.*
- VertexModifier vertexModifier = nullptr

  *Vertex modifier to adjust the radius during mesh generation.*

#### 7.12.1 Detailed Description

Descriptor structure for a curve mesh (as a rope along a given curve function).

### 7.12.2 Member Data Documentation

#### 7.12.2.1 Gs::Vector2ui Gm::MeshGenerator::CurveDescriptor::segments = Gs::Vector2ui(20, 20)

Segmentation in U (x component), and V (y component) direction.

**Remarks**

Each component will be clamped to [3, +inf). By default (20, 20).

The documentation for this struct was generated from the following file:

- MeshGenerator.h

## 7.13 Gm::MeshGenerator::CylinderDescriptor Struct Reference

Descriptor structure for a cylinder mesh.

`#include <MeshGenerator.h>`

### Public Attributes

- Gs::Vector2 radius = Gs::Vector2(Gs::Real(0.5))

  *Cylinder radius in U (x component), and V (y component) direction. By default (0.5, 0.5).*
- Gs::Real height = Gs::Real(1)

  *Cylinder height. By default 1.*
- Gs::Vector2ui mantleSegments = Gs::Vector2ui(20, 1)

  *Segmentation around the cylinder (x component), and height (y component). By default (20, 1).*
- unsigned int topCoverSegments = 1

  *Segmentation of the top cover. If 0, no top cover is generated. By default 1.*
- unsigned int bottomCoverSegments = 1

  *Segmentation of the bottom cover. If 0, no bottom cover is generated. By default 1.*
- bool alternateGrid = false

  *Specifies whether the face grids are to be alternating or uniform. By default false.*

### 7.13.1 Detailed Description

Descriptor structure for a cylinder mesh.

The documentation for this struct was generated from the following file:

- MeshGenerator.h

## 7.14 Gm::MeshGenerator::EllipsoidDescriptor Struct Reference

Descriptor structure for an ellispoid (also sphere) mesh.

`#include <MeshGenerator.h>`

### Public Attributes

- Gs::Vector3 radius = Gs::Vector3(Gs::Real(0.5))

  *Radius in X, Y, and Z direction. By default (0.5, 0.5, 0.5).*
- Gs::Vector2ui segments = Gs::Vector2ui(20, 10)
- bool alternateGrid = false

  *Specifies whether the face grids are to be alternating or uniform. By default false.*

### 7.14.1 Detailed Description

Descriptor structure for an ellispoid (also sphere) mesh.

### 7.14.2 Member Data Documentation

#### 7.14.2.1 Gs::Vector2ui Gm::MeshGenerator::EllipsoidDescriptor::segments = Gs::Vector2ui(20, 10)

Segmentation in U (x component), and V (y component) direction. X component will be clamped to [3, +inf), Y component will be clamped to [2, +inf). By default (20, 10).

The documentation for this struct was generated from the following file:

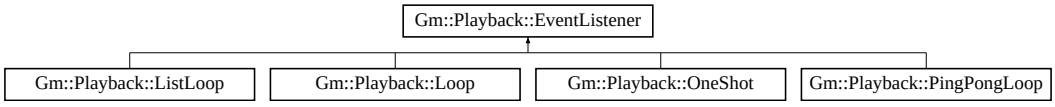
- MeshGenerator.h

## 7.15 Gm::Playback::EventListener Class Reference

Playback event listener interface.

`#include <Playback.h>`

Inheritance diagram for Gm::Playback::EventListener:



### Public Member Functions

- virtual void OnPlay (Playback &sender)

  *Receives the 'playback start' event. All playback configurations are done when this event is posted.*
- virtual void OnPause (Playback &sender)

  *Receives the 'playback paused' event. All playback configurations are done when this event is posted.*
- virtual void OnStop (Playback &sender)
- virtual void OnNextFrame (Playback &sender)

## 7.15.1 Detailed Description

Playback event listener interface.

## 7.15.2 Member Function Documentation

### 7.15.2.1 virtual void Gm::Playback::EventListener::OnNextFrame ( Playback & *sender* ) [inline], [virtual]

Receives the 'next frame' event. This event will be posted in the "Playback::Update" function.

**Remarks**

Depending on the playback speed sometimes this event will be posted several times for a single call to the "Playback::Update" function. This happens when several frames will be skipped, i.e. every single frame can be examined with this event listener interface, no matter how fast the playback speed is. All playback configurations for the next frame are done when this event is posted. This function should set the next frame (see 'nextFrame' field).

**See also**

Playback::Update
Playback::nextFrame

Reimplemented in Gm::Playback::ListLoop, Gm::Playback::PingPongLoop, Gm::Playback::Loop, and Gm::Playback::OneShot.

### 7.15.2.2 virtual void Gm::Playback::EventListener::OnPause ( Playback & *sender* ) [inline], [virtual]

Receives the 'playback paused' event. All playback configurations are done when this event is posted.

**See also**

Playback::Pause

### 7.15.2.3 virtual void Gm::Playback::EventListener::OnPlay ( Playback & *sender* ) [inline], [virtual]

Receives the 'playback start' event. All playback configurations are done when this event is posted.

**See also**

Playback::Play

#### 7.15.2.4 virtual void Gm::Playback::EventListener::OnStop ( Playback & *sender* ) [inline], [virtual]

Receives the 'playback stopped' event. All playback configurations are done when this event is posted.

**Remarks**

This will only be posted if the playback was previously being played or paused.

**See also**

Playback::Stop

Reimplemented in Gm::Playback::ListLoop.

The documentation for this class was generated from the following file:

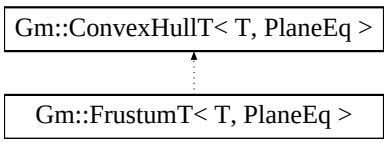
- Playback.h

## 7.16 Gm::FrustumT< T, PlaneEq > Class Template Reference

Base frustum class.

`#include <Frustum.h>`

Inheritance diagram for Gm::FrustumT< T, PlaneEq >:




### 7.16.1 Detailed Description

**template<typename T, typename PlaneEq = DefaultPlaneEquation<T>>**
**class Gm::FrustumT< T, PlaneEq >**

Base frustum class.

The documentation for this class was generated from the following file:

- Frustum.h

## 7.17 Gm::Keyframe< T > Struct Template Reference

Keyframe template structure.

`#include <KeyframeSequence.h>`

### Public Member Functions

- **Keyframe** (const Keyframe< T > &)=default
- **Keyframe** (const T &key, std::size_t frame)

### Public Attributes

- T key

  *Keyframe value (commonly Vector3 for position and scale, or Quaternion for rotation).*
- std::size_t frame = 0

  *Keyframe index.*

#### 7.17.1 Detailed Description

**template<typename T>**
**struct Gm::Keyframe< T >**

Keyframe template structure.

The documentation for this struct was generated from the following file:

- KeyframeSequence.h

## 7.18 Gm::KeyframeSequence Class Reference

Animation keyframe sequence class. This class is used to build the transformations for an animation.

`#include <KeyframeSequence.h>`

### Public Member Functions

- void **ClearKeys** ()
- void BuildKeys (std::vector< PositionKeyframe > positionKeyframes, std::vector< RotationKeyframe > rotationKeyframes, std::vector< ScaleKeyframe > scaleKeyframes)

  *Builds the interpolated keys by the specified keyframes.*
- void Interpolate (Gs::Vector3 &position, Gs::Quaternion &rotation, Gs::Vector3 &scale, std::size_t from, std::size_t to, Gs::Real interpolator)

  *Interpolates the specified keyframes and writes the result into the respective output parameter.*
- void Interpolate (Gs::AffineMatrix4 &matrix, std::size_t from, std::size_t to, Gs::Real interpolator)

  *Interpolates the specified keyframes and writes the result into the output matrix.*
- void Interpolate (Gs::AffineMatrix4 &matrix, const Playback &playback)

  *Interpolates the keyframes, specified by the playback state, and writes the result into the output matrix.*
- const std::vector< Gs::Vector3 > & GetPositionKeys () const

  *Returns the pre-computed position keys.*
- const std::vector< Gs::Quaternion > & GetRotationKeys () const

  *Returns the pre-computed rotation keys.*
- const std::vector< Gs::Vector3 > & GetScaleKeys () const

  *Returns the pre-computed scale keys.*
- std::size_t GetFrameBegin () const

  *Returns the frame end in the (half-open) range [GetFrameBegin(), GetFrameEnd()).*
- std::size_t GetFrameEnd () const

  *Returns the frame end in the (half-open) range [GetFrameBegin(), GetFrameEnd()).*

### 7.18.1 Detailed Description

Animation keyframe sequence class. This class is used to build the transformations for an animation.

### 7.18.2 Member Function Documentation

#### 7.18.2.1 void Gm::KeyframeSequence::BuildKeys ( std::vector< PositionKeyframe > *positionKeyframes,* std::vector< RotationKeyframe > *rotationKeyframes,* std::vector< ScaleKeyframe > *scaleKeyframes* )

Builds the interpolated keys by the specified keyframes.

**Parameters**

| | | |
|---|---|---|
| in | *positionKeyframes* | Specifies the position keyframes. |

#### 7.18.2.2 void Gm::KeyframeSequence::Interpolate ( Gs::Vector3 & *position,* Gs::Quaternion & *rotation,* Gs::Vector3 & *scale,* std::size_t *from,* std::size_t *to,* Gs::Real *interpolator* )

Interpolates the specified keyframes and writes the result into the respective output parameter.

**Parameters**

| | | |
|---|---|---|
| out | *position* | Specifies the interpolated output position. |
| out | *rotation* | Specifies the interpolated output rotation. |
| out | *scale* | Specifies the interpolated output scale. |
| in | *from* | Specifies the keyframe index from which to interpolate. This will be clamped to the range [GetFrameBegin(), GetFrameEnd()). |
| in | *to* | Specifies the keyframe index to which to interpolate. This will be clamped to the range [GetFrameBegin(), GetFrameEnd()). |
| in | *interpolator* | Specifies the interpolation factor in the range [0, 1]. |

**Remarks**

If this keyframe sequences has no keys, this function call has no effect. To build the keys, call "BuildKeys".

**See also**

GetFrameBegin
GetFrameEnd
BuildKeys

#### 7.18.2.3 void Gm::KeyframeSequence::Interpolate ( Gs::AffineMatrix4 & *matrix,* std::size_t *from,* std::size_t *to,* Gs::Real *interpolator* )

Interpolates the specified keyframes and writes the result into the output matrix.

**See also**

Interpolate(Gs::Vector3&, Gs::Quaternion&, Gs::Vector3&, std::size_t, std::size_t, Gs::Real)

#### 7.18.2.4 void Gm::KeyframeSequence::Interpolate ( Gs::AffineMatrix4 & *matrix,* const Playback & *playback* )

Interpolates the keyframes, specified by the playback state, and writes the result into the output matrix.

**See also**

Interpolate(Gs::AffineMatrix4&, std::size_t, std::size_t, Gs::Real)

The documentation for this class was generated from the following file:

- KeyframeSequence.h

## 7.19 Gm::Line< T > Class Template Reference

Base line class.

`#include <Line.h>`

### Public Member Functions

- **Line** (const Line< T > &)=default
- **Line** (const T &a, const T &b)
- **Line** (Gs::UninitializeTag)

### Public Attributes

- T **a**
- T **b**

### 7.19.1 Detailed Description

**template<typename T>**
**class Gm::Line< T >**

Base line class.

The documentation for this class was generated from the following file:

- Line.h

## 7.20 Gm::Line< Gs::Vector2T< T > > Class Template Reference

Specialized line class with 2D vectors.

`#include <Line.h>`

### Public Member Functions

- **Line** (const Line< Gs::Vector2T< T > > &)=default
- **Line** (const Gs::Vector2T< T > &a, const Gs::Vector2T< T > &b)
- **Line** (Gs::UninitializeTag)
- Gs::Vector2T< T > **Direction** () const
- Gs::Vector2T< T > **Lerp** (const T &t) const
- T **LengthSq** () const
- T **Length** () const
- Gs::Vector2T< T > **operator()** (const T &t) const

### Public Attributes

- Gs::Vector2T< T > **a**
- Gs::Vector2T< T > **b**

#### 7.20.1 Detailed Description

**template<typename T>**
**class Gm::Line< Gs::Vector2T< T > >**

Specialized line class with 2D vectors.

The documentation for this class was generated from the following file:

- Line.h

## 7.21 Gm::Line< Gs::Vector3T< T > > Class Template Reference

Specialized line class with 3D vectors.

`#include <Line.h>`

### Public Member Functions

- **Line** (const Line< Gs::Vector3T< T > > &)=default
- **Line** (const Gs::Vector3T< T > &a, const Gs::Vector3T< T > &b)
- **Line** (Gs::UninitializeTag)
- Gs::Vector3T< T > **Direction** () const
- Gs::Vector3T< T > **Lerp** (const T &t) const
- T **LengthSq** () const
- T **Length** () const
- Gs::Vector3T< T > **operator()** (const T &t) const

### Public Attributes

- Gs::Vector3T< T > **a**
- Gs::Vector3T< T > **b**

### 7.21.1 Detailed Description

**template<typename T>**
**class Gm::Line< Gs::Vector3T< T > >**

Specialized line class with 3D vectors.

The documentation for this class was generated from the following file:

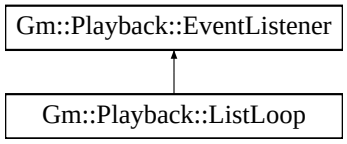
- Line.h

## 7.22 Gm::Playback::ListLoop Class Reference

List loop playback event listener.

`#include <Playback.h>`

Inheritance diagram for Gm::Playback::ListLoop:




### Public Member Functions

- void OnStop (Playback &sender) override
- void OnNextFrame (Playback &sender) override

### Public Attributes

- std::vector< FrameIndex > frames

  *Animation frame indices.*
- std::vector< FrameIndex >::size_type iterator = 0

  *Iteration index for the animation frames.*

### 7.22.1 Detailed Description

List loop playback event listener.

**Remarks**

Plays an animation by the listed frame indices in a loop fashion.

### 7.22.2 Member Function Documentation

#### 7.22.2.1 void Gm::Playback::ListLoop::OnNextFrame ( Playback & *sender* ) [override],[virtual]

Receives the 'next frame' event. This event will be posted in the "Playback::Update" function.

**Remarks**

Depending on the playback speed sometimes this event will be posted several times for a single call to the "Playback::Update" function. This happens when several frames will be skipped, i.e. every single frame can be examined with this event listener interface, no matter how fast the playback speed is. All playback configurations for the next frame are done when this event is posted. This function should set the next frame (see 'nextFrame' field).

**See also**

Playback::Update
Playback::nextFrame

Reimplemented from Gm::Playback::EventListener.

#### 7.22.2.2 void Gm::Playback::ListLoop::OnStop ( Playback & *sender* ) [override],[virtual]

Receives the 'playback stopped' event. All playback configurations are done when this event is posted.

**Remarks**

This will only be posted if the playback was previously being played or paused.

**See also**

Playback::Stop

Reimplemented from Gm::Playback::EventListener.

The documentation for this class was generated from the following file:

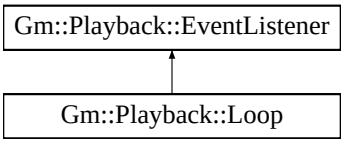
- Playback.h

## 7.23 Gm::Playback::Loop Class Reference

Loop playback event listener.

`#include <Playback.h>`

Inheritance diagram for Gm::Playback::Loop:



### Public Member Functions

- void OnNextFrame (Playback &sender) override

### 7.23.1 Detailed Description

Loop playback event listener.

#### Remarks

Plays an animation from the first to the last frame, and then starts the animation from the beginning.

### 7.23.2 Member Function Documentation

#### 7.23.2.1 void Gm::Playback::Loop::OnNextFrame ( Playback & *sender* ) [override], [virtual]

Receives the 'next frame' event. This event will be posted in the "Playback::Update" function.

#### Remarks

Depending on the playback speed sometimes this event will be posted several times for a single call to the "Playback::Update" function. This happens when several frames will be skipped, i.e. every single frame can be examined with this event listener interface, no matter how fast the playback speed is. All playback configurations for the next frame are done when this event is posted. This function should set the next frame (see 'nextFrame' field).

#### See also

Playback::Update
Playback::nextFrame

Reimplemented from Gm::Playback::EventListener.

The documentation for this class was generated from the following file:

- Playback.h

## 7.24 Gm::OBB< Vec, T > Class Template Reference

Base OBB (Oriented Bounding-Box) class.

`#include <OBB.h>`

### Public Member Functions

- **OBB** (Gs::UninitializeTag)
- **OBB** (const Vec< T > &min, const Vec< T > &max)
- **OBB** (const Vec< T > &center, const Vec< T > &xAxis, const Vec< T > &yAxis, const Vec< T > &zAxis)
- void **UpdateHalfSize** ()

### Public Attributes

- Vec< T > **center**
- Vec< T > **halfSize**
- Vec< Vec< T > > **axes**

### 7.24.1 Detailed Description

**template<template< typename > class Vec, typename T>**
**class Gm::OBB< Vec, T >**

Base OBB (Oriented Bounding-Box) class.

The documentation for this class was generated from the following file:

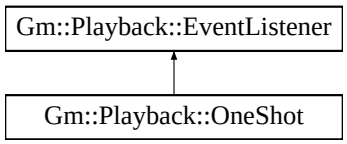
- OBB.h

## 7.25 Gm::Playback::OneShot Class Reference

One shot playback event listener.

`#include <Playback.h>`

Inheritance diagram for Gm::Playback::OneShot:




### Public Member Functions

- void OnNextFrame (Playback &sender) override

### 7.25.1 Detailed Description

One shot playback event listener.

**Remarks**

Plays an animation from the first to the last frame.

### 7.25.2 Member Function Documentation

#### 7.25.2.1 void Gm::Playback::OneShot::OnNextFrame ( Playback & *sender* ) [override], [virtual]

Receives the 'next frame' event. This event will be posted in the "Playback::Update" function.

**Remarks**

Depending on the playback speed sometimes this event will be posted several times for a single call to the "Playback::Update" function. This happens when several frames will be skipped, i.e. every single frame can be examined with this event listener interface, no matter how fast the playback speed is. All playback configurations for the next frame are done when this event is posted. This function should set the next frame (see 'nextFrame' field).

**See also**

Playback::Update
Playback::nextFrame

Reimplemented from Gm::Playback::EventListener.

The documentation for this class was generated from the following file:

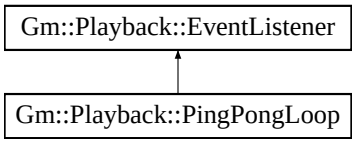
- Playback.h

## 7.26 Gm::Playback::PingPongLoop Class Reference

Ping-pong loop playback event listener.

`#include <Playback.h>`

Inheritance diagram for Gm::Playback::PingPongLoop:




### Public Member Functions

- void OnNextFrame (Playback &sender) override

### 7.26.1 Detailed Description

Ping-pong loop playback event listener.

**Remarks**

Plays an animation from the first to the last frame, and vise-versa, and then starts the animation from the beginning.

### 7.26.2 Member Function Documentation

#### 7.26.2.1 void Gm::Playback::PingPongLoop::OnNextFrame ( Playback & *sender* ) [override], [virtual]

Receives the 'next frame' event. This event will be posted in the "Playback::Update" function.

**Remarks**

Depending on the playback speed sometimes this event will be posted several times for a single call to the "Playback::Update" function. This happens when several frames will be skipped, i.e. every single frame can be examined with this event listener interface, no matter how fast the playback speed is. All playback configurations for the next frame are done when this event is posted. This function should set the next frame (see 'nextFrame' field).

**See also**

Playback::Update
Playback::nextFrame

Reimplemented from Gm::Playback::EventListener.

The documentation for this class was generated from the following file:

- Playback.h

## 7.27 Gm::MeshGenerator::PipeDescriptor Struct Reference

Descriptor structure for a pipe mesh (i.e. cylinder with a hole).

`#include <MeshGenerator.h>`

### Public Attributes

- Gs::Vector2 innerRadius = Gs::Vector2(Gs::Real(0.25))

  *Radius of the inner cylinder in U (x component), and V (y component) direction. By default (0.25, 0.25).*
- Gs::Vector2 outerRadius = Gs::Vector2(Gs::Real(0.5))

  *Radius of the outer cylinder in U (x component), and V (y component) direction. By default (0.5, 0.5).*
- Gs::Real height = Gs::Real(1)

  *Tube height. By default 1.*
- Gs::Vector2ui mantleSegments = Gs::Vector2ui(20, 1)

  *Segmentation around the (inner and outer) cylinder (x component), and height (y component). By default (20, 1).*
- unsigned int topCoverSegments = 1

  *Segmentation of the top cover. If 0, no top cover is generated. By default 1.*
- unsigned int bottomCoverSegments = 1

  *Segmentation of the top cover. If 0, no bottom cover is generated. By default 1.*
- bool alternateGrid = false

  *Specifies whether the face grids are to be alternating or uniform. By default false.*

### 7.27.1 Detailed Description

Descriptor structure for a pipe mesh (i.e. cylinder with a hole).

The documentation for this struct was generated from the following file:

- MeshGenerator.h

## 7.28 Gm::PlaneEquation_NX_eq_D< T > Struct Template Reference

### Static Public Member Functions

- static T DistanceSign (const T &d)

  *Returns 'd' (i.e. identity function).*

The documentation for this struct was generated from the following file:

- Plane.h

## 7.29 Gm::PlaneEquation_NXD_eq_Zero< T > Struct Template Reference

### Static Public Member Functions

- static T DistanceSign (const T &d)

  *Returns '-d' (i.e. negation function).*

The documentation for this struct was generated from the following file:

- Plane.h

## 7.30 Gm::PlaneT< T, PlaneEq > Class Template Reference

Plane base class with components: 'normal' and 'distance'.

`#include <Plane.h>`

### Public Member Functions

- **GM_ASSERT_FLOAT_TYPE** ("PlaneT")
- **PlaneT** (const Gs::Vector3T< T > &a, const Gs::Vector3T< T > &b, const Gs::Vector3T< T > &c)
- PlaneT (const T &x, const T &y, const T &z, const T &d)

  *Always initializes the plane with the equation n∗x + d = 0, no matter which equation this plane has as template argument.*
- **PlaneT** (const Triangle3T< T > &triangle)
- **PlaneT** (const Gs::Vector3T< T > &normal, const T &distance)
- **PlaneT** (Gs::UninitializeTag)
- void Build (const Gs::Vector3T< T > &a, const Gs::Vector3T< T > &b, const Gs::Vector3T< T > &c)

  *Builds this plane with the three specified points.*
- void Build (const Gs::Vector3T< T > &normal, const Gs::Vector3T< T > &memberPoint)

  *Builds this plane with the specified normal and member point (which lies onto the plane).*
- void UpdateDistance (const Gs::Vector3T< T > &memberPoint)

  *Updates the (signed) distance for the new specified member point.*
- void Normalize ()

  *Normalizes the normal vector and distance of this plane.*
- Gs::Vector3T< T > MemberPoint () const

  *Returns a point which lies onto this plane: normal ∗ distance;.*
- void Flip ()

  *Flips this plane.*
- PlaneT< T, PlaneEq > Flipped () const

  *Returns a flipped instance of this plane.*
- template<typename C >
  PlaneT< C, PlaneEq > **Cast** () const

### Public Attributes

- Gs::Vector3T< T > normal

  *Normal vector of the plane.*
- T distance

  *Signed distance to the origin of the coordinate system.*

## 7.30.1 Detailed Description

**template<typename T, typename PlaneEq = DefaultPlaneEquation<T>>**
**class Gm::PlaneT< T, PlaneEq >**

Plane base class with components: 'normal' and 'distance'.

#### Template Parameters

| | |
|---|---|
| *T* | Specifies the data type of the vector components. This should be a primitive data type such as float or double. |

#### Remarks

The plane equation is: ax + by + cz + d = 0, where (a, b, c) is a point on the plane, (x, y, z) is the normal vector and d is the (signed) distance to the origin.

### 7.30.2 Constructor & Destructor Documentation

#### 7.30.2.1 template<typename T, typename PlaneEq = DefaultPlaneEquation<T>> Gm::PlaneT< T, PlaneEq >::PlaneT ( const T & *x*, const T & *y*, const T & *z*, const T & *d* ) [inline]

Always initializes the plane with the equation n∗x + d = 0, no matter which equation this plane has as template argument.

**See also**

PlaneEquation_NXD_eq_Zero

### 7.30.3 Member Function Documentation

#### 7.30.3.1 template<typename T, typename PlaneEq = DefaultPlaneEquation<T>> Gs::Vector3T<T> Gm::PlaneT< T, PlaneEq >::MemberPoint ( ) const [inline]

Returns a point which lies onto this plane: normal ∗ distance;.

**Remarks**

This point is the closest point from the plane to the origin of the coordinate system.

The documentation for this class was generated from the following file:

- Plane.h

## 7.31 Gm::Playback Class Reference

Animation playback class.

`#include <Playback.h>`

### Classes

- class EventListener
  *Playback event listener interface.*
- class ListLoop
  *List loop playback event listener.*
- class Loop
  *Loop playback event listener.*
- class OneShot
  *One shot playback event listener.*
- class PingPongLoop
  *Ping-pong loop playback event listener.*

## Public Types

- enum State { State::Playing, State::Paused, State::Stopped }

  *Playback state enumeration.*
- using FrameIndex = std::size_t

  *Type of the frame indices (unsigned integral type).*

## Public Member Functions

- void Play (FrameIndex firstFrameIndex, FrameIndex lastFrameIndex, Gs::Real playbackSpeed, const std::shared_ptr< EventListener > &eventListener=nullptr)

  *Starts the playback process.*
- void Play (FrameIndex firstFrameIndex, FrameIndex lastFrameIndex, const std::shared_ptr< EventListener > &eventListener=nullptr)

  *Starts the playback process with the previous speed.*
- void Play (const std::shared_ptr< EventListener > &eventListener=nullptr)

  *Starts the playback process with the previous frame indicies and speed.*
- void Pause (bool paused=true)

  *Pauses or resumes the animation playback.*
- void Stop ()

  *Stops the animation playback. After this call the state will be State::Stopped.*
- void Update (Gs::Real deltaTime)

  *Updates the playback process. This increases (or decreases if speed is negative) the frame interpolator.*
- void SetNextFrame (FrameIndex nextFrameIndex)

  *Sets the next frame depending on the playback direction.*
- void SetNextFrame ()

  *Sets the next frame depending on the current frame, the playback direction, and the frame chronlogy.*
- bool HasEndReached () const
- State GetState () const

  *Returns the playback state.*
- bool AreFramesChrono () const

  *Returns true if the first- and last frames are chronologic.*
- bool IsForward () const

  *Returns true if the playback moves frowards.*

## Public Attributes

- FrameIndex firstFrame = 0

  *First frame index, in the range [0, +inf). By default 0.*
- FrameIndex lastFrame = 0

  *Last frame index, in the range [0, +inf). By default 0.*
- FrameIndex frame = 0

  *Current frame index, in the range [firstFrame, lastFrame]. By default 0.*
- FrameIndex nextFrame = 0

  *Next frame index, in the range [firstFrame, lastFrame]. By default 0.*
- Gs::Real interpolator = Gs::Real(0)

  *Frame interpolator, in the range [0.0, 1.0]. By default 0.*
- Gs::Real speed = Gs::Real(1)

  *Animation speed factor, in the range (-inf, +inf). By default 1.*

### 7.31.1 Detailed Description

Animation playback class.

**Remarks**

This class does not store any data about the keyframes or transformations. Only the process of playing an animation is managed by this class. Although this class has an event listener, there are no "Post..." functions, because the events are posted from the internal functions (such as "Update", "Play" etc.). No matter what direction the playback moves, an interpolation between frames is allways computed as follows:

```
auto from   = playback.frame;           // Frame index 'from' which we interpolate.
auto to     = playback.nextFrame;       // Frame index 'to' which we interpolate.
auto time   = playback.interpolator;    // Interpolation time (in the range [0.0 .. 1.0]).
auto result = Gs::Lerp(MyTransformations[from], MyTransformations[to], time);
```

### 7.31.2 Member Enumeration Documentation

#### 7.31.2.1 enum Gm::Playback::State [strong]

Playback state enumeration.

**Enumerator**

***Playing*** Animation is currently playing.

***Paused*** Animation has been paused.

***Stopped*** Animation has been stopped.

### 7.31.3 Member Function Documentation

#### 7.31.3.1 bool Gm::Playback::AreFramesChrono ( ) const [inline]

Returns true if the first- and last frames are chronologic.

**Remarks**

This is the case when the first frame is less than or equal to the last frame (firstFrame <= lastFrame).

**See also**

firstFrame
lastFrame

#### 7.31.3.2 bool Gm::Playback::HasEndReached ( ) const

Returns true if the end of animation playback has been reached.

**Remarks**

This depends on the playback direction and chronology.

**See also**

IsForward
AreFramesChrono

#### 7.31.3.3 bool Gm::Playback::IsForward ( ) const [inline]

Returns true if the playback moves frowards.

**Remarks**

This is the case when the speed is non-negative (speed >= 0.0).

**See also**

speed

#### 7.31.3.4 void Gm::Playback::Pause ( bool *paused* = true )

Pauses or resumes the animation playback.

**Parameters**

| | | |
|---|---|---|
| in | *paused* | Specifies whether the playback is to be paused or resumed. If true the playback will be paused otherwise it will be resumed. An animation playback can only be resumed if it was paused previously (state must be 'State::Paused'). An animation playback can only be paused if it was playing previously (state must be 'State::Playing'). |

**See also**

GetState
State

#### 7.31.3.5 void Gm::Playback::Play ( FrameIndex *firstFrameIndex*, FrameIndex *lastFrameIndex*, Gs::Real *playbackSpeed*, const std::shared_ptr< EventListener > & *eventListener* = nullptr )

Starts the playback process.

**Parameters**

| | | |
|---|---|---|
| in | *firstFrameIndex* | Specifies the index of the first frame. This may also be greater than the index of the last frame. |
| in | *lastFrameIndex* | Specifies the index of the last frame. This may also be less than the index of the first frame. |
| in | *newSpeed* | Specifies the animation speed factor. This may also be negative (to animate backward). By default 1.0. |
| in | *eventListener* | Shared pointer to the event listener. If this is null, the "OneShot" event listener will be used as default. By default null. |

#### 7.31.3.6 void Gm::Playback::Play ( FrameIndex *firstFrameIndex*, FrameIndex *lastFrameIndex*, const std::shared_ptr< EventListener > & *eventListener* = `nullptr` )

Starts the playback process with the previous speed.

**See also**

Play(FrameIndex, FrameIndex, Gs::Real, const std::shared_ptr<EventListener>&)

#### 7.31.3.7 void Gm::Playback::Play ( const std::shared_ptr< EventListener > & *eventListener* = `nullptr` )

Starts the playback process with the previous frame indicies and speed.

**See also**

Play(FrameIndex, FrameIndex, Gs::Real, const std::shared_ptr<EventListener>&)

#### 7.31.3.8 void Gm::Playback::SetNextFrame ( FrameIndex *nextFrameIndex* )

Sets the next frame depending on the playback direction.

**Parameters**

| | | |
|---|---|---|
| `in` | *nextFrameIndex* | Specifies the next frame index. |

**Remarks**

If the playback direction is forward the field 'nextFrame' is set, otherwise the field 'frame' is set.

**See also**

IsForward

#### 7.31.3.9 void Gm::Playback::SetNextFrame ( )

Sets the next frame depending on the current frame, the playback direction, and the frame chronlogy.

**Remarks**

The frame index range (first and last frame indices) is ignored here.

**See also**

IsForward
AreFramesChrono

#### 7.31.3.10 void Gm::Playback::Stop ( )

Stops the animation playback. After this call the state will be State::Stopped.

**See also**

GetState
State

#### 7.31.3.11 void Gm::Playback::Update ( Gs::Real *deltaTime* )

Updates the playback process. This increases (or decreases if speed is negative) the frame interpolator.

**Parameters**

| | | |
|---|---|---|
| in | *deltaTime* | Specifies the time derivation between the previous and current frame. If the application runs with 60 Hz this value should be 1.0/60.0, if the application runs with 200 Hz it should be 1.0/200.0 etc. This value will be added (and always added, also when the playback is backward) to the 'interpolator'. |

**Remarks**

If the frame interpolator is greater than or equal to 1.0 the next frame will be set. Also the "OnNextFrame" function will be called from the event listener (see EventListener class). If the animation is currently not playing (state must be 'State::Playing') or 'deltaTime' is less than or equal to 0.0, this function call has no effect.

**See also**

EventListener
Play

The documentation for this class was generated from the following file:

- Playback.h

## 7.32 Gm::UniformSpline< P, T >::Polynomial Struct Reference

Polynomial structure with four coefficients.

`#include <UniformSpline.h>`

### Public Member Functions

- const P & **operator[ ]** (std::size_t idx) const
- P & **operator[ ]** (std::size_t idx)
- P **Evaluate** (const T &t) const

### Public Attributes

- std::array< P, 4 > **coeff**

### 7.32.1 Detailed Description

**template<typename P, typename T>**
**struct Gm::UniformSpline< P, T >::Polynomial**

Polynomial structure with four coefficients.

The documentation for this struct was generated from the following file:

- UniformSpline.h

## 7.33 Gm::ProjectionT< T > Class Template Reference

projection class.

`#include <Projection.h>`

### Public Types

- using **MatrixType** = Gs::ProjectionMatrix4T< T >

### Public Member Functions

- **GM_ASSERT_FLOAT_TYPE** ("ProjectionT")
- void SetNear (const T &near)

  *Sets the near clipping plane.*
- const T & GetNear () const

  *Returns the near clipping plane.*
- void SetFar (const T &far)

  *Sets the far clipping plane.*
- const T & GetFar () const

  *Returns the far clipping plane.*
- void SetFOV (const T &fov)

  *Sets the field-of-view (FOV) in radians. By default (74∗pi/180).*
- const T & GetFOV () const

  *Returns the field-of-view (FOV) in radians.*
- void SetAspect (const T &aspect)

  *Sets the aspect ratio.*
- const T & GetAspect () const

  *Returns the aspect ratio.*
- void SetOrtho (bool isOrtho)

  *Specifies whether the projection is orthogonal or perspective. By default perspective.*
- bool GetOrtho () const

*Returns true if this projection is orthogonal.*

- void SetOrthoSize (const Gs::Vector2T< T > &orthoSize)

  *Sets the size of the orthogonal projection.*
- const Gs::Vector2T< T > & GetOrthoSize () const

  *Returns the size of the orthogonal projection.*
- void SetFlags (int flags)

  *Sets the projection matrix flags. By default 0.*
- int GetFlags () const

  *Returns the projection matrix flags.*
- const MatrixType & GetMatrix () const

  *Returns the projection matrix.*
- void GetMatrix (MatrixType &matrix, int flags) const

  *Returns the projection matrix with the specified flags.*

### 7.33.1 Detailed Description

**template<typename T>**
**class Gm::ProjectionT< T >**

projection class.

**Note**

This class can not be used with multi-threading!

### 7.33.2 Member Function Documentation

#### 7.33.2.1 template<typename T > int Gm::ProjectionT< T >::GetFlags ( ) const [inline]

Returns the projection matrix flags.

**See also**

Gs::ProjectionFlags

#### 7.33.2.2 template<typename T > void Gm::ProjectionT< T >::GetMatrix ( MatrixType & *matrix,* int *flags* ) const [inline]

Returns the projection matrix with the specified flags.

**See also**

Gs::ProjectionFlags

#### 7.33.2.3 template<typename T > void Gm::ProjectionT< T >::SetFlags ( int *flags* ) [inline]

Sets the projection matrix flags. By default 0.

**See also**

Gs::ProjectionFlags

The documentation for this class was generated from the following file:

- Projection.h

## 7.34 Gm::Ray< T > Class Template Reference

Ray base class. It's direction must always be normalized!

`#include <Ray.h>`

### Public Member Functions

- **Ray** (const T &origin, const T &direction)
- **Ray** (Gs::UninitializeTag)
- T **Lerp** (const typename Gs::ScalarType< T >::Type &t) const
- T **operator()** (const typename Gs::ScalarType< T >::Type &t) const

### Public Attributes

- T **origin**
- T **direction**

### 7.34.1 Detailed Description

**template<typename T>**
**class Gm::Ray< T >**

Ray base class. It's direction must always be normalized!

The documentation for this class was generated from the following file:

- Ray.h

## 7.35 Gm::Skeleton Class Reference

### Public Member Functions

- **Skeleton** (const Skeleton &)=delete
- Skeleton & **operator=** (const Skeleton &)=delete
- SkeletonJoint & AddRootJoint (SkeletonJointPtr &&joint)

  *Adds the specified skeleton joint and takes the ownership.*
- SkeletonJointPtr RemoveRootJoint (SkeletonJoint &joint)

  *Removes the specified skeleton joint from the list of root-joints.*
- const std::vector< SkeletonJointPtr > & GetRootJoints () const

  *Returns the list of root joints of this skeleton.*
- std::vector< SkeletonJoint * > JointList () const

  *Returns a list of all root- and sub-joints of this skeleton.*
- void ForEachJoint (const SkeletonJointIterationFunction &iterator)

  *Iterates over each joint with the specified iteration function.*
- void ForEachJoint (const SkeletonJointConstIterationFunction &iterator) const

  *Iterates over each joint with the specified iteration function.*
- void BuildJointSpace ()

  *Builds the joint-space transformations for each joint.*
- void RebuildPoseTransforms ()

  *Rebuilds the pose transformations for each joint from its joint-space transformation.*
- std::size_t NumJoints () const

  *Returns the number of all joints in thie skeleton hierarchy.*
- std::size_t FillGlobalTransformBuffer (float *buffer, std::size_t bufferSize, bool relativeTransform=true) const

  *Fills all skeleton joint matrix transformations into the specified floating-point buffer.*
- std::size_t FillLocalTransformBuffer (float *buffer, std::size_t bufferSize) const

  *Fills all skeleton joint matrix transformations into the specified floating-point buffer.*
- Skeleton & CopyFrom (const Skeleton &skeletonModel, MakeSkeletonJointFunction makeSkeleton↩ Joint=nullptr)

  *Copies the specified skeleton model into this skeleton.*

### 7.35.1 Member Function Documentation

#### 7.35.1.1 SkeletonJoint& Gm::Skeleton::AddRootJoint ( SkeletonJointPtr && *joint* )

Adds the specified skeleton joint and takes the ownership.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If the specified joint has a parent. |

**Returns**

Reference to the new skeleton joint.

#### 7.35.1.2 void Gm::Skeleton::BuildJointSpace ( )

Builds the joint-space transformations for each joint.

#### Remarks

This should be called after all joint pose transformations have been set, otherwise the 'SkeletonJoint::jointSpaceTransform' fields must be set manually.

#### See also

SkeletonJoint::poseTransform
SkeletonJoint::jointSpaceTransform

### 7.35.1.3 Skeleton& Gm::Skeleton::CopyFrom ( const Skeleton & *skeletonModel*, MakeSkeletonJointFunction *makeSkeletonJoint* = `nullptr` )

Copies the specified skeleton model into this skeleton.

#### Parameters

| | | |
|---|---|---|
| `in` | *skeletonModel* | Specifies the skeleton which is to be copied into this skeleton. |
| `in` | *makeSkeletonJoint* | Specifies an optional callback to create skeleton joints. By default the standard "SkeletonJoint" base class is created. |

#### Returns

Reference to this joint to follow the convention of copy operators.

### 7.35.1.4 std::size_t Gm::Skeleton::FillGlobalTransformBuffer ( float * *buffer*, std::size_t *bufferSize*, bool *relativeTransform* = `true` ) const

Fills all skeleton joint matrix transformations into the specified floating-point buffer.

#### Parameters

| | | |
|---|---|---|
| `in,out` | *buffer* | Specifies the output buffer. This buffer should have at least 'NumJoints() * 16' floating-point entries, 'NumJoints()' to store the matrices of all skeleton joints and '16' to store a full 4x4 matrix for each joint. |
| `in` | *bufferSize* | Specifies the size of the output buffer (in elements, not in bytes!). |
| `in` | *relativeTransform* | Specifies whether to store relative matrix transformations. This is commonly used for skeleton animation in a vertex shader. If this is false, the respective animated vertex should be multiplied with the joint-space transformation (see "SkeletonJoint::jointSpaceTransform"). If this is true, the origin transformation is already included in the transformation buffer. By default true. |

#### Returns

Number of elements written to the output buffer. In the optimal case, this should be equal to 'bufferSize'.

#### Exceptions

| | |
|---|---|
| *std::invalid_argument* | If 'buffer' is null or 'bufferSize' is not a multiple of 16. |

**See also**

NumJoints
SkeletonJoint::jointSpaceTransform

#### 7.35.1.5 std::size_t Gm::Skeleton::FillLocalTransformBuffer ( float * *buffer*, std::size_t *bufferSize* ) const

Fills all skeleton joint matrix transformations into the specified floating-point buffer.

**Parameters**

| | | |
|---|---|---|
| in,out | *buffer* | Specifies the output buffer. This buffer should have at least 'NumJoints() * 16' floating-point entries, 'NumJoints()' to store the matrices of all skeleton joints and '16' to store a full 4x4 matrix for each joint. |
| in | *bufferSize* | Specifies the size of the output buffer (in elements, not in bytes!). |

**Returns**

Number of elements written to the output buffer. In the optimal case, this should be equal to 'bufferSize'.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If 'buffer' is null or 'bufferSize' is not a multiple of 16. |

**See also**

NumJoints

#### 7.35.1.6 void Gm::Skeleton::RebuildPoseTransforms ( )

Rebuilds the pose transformations for each joint from its joint-space transformation.

**Remarks**

This should be called when the joint-space was constructed manually instead of the using "BuildJointSpace".

**See also**

SkeletonJoint::poseTransform
SkeletonJoint::jointSpaceTransform
BuildJointSpace

#### 7.35.1.7 SkeletonJointPtr Gm::Skeleton::RemoveRootJoint ( SkeletonJoint & *joint* )

Removes the specified skeleton joint from the list of root-joints.

**Returns**

Unique pointer of the removed skeleton joint, so the client programmer can take the ownership again.

The documentation for this class was generated from the following file:

- Skeleton.h

## 7.36 Gm::SkeletonJoint Class Reference

### Classes

- struct VertexWeight

  *Vertex-joint weight structure.*

### Public Types

- using TransformMatrix = Gs::AffineMatrix4

  *Transformation matrix type of skeleton joints (4x4 affine matrix).*

### Public Member Functions

- **SkeletonJoint** (const SkeletonJoint &)=delete
- SkeletonJoint & **operator=** (const SkeletonJoint &)=delete
- SkeletonJoint & AddSubJoint (SkeletonJointPtr &&joint)

  *Adds the specified skeleton joint and takes the ownership.*
- SkeletonJointPtr RemoveSubJoint (SkeletonJoint &joint)

  *Removes the specified skeleton joint from the list of sub-joints.*
- const std::vector< SkeletonJointPtr > & GetSubJoints () const

  *Returns the list of sub-joints of this skeleton joint.*
- SkeletonJoint * GetParent () const

  *Returns the parent skeleton joint or null if this joint has no parent.*
- void GlobalTransform (TransformMatrix &matrix) const

  *Stores the current global transformation of this skeleton joint in the specified output matrix parameter.*
- TransformMatrix GlobalTransform () const

  *Returns the current global transformation matrix of this skeleton joint.*

### Public Attributes

- TransformMatrix transform

  *Current local transformation of this joint.*
- TransformMatrix poseTransform

  *Local pose transformation of this joint.*
- TransformMatrix jointSpaceTransform

  *Specifies the joint-space transformation.*
- std::vector< VertexWeight > vertexWeights

  *Vertex weight, which describe how much this joint influences each vertex.*
- KeyframeSequence keyframes

  *Animation keyframe sequence.*

### Static Public Attributes

- static const std::size_t invalidID = ~0

  *Invalid ID for skeleton joints.*

### Protected Member Functions

- void BuildJointSpace (TransformMatrix parentPoseTransform)

  *Builds the joint-space transformation for this joint and all sub-joints.*
- void RebuildPoseTransforms (TransformMatrix parentPoseTransform)

  *Rebuilds the pose transformation for this joint and all sub-joints.*

### Friends

- class **Skeleton**

## 7.36.1 Member Function Documentation

#### 7.36.1.1 SkeletonJoint& Gm::SkeletonJoint::AddSubJoint ( SkeletonJointPtr && *joint* )

Adds the specified skeleton joint and takes the ownership.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If the specified joint already has a parent. |

**Returns**

Reference to the new skeleton joint.

#### 7.36.1.2 void Gm::SkeletonJoint::GlobalTransform ( TransformMatrix & *matrix* ) const

Stores the current global transformation of this skeleton joint in the specified output matrix parameter.

**See also**

transform

#### 7.36.1.3 TransformMatrix Gm::SkeletonJoint::GlobalTransform ( ) const

Returns the current global transformation matrix of this skeleton joint.

**See also**

transform

#### 7.36.1.4 SkeletonJointPtr Gm::SkeletonJoint::RemoveSubJoint ( SkeletonJoint & *joint* )

Removes the specified skeleton joint from the list of sub-joints.

**Returns**

Unique pointer of the removed skeleton joint, so the client programmer can take the ownership again.

### 7.36.2 Member Data Documentation

#### 7.36.2.1 TransformMatrix Gm::SkeletonJoint::jointSpaceTransform

Specifies the joint-space transformation.

**Remarks**

This matrix is used to transform the vertices from model-space into joint-space. This function will be overwritten whenever 'Skeleton::BuildPose' is called. Here is a vertex transformation example:

```
skinnedVertex = joint->transform * joint->jointSpaceTransform * vertex.position;
```

**See also**

poseTransform

#### 7.36.2.2 TransformMatrix Gm::SkeletonJoint::poseTransform

Local pose transformation of this joint.

**Remarks**

This is the static transformation when the joint is not being animated. When 'Skeleton::BuildPose' is called, the field 'jointSpaceTransform' will be set to the inverse global pose transformation of this joint.

**See also**

jointSpaceTransform
Skeleton::BuildPose

#### 7.36.2.3 TransformMatrix Gm::SkeletonJoint::transform

Current local transformation of this joint.

**Remarks**

This transformation will change during animation.

The documentation for this class was generated from the following file:

- SkeletonJoint.h

## 7.37 Gm::SphereT< T > Class Template Reference

Base sphere class.

```
#include <Sphere.h>
```

### Public Member Functions

- T **GetVolume** () const
- void **SetVolume** (const T &volume)
- T **GetArea** () const
- void **SetArea** (const T &area)

### Public Attributes

- Gs::Vector3T< T > **origin**
- T **radius**

### 7.37.1 Detailed Description

**template<typename T>**
**class Gm::SphereT< T >**

Base sphere class.

The documentation for this class was generated from the following file:

- Sphere.h

## 7.38 Gm::MeshGenerator::SpiralDescriptor Struct Reference

Descriptor structure for a spiral mesh.

`#include <MeshGenerator.h>`

### Public Attributes

- Gs::Vector2 ringRadius = Gs::Vector2(Gs::Real(0.5))

  *Radius of the torus ring in X, and Y direction. By default (0.5, 0.5).*
- Gs::Vector3 tubeRadius = Gs::Vector3(Gs::Real(0.25))

  *Radius of the inner tube in X, Y, and Z direction. By default (0.25, 0.25, 0.25).*
- Gs::Real displacement = Gs::Real(1)

  *The displacement for each (360 degree) turn. By default 1.*
- Gs::Real turns = Gs::Real(1)

  *Number of turns (in percent, i.e. 1.0 is a single twist, 2.5 are two and a half twist). By default 1.*
- Gs::Vector2ui mantleSegments = Gs::Vector2ui(40, 20)
- unsigned int topCoverSegments = 1

  *Segmentation of the top cover. If 0, no top cover is generated. By default 1.*
- unsigned int bottomCoverSegments = 1

  *Segmentation of the top cover. If 0, no bottom cover is generated. By default 1.*
- bool alternateGrid = false

  *Specifies whether the face grids are to be alternating or uniform. By default false.*

### 7.38.1 Detailed Description

Descriptor structure for a spiral mesh.

### 7.38.2 Member Data Documentation

#### 7.38.2.1 Gs::Vector2ui Gm::MeshGenerator::SpiralDescriptor::mantleSegments = Gs::Vector2ui(40, 20)

Segmentation of the mantle in U (x component), and V (y component) direction for a single twist. Each component will be clamped to [3, +inf). By default (40, 20).

The documentation for this struct was generated from the following file:

- MeshGenerator.h

## 7.39 Gm::Spline< P, T > Class Template Reference

Spline base class.

`#include <Spline.h>`

### Classes

- struct ControlPoint

### Public Member Functions

- **GM_ASSERT_FLOAT_TYPE** ("Spline")
- P **operator()** (const T &t) const
- const ControlPoint & **operator[ ]** (std::size_t idx) const
- ControlPoint & **operator[ ]** (std::size_t idx)
- P **Evaluate** (const T &t) const
- int **GetOrder** () const
- void **SetOrder** (int order)
- void AddPoint (const P &point, const T &t)

  *Adds a new control point.*
- const std::vector< ControlPoint > & GetPoints () const

  *Returns the list of all control points.*

### 7.39.1 Detailed Description

**template<typename P, typename T>**
**class Gm::Spline< P, T >**

Spline base class.

**Template Parameters**

| | |
|---|---|
| *P* | Specifies the type of the spline control points. |
| *T* | Specifies the base data type. This should be float or double. |

**Todo** !!!OPTIMIZE THIS!!!

### 7.39.2 Member Function Documentation

#### 7.39.2.1 template<typename P, typename T> void Gm::Spline< P, T >::AddPoint ( const P & *point*, const T & *t* ) [inline]

Adds a new control point.

**Parameters**

| | | |
|---|---|---|
| in | *point* | Specifies the point position. |
| in | *t* | Specifies the interpolation factor (or interval value). |

The documentation for this class was generated from the following file:

- Spline.h

## 7.40 Gm::MeshGenerator::TorusDescriptor Struct Reference

Descriptor structure for a torus mesh.

`#include <MeshGenerator.h>`

### Public Attributes

- Gs::Vector2 ringRadius = Gs::Vector2(Gs::Real(0.5))

  *Radius of the torus ring in X, and Y direction. By default (0.5, 0.5).*
- Gs::Vector3 tubeRadius = Gs::Vector3(Gs::Real(0.25))

  *Radius of the inner tube in X, Y, and Z direction. By default (0.25, 0.25, 0.25).*
- Gs::Vector2ui segments = Gs::Vector2ui(40, 20)

  *Segmentation in U (x component), and V (y component) direction.*
- bool alternateGrid = false

  *Specifies whether the face grids are to be alternating or uniform. By default false.*

### 7.40.1 Detailed Description

Descriptor structure for a torus mesh.

### 7.40.2 Member Data Documentation

#### 7.40.2.1 Gs::Vector2ui Gm::MeshGenerator::TorusDescriptor::segments = Gs::Vector2ui(40, 20)

Segmentation in U (x component), and V (y component) direction.

**Remarks**

Each component will be clamped to [3, +inf). By default (40, 20).

The documentation for this struct was generated from the following file:

- MeshGenerator.h

## 7.41 Gm::MeshGenerator::TorusKnotDescriptor Struct Reference

Descriptor structure for a torus-knot mesh (uses the curve generator).

`#include <MeshGenerator.h>`

### Public Attributes

- Gs::Vector3 ringRadius = Gs::Vector3(Gs::Real(0.25))
  *Radius of the torus ring in X, and Y direction. By default (0.25, 0.25, 0.25).*
- Gs::Real tubeRadius = Gs::Real(0.125)
  *Radius of the inner tube. By default 0.125.*
- Gs::Real innerRadius = Gs::Real(2)
  *Inner radius within the torus knot curve. By default 2.*
- unsigned int loops = 2
- unsigned int turns = 3
- Gs::Vector2ui segments = Gs::Vector2ui(256, 20)
  *Segmentation in U (x component), and V (y component) direction.*
- bool alternateGrid = false
  *Specifies whether the face grids are to be alternating or uniform. By default false.*
- VertexModifier vertexModifier = nullptr
  *Vertex modifier to adjust the tube radius during mesh generation.*

### 7.41.1 Detailed Description

Descriptor structure for a torus-knot mesh (uses the curve generator).

### 7.41.2 Member Data Documentation

#### 7.41.2.1 unsigned int Gm::MeshGenerator::TorusKnotDescriptor::loops = 2

Number of loops within the torus knot. By default 2.

**Remarks**

This must be coprime to 'turns', otherwise the mesh will not be a valid torus knot.

#### 7.41.2.2 Gs::Vector2ui Gm::MeshGenerator::TorusKnotDescriptor::segments = Gs::Vector2ui(256, 20)

Segmentation in U (x component), and V (y component) direction.

**Remarks**

Each component will be clamped to [3, +inf). By default (256, 20).

#### 7.41.2.3 unsigned int Gm::MeshGenerator::TorusKnotDescriptor::turns = 3

Number of turns within the torus knot. By default 3.

**Remarks**

This must be coprime to 'loops', otherwise the mesh will not be a valid torus knot.

The documentation for this struct was generated from the following file:

- MeshGenerator.h

## 7.42 Gm::Transform2T< T > Class Template Reference

2D transformation class.

`#include <Transform2.h>`

### Public Types

- using **MatrixType** = Gs::AffineMatrix3T< T >

### Public Member Functions

- **GM_ASSERT_FLOAT_TYPE** ("Transform2T")
- void **SetPosition** (const Gs::Vector2T< T > &position)
- const Gs::Vector2T< T > & **GetPosition** () const
- void **SetRotation** (const T &rotation)
- const T & **GetRotation** () const
- void **SetScale** (const Gs::Vector2T< T > &scale)
- const Gs::Vector2T< T > & **GetScale** () const
- const MatrixType & **GetMatrix** () const
- void **Turn** (const T &rotation, const Gs::Vector2T< T > &pivot)

### 7.42.1 Detailed Description

**template<typename T>**
**class Gm::Transform2T< T >**

2D transformation class.

**Note**

This class can not be used with multi-threading!

The documentation for this class was generated from the following file:

- Transform2.h

## 7.43 Gm::Transform3T< T > Class Template Reference

3D transformation class.

`#include <Transform3.h>`

### Public Types

- using **MatrixType** = Gs::AffineMatrix4T< T >

### Public Member Functions

- **GM_ASSERT_FLOAT_TYPE** ("Transform3T")
- **Transform3T** (const Gs::AffineMatrix4T< T > &matrix)
- void **SetPosition** (const Gs::Vector3T< T > &position)
- const Gs::Vector3T< T > & **GetPosition** () const
- void **SetRotation** (const Gs::QuaternionT< T > &rotation)
- const Gs::QuaternionT< T > & **GetRotation** () const
- void **SetScale** (const Gs::Vector3T< T > &scale)
- const Gs::Vector3T< T > & **GetScale** () const
- const MatrixType & **GetMatrix** () const
- void MoveGlobal (const Gs::Vector3T< T > &direction)

  *Moves this transformation into the specified direction.*
- void MoveLocal (const Gs::Vector3T< T > &direction)

  *Moves this transformation into the specified direction with respect to the current rotation.*
- void Turn (const Gs::QuaternionT< T > &rotation, const Gs::Vector3T< T > &pivot)

  *Turns this transformation with the specified (reltative) angles around the (global) pivot.*

### 7.43.1 Detailed Description

**template<typename T>**
**class Gm::Transform3T< T >**

3D transformation class.

**Note**

This class can not be used with multi-threading!

The documentation for this class was generated from the following file:

- Transform3.h

## 7.44 Gm::Triangle< T > Class Template Reference

Triangle base class.

`#include <Triangle.h>`

### Public Member Functions

- **Triangle** (const Triangle< T > &)=default
- **Triangle** (const T &a, const T &b, const T &c)
- **Triangle** (Gs::UninitializeTag)
- T & **operator[ ]** (std::size_t vertex)
- const T & **operator[ ]** (std::size_t vertex) const

### Public Attributes

- T **a**
- T **b**
- T **c**

### 7.44.1 Detailed Description

**template<typename T>**
**class Gm::Triangle< T >**

Triangle base class.

The documentation for this class was generated from the following file:

- Triangle.h

## 7.45 Gm::Triangle< Gs::Vector2T< T > > Class Template Reference

Template specializationn for 2D triangles.

`#include <Triangle.h>`

### Public Member Functions

- **Triangle** (const Triangle< Gs::Vector2T< T > > &)=default
- **Triangle** (const Gs::Vector2T< T > &a, const Gs::Vector2T< T > &b, const Gs::Vector2T< T > &c)
- **Triangle** (Gs::UninitializeTag)
- Gs::Vector2T< T > & **operator[ ]** (std::size_t vertex)
- const Gs::Vector2T< T > & **operator[ ]** (std::size_t vertex) const
- T **Area** () const
- Gs::Vector3T< T > Normal () const

  *Returns the normal vector of this triangle.*
- Gs::Vector3T< T > UnitNormal () const

  *Returns the normal vector of this triangle in unit length (length = 1.0).*
- Gs::Vector2T< T > BarycentricToCartesian (const Gs::Vector3T< T > &barycentricCoord) const

  *Computes the cartesian coordinate by the specified barycentric coordinate with respect to this triangle.*
- Triangle< Gs::Vector2T< T > > BarycentricToCartesian (const Triangle< Gs::Vector3T< T > > &barycentricTriangle) const

  *Computes the triangle with cartesian coordinates by the specified triangle with barycentric coordinates with respecti to this triangle.*
- Gs::Vector3T< T > CartesianToBarycentric (const Gs::Vector2T< T > &cartesianCoord) const

  *Computes the barycentric coordinate with respect to this triangle by the specified cartesian coordinate.*
- T Angle (std::size_t vertex) const

  *Returns the angle (in radians) of the specified triangle vertex (0, 1, or 2).*

### Public Attributes

- Gs::Vector2T< T > **a**
- Gs::Vector2T< T > **b**
- Gs::Vector2T< T > **c**

### 7.45.1 Detailed Description

**template<typename T>**
**class Gm::Triangle< Gs::Vector2T< T > >**

Template specializationn for 2D triangles.

### 7.45.2 Member Function Documentation

#### 7.45.2.1 template<typename T > Gs::Vector2T<T> Gm::Triangle< Gs::Vector2T< T > >::BarycentricToCartesian ( const Gs::Vector3T< T > & *barycentricCoord* ) const [inline]

Computes the cartesian coordinate by the specified barycentric coordinate with respect to this triangle.

**Parameters**

| | | |
|---|---|---|
| in | *barycentricCoord* | Specifies the barycentric coordinates with respect to this triangle. The sum of all components must be one, i.e. x+y+z = 1. |

#### 7.45.2.2 template<typename T > Triangle< Gs::Vector2T<T> > Gm::Triangle< Gs::Vector2T< T > >::BarycentricToCartesian ( const Triangle< Gs::Vector3T< T > > & *barycentricTriangle* ) const [inline]

Computes the triangle with cartesian coordinates by the specified triangle with barycentric coordinates with respecti to this triangle.

**Parameters**

| | | |
|---|---|---|
| in | *barycentricTriangle* | Specifies the triangle with barycentric coordinates with respect to this triangle. If this input parameter is { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } }, the result is equal to this triangle. The sum of all components must be one for each triangle vertex, i.e. x+y+z = 1. |

#### 7.45.2.3 template<typename T > Gs::Vector3T<T> Gm::Triangle< Gs::Vector2T< T > >::CartesianToBarycentric ( const Gs::Vector2T< T > & *cartesianCoord* ) const [inline]

Computes the barycentric coordinate with respect to this triangle by the specified cartesian coordinate.

**Parameters**

| | | |
|---|---|---|
| in | *cartesianCoord* | Specifies the cartesian coordinate. |

**Returns**

Barycentric coordinate with respect to this triangle.

#### 7.45.2.4 template<typename T > Gs::Vector3T<T> Gm::Triangle< Gs::Vector2T< T > >::Normal ( ) const [inline]

Returns the normal vector of this triangle.

**Remarks**

This normal vector is not guaranteed to have a unit length of 1.0! To get a normal vector of unit length use "UnitNormal".

**See also**

UnitNormal

The documentation for this class was generated from the following file:

- Triangle.h

# 7.46 Gm::Triangle< Gs::Vector3T< T > > Class Template Reference

Template specializationn for 3D triangles.

`#include <Triangle.h>`

## Public Member Functions

- **Triangle** (const Triangle< Gs::Vector3T< T > > &)=default
- **Triangle** (const Gs::Vector3T< T > &a, const Gs::Vector3T< T > &b, const Gs::Vector3T< T > &c)
- **Triangle** (Gs::UninitializeTag)
- Gs::Vector3T< T > & **operator[ ]** (std::size_t vertex)
- const Gs::Vector3T< T > & **operator[ ]** (std::size_t vertex) const
- T Area () const

  *Returns the area of this triangle.*
- Gs::Vector3T< T > Normal () const

  *Returns the normal vector of this triangle.*
- Gs::Vector3T< T > UnitNormal () const

  *Returns the normal vector of this triangle in unit length (length = 1.0).*
- Gs::Vector3T< T > BarycentricToCartesian (const Gs::Vector3T< T > &barycentricCoord) const

  *Computes the cartesian coordinate by the specified barycentric coordinate with respect to this triangle.*
- Triangle< Gs::Vector3T< T > > BarycentricToCartesian (const Triangle< Gs::Vector3T< T > > &barycentricTriangle) const

  *Computes the triangle with cartesian coordinates by the specified triangle with barycentric coordinates with respecti to this triangle.*
- Gs::Vector3T< T > CartesianToBarycentric (const Gs::Vector3T< T > &cartesianCoord) const

  *Computes the barycentric coordinate with respect to this triangle by the specified cartesian coordinate.*
- T Angle (std::size_t vertex) const

  *Returns the angle (in radians) of the specified triangle vertex (0, 1, or 2).*

## Public Attributes

- Gs::Vector3T< T > **a**
- Gs::Vector3T< T > **b**
- Gs::Vector3T< T > **c**

### 7.46.1 Detailed Description

**template<typename T>**
**class Gm::Triangle< Gs::Vector3T< T > >**

Template specializationn for 3D triangles.

### 7.46.2 Member Function Documentation

#### 7.46.2.1 template<typename T> Gs::Vector3T<T> Gm::Triangle< Gs::Vector3T< T > >::BarycentricToCartesian ( const Gs::Vector3T< T > & *barycentricCoord* ) const [inline]

Computes the cartesian coordinate by the specified barycentric coordinate with respect to this triangle.

**Parameters**

| | | |
|---|---|---|
| in | *barycentricCoord* | Specifies the barycentric coordinates with respect to this triangle. The sum of all components must be one, i.e. x+y+z = 1. |

#### 7.46.2.2 template<typename T> Triangle< Gs::Vector3T<T> > Gm::Triangle< Gs::Vector3T< T > >::BarycentricToCartesian ( const Triangle< Gs::Vector3T< T > > & *barycentricTriangle* ) const [inline]

Computes the triangle with cartesian coordinates by the specified triangle with barycentric coordinates with respecti to this triangle.

**Parameters**

| | | |
|---|---|---|
| in | *barycentricTriangle* | Specifies the triangle with barycentric coordinates with respect to this triangle. If this input parameter is { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } }, the result is equal to this triangle. The sum of all components must be one for each triangle vertex, i.e. x+y+z = 1. |

#### 7.46.2.3 template<typename T> Gs::Vector3T<T> Gm::Triangle< Gs::Vector3T< T > >::CartesianToBarycentric ( const Gs::Vector3T< T > & *cartesianCoord* ) const [inline]

Computes the barycentric coordinate with respect to this triangle by the specified cartesian coordinate.

**Parameters**

| | | |
|---|---|---|
| in | *cartesianCoord* | Specifies the cartesian coordinate. |

**Returns**

Barycentric coordinate with respect to this triangle.

#### 7.46.2.4 template<typename T> Gs::Vector3T<T> Gm::Triangle< Gs::Vector3T< T > >::Normal ( ) const [inline]

Returns the normal vector of this triangle.

**Remarks**

This normal vector is not guaranteed to have a unit length of 1.0! To get a normal vector of unit length use "UnitNormal".

**See also**

UnitNormal

The documentation for this class was generated from the following file:

- Triangle.h

## 7.47 Gm::TriangleMesh Class Reference

Triangle mesh base class.

`#include <TriangleMesh.h>`

### Classes

- struct Vertex

  *Base vertex structure. Contains the members: position, normal, and texCoord.*

### Public Types

- using **VertexIndex** = std::size_t
- using **TriangleIndex** = std::size_t
- using **Edge** = Gm::Line< VertexIndex >
- using **Triangle** = Gm::Triangle< VertexIndex >

### Public Member Functions

- **TriangleMesh** (const TriangleMesh &)=default
- **TriangleMesh** (TriangleMesh &&rhs)
- TriangleMesh & **operator=** (const TriangleMesh &rhs)=default
- TriangleMesh & **operator=** (TriangleMesh &&rhs)
- void Clear ()

  *Clears all vertices and triangles.*
- VertexIndex AddVertex (const Gs::Vector3 &position, const Gs::Vector3 &normal, const Gs::Vector2 &texCoord)

  *Adds a new vertex with the specified attributes and returns the index of the new vertex.*
- TriangleIndex AddTriangle (VertexIndex v0, VertexIndex v1, VertexIndex v2)

  *Adds a new triangle with the specified three indices and returns the index of the new triangle.*
- Vertex Barycentric (TriangleIndex triangleIndex, const Gs::Vector3 &barycentricCoords) const

  *Returns the vertex, interpolated from the triangle with the specified barycentric coordinates.*
- std::vector< Edge > Edges () const

  *Computes the set of all triangle edges.*
- std::vector< Edge > SilhouetteEdges (Gs::Real toleranceAngle=Gs::Real(0)) const

  *Computes the set of all triangle edges which are part of the silhouette.*
- std::set< TriangleIndex > TriangleNeighbors (std::set< TriangleIndex > triangleIndices, std::size_t searchDepth=1, bool edgeBondOnly=false, bool searchViaPosition=false) const

  *Computes the list of all neighbors of the specified triangle.*
- std::vector< TriangleIndex > FindTriangles (VertexIndex vertexIndex) const

  *Computes the list of all triangles that are connected to the specified vertex.*
- std::vector< TriangleIndex > FindTriangles (const Edge &edge) const

  *Computes the list of all triangles that are connected to the specified edge.*
- std::vector< Gm::Triangle< Vertex > > TriangleList () const

  *Computes the list of all triangles with their own vertices, but without indices.*
- Gs::Vector3 TriangleNormal (TriangleIndex triangleIndex) const

  *Returns the normal vector of the specified triangle (in unit length of 1.0).*
- AABB3 BoundingBox () const

  *Computes the axis-aligned bounding-box of this mesh.*
- AABB3 BoundingBox (const Gs::AffineMatrix4 &matrix) const

  *Computes the axis-aligned bounding-box of this mesh with the specified transformation matrix.*
- void Append (const TriangleMesh &other)

  *Appends the specified triangle mesh to this mesh.*

### Public Attributes

- std::vector< Vertex > vertices

  *Vertex array list.*
- std::vector< Triangle > triangles

  *Triangle array list. Make sure that all triangle indices are less than the number of vertices of this mesh!*

## 7.47.1 Detailed Description

Triangle mesh base class.

**Remarks**

This class is used for generation and modification of all triangle meshes. However, it is only meant to be used to operate with this library, but not to use it within the graphics engine of your project.

## 7.47.2 Member Function Documentation

### 7.47.2.1 std::vector<Edge> Gm::TriangleMesh::SilhouetteEdges ( Gs::Real *toleranceAngle* = `Gs::Real(0)` ) const

Computes the set of all triangle edges which are part of the silhouette.

**Parameters**

| | | |
|---|---|---|
| in | *toleranceAngle* | Specifies the tolerance angle (in radians) to reject edges. Must be in the range [0, pi]. |

**See also**

Edges

### 7.47.2.2 std::set<TriangleIndex> Gm::TriangleMesh::TriangleNeighbors ( std::set< TriangleIndex > *triangleIndices*, std::size_t *searchDepth* = `1`, bool *edgeBondOnly* = `false`, bool *searchViaPosition* = `false` ) const

Computes the list of all neighbors of the specified triangle.

**Parameters**

| | | |
|---|---|---|
| in | *triangleIndices* | Specifies the indices of the root triangles to search for neighbors. |
| in | *searchDepth* | Specifies the number of iterations |
| in | *edgeBondOnly* | Specifies whether to only search triangles that are connected at their edges and not only at their corners. By default false. |
| in | *searchViaPosition* | Specifies whether to search triangles via the position of their vertices (true), or only search via the index of their vertices (false). By default false. |

**Returns**

Set of triangle indices of the neighbor search result including the input triangle indices.

The documentation for this class was generated from the following file:

- TriangleMesh.h

## 7.48 Gm::UniformSpline< P, T > Class Template Reference

Spline class with uniform weights.

`#include <UniformSpline.h>`

### Classes

- struct Polynomial

  *Polynomial structure with four coefficients.*

### Public Member Functions

- **GM_ASSERT_FLOAT_TYPE** ("UniformSpline")
- P **operator()** (const T &t) const
- void Build (const std::vector< P > &points, const T &expansion=T(1))

  *Builds the spline polynomials.*
- void Clear ()

  *Clears the spline polynoms.*
- const Polynomial & **operator[ ]** (std::size_t idx) const
- Polynomial & **operator[ ]** (std::size_t idx)
- P **Evaluate** (T t) const
- const std::vector< Polynomial > & **GetPolynomials** () const

### Static Public Attributes

- static const std::size_t dimension = P::components

  *Spline dimension (e.g. 2 for a 2D-vector).*

### 7.48.1 Detailed Description

**template<typename P, typename T>**
**class Gm::UniformSpline< P, T >**

Spline class with uniform weights.

**See also**

Spline

### 7.48.2 Member Function Documentation

#### 7.48.2.1 template<typename P, typename T> void Gm::UniformSpline< P, T >::Build ( const std::vector< P > & *points*, const T & *expansion* = `T(1)` ) `[inline]`

Builds the spline polynomials.

**Parameters**

| | | |
|---|---|---|
| in | *points* | Specifies the control points. |
| in | *expansion* | Specifies the expansion of the polynomials. If the expansion is 0.0, this spline will be a linear spline. By default 1.0. |

The documentation for this class was generated from the following file:

- UniformSpline.h

## 7.49 Gm::TriangleMesh::Vertex Struct Reference

Base vertex structure. Contains the members: position, normal, and texCoord.

`#include <TriangleMesh.h>`

### Public Member Functions

- **Vertex** (const Vertex &)=default
- **Vertex** (const Gs::Vector3 &position, const Gs::Vector3 &normal, const Gs::Vector2 &texCoord)
- Vertex & **operator+=** (const Vertex &rhs)
- Vertex & **operator∗=** (Gs::Real rhs)

### Public Attributes

- Gs::Vector3 **position**
- Gs::Vector3 **normal**
- Gs::Vector2 **texCoord**

### 7.49.1 Detailed Description

Base vertex structure. Contains the members: position, normal, and texCoord.

The documentation for this struct was generated from the following file:

- TriangleMesh.h

## 7.50 Gm::MeshModifier::VertexAttributeDescriptor Struct Reference

Vertex attribute descriptor structure.

`#include <MeshModifier.h>`

### Public Member Functions

- **VertexAttributeDescriptor** (std::size_t offset, unsigned int components)

### Public Attributes

- std::size_t offset = 0

  *Byte offset within each vertex.*
- unsigned int components = 1

  *Number of components of this vertex attribute. By default 1.*

#### 7.50.1 Detailed Description

Vertex attribute descriptor structure.

**Note**

This mesh modifier can only handle vertex attributes with components of type "Gs::Real".

The documentation for this struct was generated from the following file:

- MeshModifier.h

## 7.51 Gm::MeshModifier::VertexDescriptor Struct Reference

Vertex descriptor structure.

`#include <MeshModifier.h>`

### Public Member Functions

- **VertexDescriptor** (const std::vector< VertexAttributeDescriptor > &attributes, std::size_t stride=0)

### Public Attributes

- std::vector< VertexAttributeDescriptor > attributes

  *Vertex attribute descriptors.*
- std::size_t stride = 0

  *Byte offset to the next vertex. By default 0.*

#### 7.51.1 Detailed Description

Vertex descriptor structure.

### 7.51.2 Member Data Documentation

#### 7.51.2.1 std::size_t Gm::MeshModifier::VertexDescriptor::stride = 0

Byte offset to the next vertex. By default 0.

**Remarks**

If this is zero, the size of all vertex attributes is used.

The documentation for this struct was generated from the following file:

- MeshModifier.h

## 7.52 Gm::SkeletonJoint::VertexWeight Struct Reference

Vertex-joint weight structure.

`#include <SkeletonJoint.h>`

### Public Member Functions

- **VertexWeight** (const VertexWeight &)=default
- **VertexWeight** (TriangleMesh::VertexIndex index, Gs::Real weight)

### Public Attributes

- TriangleMesh::VertexIndex index = 0

  *Vertex index within the respective mesh.*
- Gs::Real weight = Gs::Real(0)

  *Weight factor. This should be in the range (0, +inf).*

### 7.52.1 Detailed Description

Vertex-joint weight structure.

**Remarks**

Vertex weight determine how much a skeleton joint influences a vertex.

The documentation for this struct was generated from the following file:

- SkeletonJoint.h

# Index


AABB
    Gm::AABB, 20
AddPoint
    Gm::Spline, 62
AddRootJoint
    Gm::Skeleton, 54
AddSubJoint
    Gm::SkeletonJoint, 58
AreFramesChrono
    Gm::Playback, 47

BarycentricToCartesian
    Gm::Triangle< Gs::Vector2T< T > >, 67, 68
    Gm::Triangle< Gs::Vector3T< T > >, 69, 70
Build
    Gm::UniformSpline, 73
BuildJointSpace
    Gm::Skeleton, 54
BuildKeys
    Gm::KeyframeSequence, 34

CartesianToBarycentric
    Gm::Triangle< Gs::Vector2T< T > >, 68
    Gm::Triangle< Gs::Vector3T< T > >, 70
ClipMesh
    Gm::MeshModifier, 16
CopyFrom
    Gm::Skeleton, 55
CurveFunction
    Gm::MeshGenerator, 15

ellipsoidSegments
    Gm::MeshGenerator::CapsuleDescriptor, 25
Evaluate
    Gm::BezierPatch, 21

FillGlobalTransformBuffer
    Gm::Skeleton, 55
FillLocalTransformBuffer
    Gm::Skeleton, 56

GetControlPoint
    Gm::BezierPatch, 22
    Gm::BezierTriangle, 24
GetDefaultVertexDesc
    Gm::MeshModifier, 16
GetFlags
    Gm::ProjectionT, 52
GetMatrix
    Gm::ProjectionT, 52
GlobalTransform
    Gm::SkeletonJoint, 58
Gm::AABB< Vec, T >, 19
Gm::AABB
    AABB, 20
Gm::BezierCurve< P, T >, 20
Gm::BezierPatch
    Evaluate, 21
    GetControlPoint, 22
    SetControlPoint, 22
Gm::BezierPatch< P, T >, 21
Gm::BezierTriangle
    GetControlPoint, 24
    SetControlPoint, 24
Gm::BezierTriangle< P, T >, 23
Gm::ClippedPolygon< T >, 25
Gm::ConvexHullT< T, PlaneEq >, 26
Gm::ConvexHullT
    Normalize, 27
    planes, 27
Gm::FrustumT< T, PlaneEq >, 32
Gm::Keyframe< T >, 32
Gm::KeyframeSequence, 33
    BuildKeys, 34
    Interpolate, 34
Gm::Line< Gs::Vector2T< T > >, 35
Gm::Line< Gs::Vector3T< T > >, 36
Gm::Line< T >, 35
Gm::MeshGenerator, 13
    CurveFunction, 15
    VertexModifier, 15
Gm::MeshGenerator::BezierPatchDescriptor, 22
    segments, 23
Gm::MeshGenerator::CapsuleDescriptor, 24
    ellipsoidSegments, 25
Gm::MeshGenerator::ConeDescriptor, 26
Gm::MeshGenerator::CuboidDescriptor, 27
Gm::MeshGenerator::CurveDescriptor, 28
    segments, 29
Gm::MeshGenerator::CylinderDescriptor, 29
Gm::MeshGenerator::EllipsoidDescriptor, 30
    segments, 30
Gm::MeshGenerator::PipeDescriptor, 42
Gm::MeshGenerator::SpiralDescriptor, 60
    mantleSegments, 61
Gm::MeshGenerator::TorusDescriptor, 62
    segments, 63
Gm::MeshGenerator::TorusKnotDescriptor, 63
    loops, 63
    segments, 63

turns, 64
Gm::MeshModifier, 15
ClipMesh, 16
GetDefaultVertexDesc, 16
InterpolateBarycentric, 16
Gm::MeshModifier::VertexAttributeDescriptor, 74
Gm::MeshModifier::VertexDescriptor, 75
stride, 76
Gm::OBB< Vec, T >, 39
Gm::PlaneEquation_NX_eq_D< T >, 43
Gm::PlaneEquation_NXD_eq_Zero< T >, 43
Gm::PlaneT< T, PlaneEq >, 43
Gm::PlaneT
MemberPoint, 45
PlaneT, 45
Gm::Playback, 45
AreFramesChrono, 47
HasEndReached, 47
IsForward, 47
Pause, 48
Paused, 47
Play, 48, 49
Playing, 47
SetNextFrame, 49
State, 47
Stop, 49
Stopped, 47
Update, 50
Gm::Playback::EventListener, 30
OnNextFrame, 31
OnPause, 31
OnPlay, 31
OnStop, 31
Gm::Playback::ListLoop, 37
OnNextFrame, 38
OnStop, 38
Gm::Playback::Loop, 38
OnNextFrame, 39
Gm::Playback::OneShot, 40
OnNextFrame, 41
Gm::Playback::PingPongLoop, 41
OnNextFrame, 42
Gm::ProjectionT< T >, 51
Gm::ProjectionT
GetFlags, 52
GetMatrix, 52
SetFlags, 52
Gm::Ray< T >, 53
Gm::Skeleton, 54
AddRootJoint, 54
BuildJointSpace, 54
CopyFrom, 55
FillGlobalTransformBuffer, 55
FillLocalTransformBuffer, 56
RebuildPoseTransforms, 56
RemoveRootJoint, 56
Gm::SkeletonJoint, 57
AddSubJoint, 58
GlobalTransform, 58
jointSpaceTransform, 59
poseTransform, 59
RemoveSubJoint, 58
transform, 59
Gm::SkeletonJoint::VertexWeight, 76
Gm::SphereT< T >, 59
Gm::Spline
AddPoint, 62
Gm::Spline< P, T >, 61
Gm::Spline< P, T >::ControlPoint, 26
Gm::Transform2T< T >, 64
Gm::Transform3T< T >, 65
Gm::Triangle< Gs::Vector2T< T > >, 67
BarycentricToCartesian, 67, 68
CartesianToBarycentric, 68
Normal, 68
Gm::Triangle< Gs::Vector3T< T > >, 69
BarycentricToCartesian, 69, 70
CartesianToBarycentric, 70
Normal, 70
Gm::Triangle< T >, 66
Gm::TriangleMesh, 71
SilhouetteEdges, 72
TriangleNeighbors, 72
Gm::TriangleMesh::Vertex, 74
Gm::UniformSpline
Build, 73
Gm::UniformSpline< P, T >, 73
Gm::UniformSpline< P, T >::Polynomial, 50

HasEndReached
Gm::Playback, 47

Interpolate
Gm::KeyframeSequence, 34
InterpolateBarycentric
Gm::MeshModifier, 16
IsForward
Gm::Playback, 47

jointSpaceTransform
Gm::SkeletonJoint, 59

loops
Gm::MeshGenerator::TorusKnotDescriptor, 63

mantleSegments
Gm::MeshGenerator::SpiralDescriptor, 61
MemberPoint
Gm::PlaneT, 45

Normal
Gm::Triangle< Gs::Vector2T< T > >, 68
Gm::Triangle< Gs::Vector3T< T > >, 70
Normalize
Gm::ConvexHullT, 27

OnNextFrame
Gm::Playback::EventListener, 31

Gm::Playback::ListLoop, 38
Gm::Playback::Loop, 39
Gm::Playback::OneShot, 41
Gm::Playback::PingPongLoop, 42
OnPause
Gm::Playback::EventListener, 31
OnPlay
Gm::Playback::EventListener, 31
OnStop
Gm::Playback::EventListener, 31
Gm::Playback::ListLoop, 38

Pause
Gm::Playback, 48
Paused
Gm::Playback, 47
planes
Gm::ConvexHullT, 27
PlaneT
Gm::PlaneT, 45
Play
Gm::Playback, 48, 49
Playing
Gm::Playback, 47
poseTransform
Gm::SkeletonJoint, 59

RebuildPoseTransforms
Gm::Skeleton, 56
RemoveRootJoint
Gm::Skeleton, 56
RemoveSubJoint
Gm::SkeletonJoint, 58

segments
Gm::MeshGenerator::BezierPatchDescriptor, 23
Gm::MeshGenerator::CurveDescriptor, 29
Gm::MeshGenerator::EllipsoidDescriptor, 30
Gm::MeshGenerator::TorusDescriptor, 63
Gm::MeshGenerator::TorusKnotDescriptor, 63
SetControlPoint
Gm::BezierPatch, 22
Gm::BezierTriangle, 24
SetFlags
Gm::ProjectionT, 52
SetNextFrame
Gm::Playback, 49
SilhouetteEdges
Gm::TriangleMesh, 72
State
Gm::Playback, 47
Stop
Gm::Playback, 49
Stopped
Gm::Playback, 47
stride
Gm::MeshModifier::VertexDescriptor, 76

transform
Gm::SkeletonJoint, 59
TriangleNeighbors
Gm::TriangleMesh, 72
turns
Gm::MeshGenerator::TorusKnotDescriptor, 64

Update
Gm::Playback, 50

VertexModifier
Gm::MeshGenerator, 15