MTRX Engine

Generated by Doxygen 1.8.16

Chapter 1

MTRXEngine

1.1 Author:

Mohamed Kazma

1.2 Description:

A basic C++ based physics engine which uses some libraries like GLM, SPDLOG, GLAD, GLFW.

- Implements rigidbody dynamics with force and torque integration.
- Implements force generators that can be used to simulate certain forces like gravity, drag, buoyancy, stiff springs, etc...
- Implements a basic collision detection system with basic bounding volumes (sphere, capsule, box).
- Implements convex shape colliders with GJK Collision detection
- Basic raycasting that can be used to grab a certain collider using a certain ray with the option to filter out certain colliders
- Very rough implementation of Bounding Volume Hierarchies(untested) that can be used to optimize collision detection checks

1.3 Book Sources

These are some of the books that i have used as inspiration:

```
Real-Time Collision Detection - Christer Ericson
Game Physics Engine Development - Ian Millington
Game Engine Architecture - Jason Gregory
```

1.4 Fixing timesteps:

https://web.archive.org/web/20180321070852/https://.gafferongames.com/post/fix_your_timestep

1.5 GJK implementation sources:

https://caseymuratori.com/blog_0003 http://www.dyn4j.org/2010/04/gjk-gilbert-johnson-ke
http://vec3.ca/gjk/implementation/ http://in2gpu.com/2014/05/18/gjk-algorithm-3d/

1.6 Segment-Segment minimum distance implementation:

http://geomalgorithms.com/a07-_distance.html#dist3D_Segment_to_Segment

1.7 TRELLO

https://trello.com/b/Cwb55iBt/mtrx-engine

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Namespace Documentation

5.1 mtrx Namespace Reference

Namespaces

- ColliderDetectionUtil

  Utility namespace used for resolving collision detection algorithms to be used based on collider types.
- CollisionUtil

  Namespace used to define some collision detection algorithms that are used to check for collision between different colliders. None of these functions require colliders themselves, They use mostly primitive classes instead.
- PhysicsUtil

  Utility namespace for some basic calculations that are useful for collision detection and other physics related functionality.
- RaycastCollisionUtil

  Utility namespace used for raycasting functionality.

Classes

- class AABBCollider

  Implmentation of an Axis Aligned Bounding Box used for collision systems.
- class Body

  Implementation of a basic particle body.
- class BVHNode

  An implementation of a BVH node used for Bounding Volume Hierarchies.
- class CapsuleCollider

  Implementation of Capsule colliders.
- class Collider

  Basic API for all colliders.
- class CollisionSystem

  The collision detection system of the engine.
- class ConvexShapeCollider

  Implementation of a Convex Shape collider.
- class GameTime

  Basic implementation of time values mostly used for the calculation of delta time.
- class GJKUtil

Utility class for the implementation of the GJK algorithm used for collision detection between convex shape colliders.

- class IBoundingVolume

  Interface used to define a bounding volume that can be used in a Bounding volume Hierarchy.
- class IIntegratable

  Interface for integration PS: Not used that much really.
- class IRigidbodyForceGenerator

  Force generation interface used to apply forces on rigidbodies.
- class LogManager

  Logger wrapper for logging functionality in the engine.
- class mtrxDynamicWorld

  Entry point of the user to the engine. This class defines the API that user will be using to interface with the engine.
- struct ObjectAxes

  The axes that define an objects world.
- class OOBBCollider

  Implementation of OOBBs.
- struct PotentialCollision

  Potential collisions struct.
- class Ray

  Implementation of a ray.
- class rb_BuoyancyForceGenerator

  Implementation of a buoyancy force generator for rigidbodies.
- class rb_ForceGenerationRegistry

  Registry for force generators used to map a rigidbody to all of the force generators that are to be applied to said rigidbody.
- class rb_GravityForceGenerator

  Implementation of gravitational force generator.
- class Rigidbody

  Implementation of rigidbodies with rigidbody dynamics newtonian force integration and rotational forces.
- class RigidbodyManager

  Manager of rigidbodies.
- struct Simplex

  Simplest shape that can encapsulate a point in 3d space Used within GJK collision detection algorithm.
- class SphereCollider

  Implmentation of a Sphere collider used in collision systems.
- class Transform

  Wrapper implementation of a transform that holds position, orientation, scale values and some functionality based on these values.
- struct Triangle

  Struct for storing a triangle.

Enumerations

- enum ColliderType : char {
  Sphere, **AABB**, **OOBB**, **Capsule**,
  ConvexShape }

  Type of colliders that are supported in the engine.

Functions

- static int RandomInt (int min, int max)

  Generating a random integer between 2 integer values.
- static glm::mat3 GenerateCuboidIT (float mass, float ∗extents)

  Generate a cuboid inertia tensor.
- static glm::mat3 GenerateSphereIT (float mass, float radius)

  Generate a sphere inertia tensor.

Variables

- static float **gravity** = 9.81f
- static std::string **projectDir** = std::filesystem::current_path().string()
- static const glm::vec3 **worldUp** = glm::vec3(0.f, 1.f, 0.f)
- static const glm::vec3 **worldSide** = glm::vec3(1.f, 0.f, 0.f)
- static const glm::vec3 **worldForward** = glm::vec3(0.f, 0.f, -1.f)

5.1.1 Detailed Description

DEPRECATED DO NOT USE THIS

5.1.2 Function Documentation

5.1.2.1 GenerateCuboidIT()

```
static glm::mat3 mtrx::GenerateCuboidIT (
            float mass,
            float * extents )  [static]
```

Generate a cuboid inertia tensor.

Parameters

mass	Mass of the cuboid
extents	The extents of the cuboid

Returns

glm::mat3 The inertia tensor generated

5.1.2.2 GenerateSphereIT()

```
static glm::mat3 mtrx::GenerateSphereIT (
            float mass,
            float radius )  [static]
```

Generate a sphere inertia tensor.

Parameters

mass	The mass of the sphere
radius	The radius of the sphere

Returns

glm::mat3 The generated inertia tensor

5.1.2.3 RandomInt()

```
static int mtrx::RandomInt (
            int min,
            int max )  [static]
```

Generating a random integer between 2 integer values.

Parameters

min	Minimum integer value (inclusive)
max	Maximum integer value (exclusive)

Returns

int A random integer within the range of min and max

5.2 mtrx::ColliderDetectionUtil Namespace Reference

Utility namespace used for resolving collision detection algorithms to be used based on collider types.

Functions

- bool Collide (const Collider &collider1, const Collider &collider2)
  Check for whether 2 colliders collide.
- bool SphereCollisionOptions (const SphereCollider &sphCollider, const Collider &collider)
  Collision algorithm options for sphere colliders.
- bool AABBCollisionOptions (const AABBCollider &aabb, const Collider &collider)
  Collision algorithm options for AABB colliders.
- bool OOBBCollisionOptions (const OOBBCollider &oobb, const Collider &collider)
  Collision algorithm options for OOBB colliders.
- bool CapsuleCollisionOptions (const CapsuleCollider &capCollider, const Collider &collider)
  Collision algorithm options for capsule colliders @mtrx::CapsuleCollider.
- bool ConvexShapeCollisionOptions (const ConvexShapeCollider &convexCollider, const Collider &collider)
  Collision algotithm options for convex shape colliders.

5.2.1 Detailed Description

Utility namespace used for resolving collision detection algorithms to be used based on collider types.

See also

mtrx::ColliderType

5.2.2 Function Documentation

5.2.2.1 AABBCollisionOptions()

```
bool mtrx::ColliderDetectionUtil::AABBCollisionOptions (
            const AABBCollider & aabb,
            const Collider & collider )
```

Collision algorithm options for AABB colliders.

See also

mtrx::AABBCollider

Parameters

aabb	The aabb collider
collider	The other collider

Returns

true The 2 colliders collide

false The 2 colliders do not collide

5.2.2.2 CapsuleCollisionOptions()

```
bool mtrx::ColliderDetectionUtil::CapsuleCollisionOptions (
            const CapsuleCollider & capCollider,
            const Collider & collider )
```

Collision algorithm options for capsule colliders @mtrx::CapsuleCollider.

Parameters

capCollider	The capsule collider
collider	The other collider

Returns

true The 2 colliders collide

false The 2 colliders do not collide

5.2.2.3 Collide()

```
bool mtrx::ColliderDetectionUtil::Collide (
            const Collider & collider1,
            const Collider & collider2 )
```

Check for whether 2 colliders collide.

Parameters

collider1	The first collider
collider2	The second collider

Returns

true The 2 colliders collide

false the 2 colliders do not collide

Here is the caller graph for this function:

5.2.2.4 ConvexShapeCollisionOptions()

```
bool mtrx::ColliderDetectionUtil::ConvexShapeCollisionOptions (
            const ConvexShapeCollider & convexCollider,
            const Collider & collider )
```

Collision algotithm options for convex shape colliders.

See also

mtrx::ConvexShapeCollider

Parameters

convexCollider	The convex shape collider
collider	The other collider

Returns

true The 2 colliders collide

false The 2 colliders do not collide

5.2.2.5 OOBBCollisionOptions()

```
bool mtrx::ColliderDetectionUtil::OOBBCollisionOptions (
            const OOBBCollider & oobb,
            const Collider & collider )
```

Collision algorithm options for OOBB colliders.

See also

mtrx::OOBBCollider

Parameters

oobb	The OOBB collider
collider	The other collider

Returns

true The 2 colliders collide

false The 2 colliders do not collide

5.2.2.6 SphereCollisionOptions()

```
bool mtrx::ColliderDetectionUtil::SphereCollisionOptions (
            const SphereCollider & sphCollider,
            const Collider & collider )
```

Collision algorithm options for sphere colliders.

See also

mtrx::SphereCollider

Parameters

sphCollider	The sphere collider
collider	The other collider

Returns

true The 2 colliders collide

false The 2 colliders do not collide

5.3 mtrx::CollisionUtil Namespace Reference

Namespace used to define some collision detection algorithms that are used to check for collision between different colliders. None of these functions require colliders themselves, They use mostly primitive classes instead.

Functions

- bool SphereSphereCollision (const glm::vec3 &center1, const glm::vec3 &center2, float radius1, float radius2)

  Sphere Sphere collision detection algorithm.
- bool SphereCapsuleCollision (const glm::vec3 &center1, const glm::vec3 &center2, float radius1, float radius2, const glm::vec3 &A, const glm::vec3 &B)

  Sphere Capsule collision detection algorithm.
- bool SphereAABBCollision (const glm::vec3 &center, const float radius, const glm::vec3 &center1, const float *halfExtents)

  Sphere AABB collider collision detection algorithm.
- bool SphereOOBBCollision (const glm::vec3 &center, const float radius, const glm::vec3 &center1, const glm::vec3 *axes, const float *halfExtents)

  Sphere OOBB collider collision detection algorithm.
- bool CapsuleCapsuleCollision (const glm::vec3 &A1, const glm::vec3 &B1, const glm::vec3 &A2, const glm::vec3 &B2, float radius1, float radius2)

  Capsule Capsule collision detection algorithm.
- bool CapsuleAABBCollision (const glm::vec3 &A, const glm::vec3 &B, const float radii, const glm::vec3 &center, const float *halfExtents)

  Capsule AABB collision detection algorithm.
- bool AABBCollision (const glm::vec3 &center, const float *halfExtents, const glm::vec3 &center1, const float *halfExtents1)

  AABB AABB collision detection algorithm.
- template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator>::value_type>::value>, typename Iterator1 , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator1>::value_type>::value>>
  bool ConvexShapeCollision (const Iterator &startVertices1, const Iterator &endVertices1, const Iterator1 &startVertices2, const Iterator1 &endVertices2)

  Convex Shape collision detection algorithm. Uses GJK collision detection algorithm.
- template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator>::value_type>::value>>
  bool ConvexShapeCapsuleCollision (const Iterator &startVertices, const Iterator &endVertices, const int size, const glm::vec3 &A, const glm::vec3 &B, const float radii)

  Convex Shape Capsule collision detection algorithm.
- template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator>::value_type>::value>>
  bool CapsuleOOBBCollision (const glm::vec3 &A, const glm::vec3 &B, float radii, const Iterator &startVertices, const Iterator &endVertices, const int size)

  Capsule OOBB Collision detection algorithm.
- template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator>::value_type>::value>, typename Iterator1 , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator1>::value_type>::value>>
  bool AABBOOBBCollision (const Iterator &startVertices1, const Iterator &endVertices1, const Iterator1 &startVertices2, const Iterator1 &endVertices2)

AABB and OOBB collision detection algorithm.

- template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator>::value_type>::value>, typename Iterator1 , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator1>::value_type>::value>>
  bool OOBBCollision (const Iterator &startVertices1, const Iterator &endVertices1, const Iterator1 &startVertices2, const Iterator1 &endVertices2)

  OOBB and OOBB collision detection algorithm.
- template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator>::value_type>::value>>
  bool ConvexShapeSphereCollision (const Iterator &startVertices, const Iterator &endVertices, const int size, const glm::vec3 &center, const float radius)

  Convex Shape Sphere collision detection algorithm.

5.3.1 Detailed Description

Namespace used to define some collision detection algorithms that are used to check for collision between different colliders. None of these functions require colliders themselves, They use mostly primitive classes instead.

5.3.2 Function Documentation

5.3.2.1 AABBCollision()

```
bool mtrx::CollisionUtil::AABBCollision (
            const glm::vec3 & center,
            const float * halfExtents,
            const glm::vec3 & center1,
            const float * halfExtents1 )
```

AABB AABB collision detection algorithm.

Parameters

center	Center of the first AABB collider
halfExtents	Half extents of the first AABB collider
center1	Center of the second AABB collider
halfExtents1	Half extents of the second AABB collider

Returns

true The AABB colliders collide

false The AABB colliders do not collide

5.3.2.2 AABBOOBBCollision()

```
template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename
std::iterator_traits<Iterator>::value_type>::value>, typename Iterator1 , typename = std↩
::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator1>::value_↩
type>::value>>
bool mtrx::CollisionUtil::AABBOOBBCollision (
            const Iterator & startVertices1,
            const Iterator & endVertices1,
            const Iterator1 & startVertices2,
            const Iterator1 & endVertices2 )
```

AABB and OOBB collision detection algorithm.

See also

mtrx::CollisionUtil::ConvexShapeCollision

Template Parameters

Iterator	An iterator type for the data structure that holds the vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator>::value_type>::value> Template parameter check
Iterator1	An iterator type for the data structure that holds the vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator1>::value_type>::value> Template parameter check

Parameters

startVertices1	Iterator to the beginning of the vertex list of the AABB
endVertices1	Iterator to the end of the vertex list of the AABB
startVertices2	Iterator to the beginning of the vertex list of the OOBB
endVertices2	Iterator to the end of the vertex list of the OOBB

Returns

true The AABB and OOBB colliders collide

false The AABB and OOBB colliders do not collide

Here is the call graph for this function:

5.3.2.3 CapsuleAABBCollision()

```
bool mtrx::CollisionUtil::CapsuleAABBCollision (
            const glm::vec3 & A,
            const glm::vec3 & B,
            const float radii,
            const glm::vec3 & center,
            const float * halfExtents )
```

Capsule AABB collision detection algorithm.

Parameters

A	Center of the topmost sphere for capsule
B	Center of the bottommost sphere for capsule
radii	Radii of the capsule
center	Center of the AABB collider
halfExtents	Half extents of the AABB collider

Returns

true The capsule and AABB colliders collide

false The capsule and AABB colliders do not collide

5.3.2.4 CapsuleCapsuleCollision()

```
bool mtrx::CollisionUtil::CapsuleCapsuleCollision (
            const glm::vec3 & A1,
            const glm::vec3 & B1,
            const glm::vec3 & A2,
            const glm::vec3 & B2,
            float radius1,
            float radius2 )
```

Capsule Capsule collision detection algorithm.

Parameters

A1	Center of topmost sphere for first capsule
B1	Center of bottommost sphere for first capsule
A2	Center of topmost sphere for second capsule
B2	Center of bottommost sphere for second capsule
radius1	Radius of first capsule
radius2	Radius of second capsule

Returns

true Capsule colliders collide

false Capsule colliders do not collide

5.3.2.5 CapsuleOOBBCollision()

```
template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename
std::iterator_traits<Iterator>::value_type>::value>>
bool mtrx::CollisionUtil::CapsuleOOBBCollision (
```

```
            const glm::vec3 & A,
            const glm::vec3 & B,
            float radii,
            const Iterator & startVertices,
            const Iterator & endVertices,
            const int size )
```

Capsule OOBB Collision detection algorithm.

Template Parameters

Iterator	An iterator type for the data structure that holds the vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator>::value_type>::value> Template parameter check

Parameters

A	Center of topmost sphere of capsule collider
B	Center of bottommost sphere of capsule collider
radii	Radii of the capsule collider
startVertices	Iterator to the beginning of the vertices list
endVertices	Iterator to the end of the vertices list
size	Number of vertices

Returns

true The capsule and OOBB colliders collide

false The capsule and OOBB colliders do not collide

Here is the call graph for this function:

5.3.2.6 ConvexShapeCapsuleCollision()

```
template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename
std::iterator_traits<Iterator>::value_type>::value>>
bool mtrx::CollisionUtil::ConvexShapeCapsuleCollision (
            const Iterator & startVertices,
            const Iterator & endVertices,
            const int size,
            const glm::vec3 & A,
            const glm::vec3 & B,
            const float radii )
```

Convex Shape Capsule collision detection algorithm.

Template Parameters

Iterator	An iterator type for the data structure that holds the vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator>::value_type>::value> Template parameter check

Parameters

startVertices	Iterator to the beginning of the vertex list
endVertices	Iterator to the end of the vertex list
size	Number of vertices
A	Center of the topmost sphere in the capsule collider
B	Center of the bottommost sphere in the capsule collider
radii	Radii of the capsule collider

Returns

true The convex shape and capsule colliders collide

false The convex shape and capsule colliders collide

Here is the call graph for this function: Here is the caller graph for this function:

5.3.2.7 ConvexShapeCollision()

```
template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename
std::iterator_traits<Iterator>::value_type>::value>, typename Iterator1 , typename = std↩
::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator1>::value_↩
type>::value>>
bool mtrx::CollisionUtil::ConvexShapeCollision (
            const Iterator & startVertices1,
            const Iterator & endVertices1,
            const Iterator1 & startVertices2,
            const Iterator1 & endVertices2 )
```

Convex Shape collision detection algorithm. Uses GJK collision detection algorithm.

See also

mtrx::GJKUtil

Template Parameters

Iterator	An iterator type for the data structure that holds the vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator>::value_type>::value> Template parameter check
Iterator1	An iterator type for the data structure that holds the vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator1>::value_type>::value> Template parameter check

Parameters

startVertices1	Iterator to the beginning of the vertex list for first convex collider
endVertices1	Iterator to the end of the vertex list for first convex collider
startVertices2	Iterator to the beginning of the vertex list for the second convex shape collider

Parameters

endVertices2	Iterator to the end of the vertex list for the second convex shape collider

Returns

true The convex shape colliders collide

false The convex shape colliders do not collide

Here is the call graph for this function: Here is the caller graph for this function:

5.3.2.8 ConvexShapeSphereCollision()

```
template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename
std::iterator_traits<Iterator>::value_type>::value>>
bool mtrx::CollisionUtil::ConvexShapeSphereCollision (
            const Iterator & startVertices,
            const Iterator & endVertices,
            const int size,
            const glm::vec3 & center,
            const float radius )
```

Convex Shape Sphere collision detection algorithm.

Template Parameters

Iterator	An iterator type for the data structure that holds the vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator>::value_type>::value> Template parameter check

Parameters

startVertices	Iterator to the beginning of the vertex list of the convex shape
endVertices	Iterator to the end of the vertex list of the convex shape
size	Number of vertices of convex shape collider
center	Center of the sphere collider
radius	Radius of the sphere collider

Returns

true The convex shape and sphere colliders collide

false The convex shape and sphere colliders do not collide

Here is the call graph for this function:

5.3.2.9 OOBBCollision()

```
template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename
std::iterator_traits<Iterator>::value_type>::value>, typename Iterator1 , typename = std↩
```

```
::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator1>::value_↩
type>::value>>
bool mtrx::CollisionUtil::OOBBCollision (
            const Iterator & startVertices1,
            const Iterator & endVertices1,
            const Iterator1 & startVertices2,
            const Iterator1 & endVertices2 )
```

OOBB and OOBB collision detection algorithm.

See also

mtrx::CollisionUtil::ConvexShapeCollision

Template Parameters

Iterator	An iterator type for the data structure that holds the vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator>::value_type>::value> Template parameter check
Iterator1	An iterator type for the data structure that holds the vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator1>::value_type>::value> Template parameter check

Parameters

startVertices1	Iterator to the beginning of the vertex list of the first OOBB
endVertices1	Iterator to the end of the vertex list of the first OOBB
startVertices2	Iterator to the beginning of the vertex list of the second OOBB
endVertices2	Iterator to the end of the vertex list of the second OOBB

Returns

true The OOBB colliders collide

false The OOBB colliders do not collide

Here is the call graph for this function:

5.3.2.10 SphereAABBCollision()

```
bool mtrx::CollisionUtil::SphereAABBCollision (
            const glm::vec3 & center,
            const float radius,
            const glm::vec3 & center1,
            const float * halfExtents )
```

Sphere AABB collider collision detection algorithm.

Parameters

center	Center of the sphere collider
radius	Radius of the sphere collider
center1	Center of the AABB collider
halfExtents	Half Extents of the AABB collider

Returns

true The sphere and AABB colliders collide

false The sphere and AABB colliders do not collide

5.3.2.11 SphereCapsuleCollision()

```
bool mtrx::CollisionUtil::SphereCapsuleCollision (
            const glm::vec3 & center1,
            const glm::vec3 & center2,
            float radius1,
            float radius2,
            const glm::vec3 & A,
            const glm::vec3 & B )
```

Sphere Capsule collision detection algorithm.

Parameters

center1	Center of the sphere collider
center2	Center of the capsule collider
radius1	Radius of the sphere collider
radius2	Radius of the capsule collider
A	Topmost sphere center for capsule collider
B	Bottommost sphere center for capsule collider

Returns

true The sphere and capsule colliders collide

false The sphere and capsule colliders do not collide

5.3.2.12 SphereOOBBCollision()

```
bool mtrx::CollisionUtil::SphereOOBBCollision (
            const glm::vec3 & center,
            const float radius,
            const glm::vec3 & center1,
```

```
const glm::vec3 * axes,
const float * halfExtents )
```

Sphere OOBB collider collision detection algorithm.

Parameters

center	Center of the sphere collider
radius	Radius of the sphere collider
center1	Center of the OOBB collider
axes	Axes that define the OOBB world
halfExtents	Half extent values of OOBB colliders

Returns

true Sphere and OOBB colliders collide

false Sphere and OOBB colliders do not collide

5.3.2.13 SphereSphereCollision()

```
bool mtrx::CollisionUtil::SphereSphereCollision (
            const glm::vec3 & center1,
            const glm::vec3 & center2,
            float radius1,
            float radius2 )
```

Sphere Sphere collision detection algorithm.

Parameters

center1	Center of the first sphere collider
center2	Center of the second sphere collider
radius1	Radius of the first sphere collider
radius2	Radius of the second sphere collider

Returns

true The 2 sphere colliders collide

false The 2 sphere colliders do not collide

5.4 mtrx::PhysicsUtil Namespace Reference

Utility namespace for some basic calculations that are useful for collision detection and other physics related functionality.

Functions

- glm::vec3 TripleCross (const glm::vec3 &a, const glm::vec3 &b, const glm::vec3 &c)

  Implementation of triple cross product.

- float MinDistanceTwoLines (const glm::vec3 &A, const glm::vec3 &B, const glm::vec3 &C, const glm::vec3 &D)

  Minimum distance squared between 2 infinite line. The square of the minimum distance is used for optimization purposes.
- float MinDistanceSquaredTwoSegments (const glm::vec3 &A, const glm::vec3 &B, const glm::vec3 &C, const glm::vec3 &D)

  Minimum distance squared between 2 line segments. The square of the minimum distance is used for optimization purposes http://geomalgorithms.com/a07-_distance.html#dist3D_Segment_to_Segment.
- std::pair< float, glm::vec3 > MinDistanceSquaredPointRay (const glm::vec3 &point, const glm::vec3 &start↩ PointRay, const glm::vec3 &rayDirection)

  Minimum distance squared of a point and ray. The square of the minimum distance is used for optimization purposes.
- std::pair< float, glm::vec3 > MinDistanceSquaredPointSegment (const glm::vec3 &A, const glm::vec3 &B, const glm::vec3 &C)

  Minimum distance squared of a point and a line segment. The square of the minimum distance is used for optimization purposes.
- float MinDistanceSquaredLineSegmentRay (const glm::vec3 &a, const glm::vec3 &b, const glm::vec3 &ray↩ Start, const glm::vec3 &rayDirection)

  Minimum distance squared Line segment and ray. Can be broken down into a line segment minimum distance squared.
- float MinDistanceSquaredPointTriangle (const glm::vec3 &pt, const glm::vec3 &a, const glm::vec3 &b, const glm::vec3 &c)

  Minimum distance squared between a point and triangle. The square of the minimum distance is used for optimization purposes.
- float MinDistanceSquaredLineSegmentTriangle (const glm::vec3 &a, const glm::vec3 &b, const glm::vec3 &c, const glm::vec3 &d, const glm::vec3 &e)

  Minimum distance squared of line segment and triangle. The square of the minimum distance is used for optimization purposes.
- std::pair< float, glm::vec3 > MinDistanceSquaredPointAABB (const glm::vec3 &pt, const glm::vec3 &center, const float *halfExtents)

  Minimum distance squared between a point and an AABB. The square of the minimum distance is used for optimization purposes.
- glm::quat Slerp (const glm::quat &firstRotation, const glm::quat &secondRotation, float t)

  Implementation of Slerp which is a lerp for rotations.
- glm::vec3 Lerp (const glm::vec3 &startingPosition, const glm::vec3 &destination, float t)

  Implementation of linear interpolation.
- float Ease (float t)

  Easing function for the t parameter to be used for lerping functions. The easing function is a half sin curve that allows for a slow start and a slow end.
- template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator>::value↩ _type>::value>>
  std::vector< Triangle > TriangulateConvexShape (const Iterator &startVertices, const Iterator &endVertices, const int size)

  Triangulating a convex shape by fan triangulation which is using the first vertex as a starting point for all triangules and fanning out to the other vertices.

5.4.1 Detailed Description

Utility namespace for some basic calculations that are useful for collision detection and other physics related functionality.

5.4.2 Function Documentation

5.4.2.1 Ease()

```
float mtrx::PhysicsUtil::Ease (
            float t )
```

Easing function for the t parameter to be used for lerping functions. The easing function is a half sin curve that allows for a slow start and a slow end.

Parameters

t	Parameter between [0, 1]

Returns

float New t value according to the easing function

5.4.2.2 Lerp()

```
glm::vec3 mtrx::PhysicsUtil::Lerp (
            const glm::vec3 & startingPosition,
            const glm::vec3 & destination,
            float t )
```

Implementation of linear interpolation.

Parameters

startingPosition	Start position
destination	End position
t	Parameter between [0, 1]

Returns

glm::vec3 A vector between the 2 vectors influenced by the t value

5.4.2.3 MinDistanceSquaredLineSegmentRay()

```
float mtrx::PhysicsUtil::MinDistanceSquaredLineSegmentRay (
            const glm::vec3 & a,
            const glm::vec3 & b,
            const glm::vec3 & rayStart,
            const glm::vec3 & rayDirection )
```

Minimum distance squared Line segment and ray. Can be broken down into a line segment minimum distance squared.

See also

mtrx::PhysicsUtil::MinimumDistanceSquaredTwoSegments The square of the minimum distance is used for optimization purposes

Parameters

a	The first end point of the line segment
b	The second end point of the line segment
rayStart	The starting point of the ray
rayDirection	The normalized direction of the ray

Returns

float The minimum distance squared of the line segment and the ray

5.4.2.4 MinDistanceSquaredLineSegmentTriangle()

```
float mtrx::PhysicsUtil::MinDistanceSquaredLineSegmentTriangle (
            const glm::vec3 & a,
            const glm::vec3 & b,
            const glm::vec3 & c,
            const glm::vec3 & d,
            const glm::vec3 & e )
```

Minimum distance squared of line segment and triangle. The square of the minimum distance is used for optimization purposes.

Parameters

a	The first end point of the line segment
b	The second end point of the line segment
c	First point of the triangle
d	Second point of the triangle
e	Third point of the triangle

Returns

float The minimum distance squared between the line segment and the triangle

Here is the caller graph for this function:

5.4.2.5 MinDistanceSquaredPointAABB()

```
std::pair<float, glm::vec3> mtrx::PhysicsUtil::MinDistanceSquaredPointAABB (
            const glm::vec3 & pt,
            const glm::vec3 & center,
            const float * halfExtents )
```

Minimum distance squared between a point and an AABB. The square of the minimum distance is used for optimization purposes.

Parameters

pt	The point we want to check the distance for
center	The center of the AABB collider

See also

mtrx::AABBCollider

Parameters

halfExtents	The half extents of the AABBCollider

See also

mtrx::AABBCollider

Returns

std::pair<float, glm::vec3> The minimum distance squared and the closest point

5.4.2.6 MinDistanceSquaredPointRay()

```
std::pair<float, glm::vec3> mtrx::PhysicsUtil::MinDistanceSquaredPointRay (
            const glm::vec3 & point,
            const glm::vec3 & startPointRay,
            const glm::vec3 & rayDirection )
```

Minimum distance squared of a point and ray. The square of the minimum distance is used for optimization purposes.

Parameters

point	A point in 3d space we want to check the distance for
startPointRay	Start point of the ray
rayDirection	The normalized direction of the ray

Returns

std::pair<float, glm::vec3> The minimum distance squared and the closest point

5.4.2.7 MinDistanceSquaredPointSegment()

```
std::pair<float, glm::vec3> mtrx::PhysicsUtil::MinDistanceSquaredPointSegment (
            const glm::vec3 & A,
```

```
            const glm::vec3 & B,
            const glm::vec3 & C )
```

Minimum distance squared of a point and a line segment. The square of the minimum distance is used for optimization purposes.

Parameters

A	The point we want to check the distance for
B	The first end point of the line segment
C	The second end point of the line segment

Returns

std::pair<float, glm::vec3> The minimum distance squared and the closest point

5.4.2.8 MinDistanceSquaredPointTriangle()

```
float mtrx::PhysicsUtil::MinDistanceSquaredPointTriangle (
            const glm::vec3 & pt,
            const glm::vec3 & a,
            const glm::vec3 & b,
            const glm::vec3 & c )
```

Minimum distance squared between a point and triangle. The square of the minimum distance is used for optimization purposes.

Parameters

pt	The point we want to check the distance for
a	The first point of the triangle
b	The second point of the triangle
c	The third point of the triangle

Returns

float The minimum distance squared

Here is the caller graph for this function:

5.4.2.9 MinDistanceSquaredTwoSegments()

```
float mtrx::PhysicsUtil::MinDistanceSquaredTwoSegments (
            const glm::vec3 & A,
            const glm::vec3 & B,
            const glm::vec3 & C,
            const glm::vec3 & D )
```

Minimum distance squared between 2 line segments. The square of the minimum distance is used for optimization purposes http://geomalgorithms.com/a07-_distance.html#dist3D_Segment_to_Segment.

Parameters

A	First end of the line segment
B	Second end of the line segment
C	First end of the second line segment
D	Second end of the second line segment

Returns

float The minimum distance squared between 2 line segments

5.4.2.10 MinDistanceTwoLines()

```
float mtrx::PhysicsUtil::MinDistanceTwoLines (
            const glm::vec3 & A,
            const glm::vec3 & B,
            const glm::vec3 & C,
            const glm::vec3 & D )
```

Minimum distance squared between 2 infinite line. The square of the minimum distance is used for optimization purposes.

Parameters

A	Point on first line
B	Point on first line
C	Point on second line
D	Point on second line

Returns

float The minimum distance squared between the 2 infinite lines

5.4.2.11 Slerp()

```
glm::quat mtrx::PhysicsUtil::Slerp (
            const glm::quat & firstRotation,
            const glm::quat & secondRotation,
            float t )
```

Implementation of Slerp which is a lerp for rotations.

Parameters

firstRotation	start position of the rotation
secondRotation	End position of the rotation
t	parameter between [0, 1] to interpolate between the 2 rotations

Returns

glm::quat The rotation interpolated between the 2 rotations

5.4.2.12 TriangulateConvexShape()

```
template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename
std::iterator_traits<Iterator>::value_type>::value>>
std::vector<Triangle> mtrx::PhysicsUtil::TriangulateConvexShape (
            const Iterator & startVertices,
            const Iterator & endVertices,
            const int size )
```

Triangulating a convex shape by fan triangulation which is using the first vertex as a starting point for all triangules and fanning out to the other vertices.

Template Parameters

Iterator	Iterator type that is used by the collection of vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator>::value_type>::value> Iterator template check

Parameters

startVertices	Iterator to the beginning of the vertex list
endVertices	Iterator to the end of the vertex list
size	The size of the vertex list

Returns

std::vector<Triangle> A vector of triangles representing the triangulated mesh

Here is the caller graph for this function:

5.4.2.13 TripleCross()

```
glm::vec3 mtrx::PhysicsUtil::TripleCross (
            const glm::vec3 & a,
            const glm::vec3 & b,
            const glm::vec3 & c )
```

Implementation of triple cross product.

Parameters

a	First vector
b	Second vector
c	Third vector

Returns

glm::vec3 The result of the triple cross product

Here is the caller graph for this function:

5.5 mtrx::RaycastCollisionUtil Namespace Reference

Utility namespace used for raycasting functionality.

Functions

- bool RaySphereCollision (const glm::vec3 &sphereCenter, float sphereRadius, const glm::vec3 &startPoint↩ Ray, const glm::vec3 &rayDirection)

  Raycasting onto a sphere collider.
- bool RayCapsuleCollision (const glm::vec3 &startPositionRay, const glm::vec3 &direction, const glm::vec3 &A, const glm::vec3 &B, float capsRadius)

  Raycasting onto a capsule collider.
- bool RayBoxCollision (const glm::vec3 &rayStart, const glm::vec3 &rayDirection, const glm::vec3 &center, const glm::vec3 *axes, const float *halfExtents)

  Raycasting onto a box collider.
- Collider * RaycastFiltered (const std::map< int, Collider * > &colliders, const std::vector< Collider * > &filterColliders, const glm::vec3 &rayStartPosition, const glm::vec3 &rayDirection)

  NOT TESTED OR USED.
- Collider * RaycastUnfiltered (const std::map< int, Collider * > &colliders, const glm::vec3 &rayStartPosition, const glm::vec3 &rayDirection)

  NOT TESTED OR USED.

5.5.1 Detailed Description

Utility namespace used for raycasting functionality.

5.5.2 Function Documentation

5.5.2.1 RayBoxCollision()

```
bool mtrx::RaycastCollisionUtil::RayBoxCollision (
            const glm::vec3 & rayStart,
            const glm::vec3 & rayDirection,
            const glm::vec3 & center,
            const glm::vec3 * axes,
            const float * halfExtents )
```

Raycasting onto a box collider.

See also

mtrx::OOBBCollider

mtrx::AABBCollider

mtrx::Ray

Parameters

rayStart	The starting point of the ray
rayDirection	The normalized direction of the ray
center	The center of the box collider
axes	The axes of the collider
halfExtents	the half extents of the collider

Returns

true The ray and box collide

false The ray and box do not collide

Here is the caller graph for this function:

5.5.2.2 RayCapsuleCollision()

```
bool mtrx::RaycastCollisionUtil::RayCapsuleCollision (
            const glm::vec3 & startPositionRay,
            const glm::vec3 & direction,
            const glm::vec3 & A,
            const glm::vec3 & B,
            float capsRadius )
```

Raycasting onto a capsule collider.

See also

mtrx::CapsuleCollider

mtrx::Ray

Parameters

startPositionRay	The starting position of the ray
direction	The normalized direction of the ray
A	The center of the topmost sphere of the capsule
B	The center of the bottommost sphere of the capsule
capsRadius	The radius of the capsule

Returns

true The ray and the capsule collider collide

false The ray and the capsule collider do not collide

Here is the caller graph for this function:

5.5.2.3 RaycastFiltered()

```
Collider* mtrx::RaycastCollisionUtil::RaycastFiltered (
            const std::map< int, Collider * > & colliders,
            const std::vector< Collider * > & filterColliders,
            const glm::vec3 & rayStartPosition,
            const glm::vec3 & rayDirection )
```

NOT TESTED OR USED.

Parameters

colliders	
filterColliders	
rayStartPosition	
rayDirection	

Returns

Collider*

5.5.2.4 RaycastUnfiltered()

```
Collider* mtrx::RaycastCollisionUtil::RaycastUnfiltered (
            const std::map< int, Collider * > & colliders,
            const glm::vec3 & rayStartPosition,
            const glm::vec3 & rayDirection )
```

NOT TESTED OR USED.

Parameters

colliders	
rayStartPosition	
rayDirection	

Returns

Collider*

5.5.2.5 RaySphereCollision()

```
bool mtrx::RaycastCollisionUtil::RaySphereCollision (
            const glm::vec3 & sphereCenter,
            float sphereRadius,
            const glm::vec3 & startPointRay,
            const glm::vec3 & rayDirection )
```

Raycasting onto a sphere collider.

See also

mtrx::SphereCollider

mtrx::Ray

Parameters

sphereCenter	Center of the sphere collider
sphereRadius	Radius of the sphere collider
startPointRay	Start point of the ray
rayDirection	Normalized direction of the ray

Returns

true The ray and the sphere collider collide

false The ray and the sphere collider do not collide

Here is the caller graph for this function:

Chapter 6

Class Documentation

6.1 mtrx::AABBCollider Class Reference

Implmentation of an Axis Aligned Bounding Box used for collision systems.

`#include <AABBCollider.h>`

Inheritance diagram for mtrx::AABBCollider:

Collaboration diagram for mtrx::AABBCollider:

Public Member Functions

- AABBCollider (const glm::vec3 &center=glm::vec3(), const glm::vec3 &scale=glm::vec3(1, 1, 1))

  Construct a new AABBCollider object.
- virtual ~AABBCollider ()=default

  Destroy the AABBCollider object.
- virtual bool RaycastCollision (const Ray &ray) override

  Collision function between a ray and an AABB.
- virtual void SetOrientation (const glm::quat &orientation) override

  This function should not be used and is empty as AABBs should not change orientations.
- virtual float GetSize () override

  Calculate the size of an AABB defined as length ∗ width ∗ height.
- const glm::vec3 ∗ GetAxes () const

  Get the Axes that define the AABB's world They would constant in this case since the orientation never changes.
- const float ∗ GetHalfExtents () const

  Get the Half Extents of the AABB (This is the const version)
- float ∗ GetHalfExtents ()
- virtual void SetScale (const glm::vec3 &scale) override

  Set the Scale object overriden to modify the halfExtents to represent scale change.

Private Attributes

- float **halfExtents** [3]
- ObjectAxes **axes**

Additional Inherited Members

6.1.1 Detailed Description

Implmentation of an Axis Aligned Bounding Box used for collision systems.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 AABBCollider()

```
mtrx::AABBCollider::AABBCollider (
            const glm::vec3 & center = glm::vec3(),
            const glm::vec3 & scale = glm::vec3(1, 1, 1) )
```

Construct a new AABBCollider object.

Parameters

center	Center point of the AABB
scale	Scale vector of the AABB

See also

mtrx::Transform

6.1.2.2 ~AABBCollider()

```
virtual mtrx::AABBCollider::~AABBCollider ( )  [virtual], [default]
```

Destroy the AABBCollider object.

6.1.3 Member Function Documentation

6.1.3.1 GetAxes()

```
const glm::vec3* mtrx::AABBCollider::GetAxes ( ) const  [inline]
```

Get the Axes that define the AABB's world They would constant in this case since the orientation never changes.

Returns

const glm::vec3* pointer to the array of 3 axes

6.1.3.2 GetHalfExtents() [1/2]

```
float* mtrx::AABBCollider::GetHalfExtents ( )  [inline]
```

Non const version of GetHalfExtents()

Returns

float* pointer to the array of halfExtents

6.1.3.3 GetHalfExtents() [2/2]

```
const float* mtrx::AABBCollider::GetHalfExtents ( ) const  [inline]
```

Get the Half Extents of the AABB (This is the const version)

Returns

const float* pointer to the array of half extents values in each dimension

6.1.3.4 GetSize()

```
virtual float mtrx::AABBCollider::GetSize ( )  [inline], [override], [virtual]
```

Calculate the size of an AABB defined as length * width * height.

Implements mtrx::IBoundingVolume.

6.1.3.5 RaycastCollision()

```
virtual bool mtrx::AABBCollider::RaycastCollision (
            const Ray & ray )  [inline], [override], [virtual]
```

Collision function between a ray and an AABB.

Parameters

ray	The ray that we will be checking collision on

See also

mtrx::ray

Returns

true If there is a collision

false If there is no collision

Reimplemented from mtrx::ConvexShapeCollider.

Here is the call graph for this function:

6.1.3.6 SetOrientation()

```
virtual void mtrx::AABBCollider::SetOrientation (
            const glm::quat & orientation )  [inline], [override], [virtual]
```

This function should not be used and is empty as AABBs should not change orientations.

Parameters

orientation	

Reimplemented from mtrx::ConvexShapeCollider.

6.1.3.7 SetScale()

```
virtual void mtrx::AABBCollider::SetScale (
            const glm::vec3 & scale )  [inline], [override], [virtual]
```

Set the Scale object overriden to modify the halfExtents to represent scale change.

Parameters

scale	The new scale vector of the AABB

Reimplemented from mtrx::ConvexShapeCollider.

Here is the call graph for this function:

6.2 mtrx::Body Class Reference

Implementation of a basic particle body.

```
#include <Body.h>
```

Inheritance diagram for mtrx::Body:

Collaboration diagram for mtrx::Body:

Public Member Functions

- Body (const glm::vec3 &position=glm::vec3(), const glm::quat &orientation=glm::quat(), const glm::vec3 &scale=glm::vec3(), const float mass=MAX_MASS)

  Construct a new Body object.
- virtual ~Body ()=default

  Destroy the Body object.
- void AddForce (const glm::vec3 &force)

  Adding a force vector at the center of gravity of the body.
- virtual void Integrate (float deltaTime)=0

  Integrating the forces applied to the body.
- virtual void ClearAccumulators ()=0

  Clearing the accumulators of the body.
- void SetInverseMass (const float inverseMass)

  Set the Inverse Mass of the body.
- void SetPosition (const glm::vec3 &position)

  Set the Position of the body.
- void SetVelocity (const glm::vec3 &velocity)

  Set the Velocity of the body.
- void SetAcceleration (const glm::vec3 &acceleration)

  Set the Acceleration of the body.
- void SetLinearDamping (const float damping)

  Set the Linear Damping of the body.
- void SetMass (const float mass)

  Set the mass of the body.
- glm::quat & GetOrientation ()

  Get the Orientation of the body.
- bool GetIsInfiniteMass ()

  Checking whether we this body is of infinite mass by checking that the inverse mass is 0.
- float GetInverseMass () const

  Get the Inverse Mass of the body.
- float GetDamping () const

  Get the linear damping of the body.
- glm::vec3 & GetPosition ()

  Get the Position of the body.
- glm::vec3 & GetVelocity ()

  Get the Velocity of the body.
- glm::vec3 & GetAcceleration ()

  Get the Acceleration of the body.
- glm::vec3 & GetAccumForces ()

  Get the accumulated forces applied on the body.
- Transform & GetTransform ()

  Get the Transform of the body.
- float GetMass () const

  Get the Mass of the body.

Protected Attributes

- Transform **transform**
- glm::vec3 **velocity**
- glm::vec3 **acceleration**
- glm::vec3 **accumForces**
- float **linearDamping**
- float **inverseMass**

Additional Inherited Members

6.2.1 Detailed Description

Implementation of a basic particle body.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Body()

```
mtrx::Body::Body (
            const glm::vec3 & position = glm::vec3(),
            const glm::quat & orientation = glm::quat(),
            const glm::vec3 & scale = glm::vec3(),
            const float mass = MAX_MASS )
```

Construct a new Body object.

Parameters

position	The position of the body
orientation	Orientation of the body
scale	Scale of the body
mass	Mass of the body

6.2.2.2 ~Body()

```
virtual mtrx::Body::~Body ( )  [virtual], [default]
```

Destroy the Body object.

6.2.3 Member Function Documentation

6.2.3.1 AddForce()

```
void mtrx::Body::AddForce (
            const glm::vec3 & force )  [inline]
```

Adding a force vector at the center of gravity of the body.

Parameters

force	The force vector that we will be added to the body's force accumulator

6.2.3.2 ClearAccumulators()

```
virtual void mtrx::Body::ClearAccumulators ( )  [pure virtual]
```

Clearing the accumulators of the body.

Implemented in mtrx::Rigidbody.

6.2.3.3 GetAcceleration()

```
glm::vec3& mtrx::Body::GetAcceleration ( )  [inline]
```

Get the Acceleration of the body.

Returns

glm::vec3& The value of the acceleration

6.2.3.4 GetAccumForces()

```
glm::vec3& mtrx::Body::GetAccumForces ( )  [inline]
```

Get the accumulated forces applied on the body.

Returns

glm::vec3& The value of the accumulated forces

6.2.3.5 GetDamping()

```
float mtrx::Body::GetDamping ( ) const  [inline]
```

Get the linear damping of the body.

Returns

float The value of the linear damping

6.2.3.6 GetInverseMass()

```
float mtrx::Body::GetInverseMass ( ) const  [inline]
```

Get the Inverse Mass of the body.

Returns

float The value of the inverse mass of the body

6.2.3.7 GetIsInfiniteMass()

```
bool mtrx::Body::GetIsInfiniteMass ( )  [inline]
```

Checking whether we this body is of infinite mass by checking that the inverse mass is 0.

Returns

true The body is of infinite mass

false The body is not of infinite mass

6.2.3.8 GetMass()

```
float mtrx::Body::GetMass ( ) const
```

Get the Mass of the body.

Returns

float The mass of the body

6.2.3.9 GetOrientation()

```
glm::quat& mtrx::Body::GetOrientation ( )  [inline]
```

Get the Orientation of the body.

Returns

glm::quat& The quaternion orientation value

Here is the call graph for this function:

6.2.3.10 GetPosition()

```
glm::vec3& mtrx::Body::GetPosition ( )  [inline]
```

Get the Position of the body.

Returns

glm::vec3& The position value of the body

Here is the call graph for this function:

6.2.3.11 GetTransform()

```
Transform& mtrx::Body::GetTransform ( )  [inline]
```

Get the Transform of the body.

Returns

Transform& The transform of the body

6.2.3.12 GetVelocity()

```
glm::vec3& mtrx::Body::GetVelocity ( )  [inline]
```

Get the Velocity of the body.

Returns

glm::vec3& The value of the velocity

6.2.3.13 Integrate()

```
virtual void mtrx::Body::Integrate (
            float deltaTime )  [pure virtual]
```

Integrating the forces applied to the body.

Parameters

deltaTime	The time elapsed between frames

Implements mtrx::IIntegratable.

Implemented in mtrx::Rigidbody.

6.2.3.14 SetAcceleration()

```
void mtrx::Body::SetAcceleration (
            const glm::vec3 & acceleration )  [inline]
```

Set the Acceleration of the body.

Parameters

acceleration	The new acceleration value of the body

6.2.3.15 SetInverseMass()

```
void mtrx::Body::SetInverseMass (
            const float inverseMass )  [inline]
```

Set the Inverse Mass of the body.

Parameters

inverseMass	The new value of the inverse mass

6.2.3.16 SetLinearDamping()

```
void mtrx::Body::SetLinearDamping (
            const float damping )  [inline]
```

Set the Linear Damping of the body.

Parameters

damping	the new linear damping value

6.2.3.17 SetMass()

```
void mtrx::Body::SetMass (
            const float mass )
```

Set the mass of the body.

Parameters

mass	The new value of the mass

6.2.3.18 SetPosition()

```
void mtrx::Body::SetPosition (
            const glm::vec3 & position )  [inline]
```

Set the Position of the body.

Parameters

position	The new position of the body

Here is the call graph for this function:

6.2.3.19 SetVelocity()

```
void mtrx::Body::SetVelocity (
            const glm::vec3 & velocity )  [inline]
```

Set the Velocity of the body.

Parameters

velocity	The new velocity value of the body

6.3 mtrx::BVHNode< BoundingVolume > Class Template Reference

An implementation of a BVH node used for Bounding Volume Hierarchies.

`#include <BVHNode.h>`

Collaboration diagram for mtrx::BVHNode< BoundingVolume >:

Public Member Functions

- BVHNode ()

  Construct a new BVHNode object.
- BVHNode (BVHNode< BoundingVolume > *parent, BoundingVolume &volume, Body *body=nullptr)

  Construct a new BVHNode object.
- ~BVHNode ()

Destroy the BVHNode object.

- bool IsLeaf ()

  Check whether this node is a leaf.
- bool IsCollision (BVHNode< BoundingVolume > &other)

  Check whether 2 nodes collide by checking that the bounding volumes collide.
- bool DescendA (BVHNode< BoundingVolume > &a, BVHNode< BoundingVolume > &b)

  Utility function that allows that helps with figuring out which direction in the structure we want to go down when trying to find potential contacts.
- void GetPotentialContacts (std::list< PotentialCollision > &potentialCollisions)

  Get the Potential Contacts that could occur from the hierarchy.
- void Insert (Body *body, Collider &collider)

  Insert collider as a leaf of the stucture. We attempt to add it in a position in the tree where we would get the least amount of growth.
- void RecalculateBoundingVolume ()

  Recalculate the bounding volumes after the structure has been modified.

Public Attributes

- BVHNode< BoundingVolume > * **parent**
- BVHNode< BoundingVolume > * **children** [2]
- BoundingVolume **volume**
- Body * **body**

6.3.1 Detailed Description

template<class BoundingVolume>
class mtrx::BVHNode< BoundingVolume >

An implementation of a BVH node used for Bounding Volume Hierarchies.

Template Parameters

BoundingVolume	The type of bounding volume that will be used within the hierarchy

6.3.2 Constructor & Destructor Documentation

6.3.2.1 BVHNode() [1/2]

```
template<class BoundingVolume>
mtrx::BVHNode< BoundingVolume >::BVHNode ( )
```

Construct a new BVHNode object.

6.3.2.2 BVHNode() [2/2]

```
template<class BoundingVolume>
mtrx::BVHNode< BoundingVolume >::BVHNode (
            BVHNode< BoundingVolume > * parent,
            BoundingVolume & volume,
            Body * body = nullptr )
```

Construct a new BVHNode object.

Parameters

parent	Parent of this node
volume	The bounding volume/collider used
body	The body that the BVHNode represents

See also

mtrx::Body

6.3.2.3 ~BVHNode()

```
template<class BoundingVolume>
mtrx::BVHNode< BoundingVolume >::~BVHNode ( )
```

Destroy the BVHNode object.

6.3.3 Member Function Documentation

6.3.3.1 DescendA()

```
template<class BoundingVolume>
bool mtrx::BVHNode< BoundingVolume >::DescendA (
            BVHNode< BoundingVolume > & a,
            BVHNode< BoundingVolume > & b )  [inline]
```

Utility function that allows that helps with figuring out which direction in the structure we want to go down when trying to find potential contacts.

Parameters

a	
b	

Returns

true

false

Here is the call graph for this function:

6.3.3.2 GetPotentialContacts()

```
template<class BoundingVolume>
void mtrx::BVHNode< BoundingVolume >::GetPotentialContacts (
            std::list< PotentialCollision > & potentialCollisions )
```

Get the Potential Contacts that could occur from the hierarchy.

Parameters

potentialCollisions	Structure to store the potential collsions

See also

mtrx::PotentialCollision

6.3.3.3 Insert()

```
template<class BoundingVolume>
void mtrx::BVHNode< BoundingVolume >::Insert (
            Body * body,
            Collider & collider )
```

Insert collider as a leaf of the stucture. We attempt to add it in a position in the tree where we would get the least amount of growth.

Parameters

body	The body that we want to add
collider	The bounding volume that we want to add

6.3.3.4 IsCollision()

```
template<class BoundingVolume>
bool mtrx::BVHNode< BoundingVolume >::IsCollision (
            BVHNode< BoundingVolume > & other )  [inline]
```

Check whether 2 nodes collide by checking that the bounding volumes collide.

Parameters

other	The BVHNode that we want to check collision with

Returns

true The 2 nodes collide

false The 2 nodes do not collide

6.3.3.5 IsLeaf()

```
template<class BoundingVolume>
bool mtrx::BVHNode< BoundingVolume >::IsLeaf ( )  [inline]
```

Check whether this node is a leaf.

Returns

true This node is a leaf

false This node is not a leaf and has at least one child

Here is the caller graph for this function:

6.3.3.6 RecalculateBoundingVolume()

```
template<class BoundingVolume>
void mtrx::BVHNode< BoundingVolume >::RecalculateBoundingVolume ( )
```

Recalculate the bounding volumes after the structure has been modified.

6.4 mtrx::CapsuleCollider Class Reference

Implementation of Capsule colliders.

```
#include <CapsuleCollider.h>
```

Inheritance diagram for mtrx::CapsuleCollider:

Collaboration diagram for mtrx::CapsuleCollider:

Public Member Functions

- CapsuleCollider (const glm::vec3 &center=glm::vec3(), const glm::quat &orientation=glm::angleAxis(0.f, worldUp), const glm::vec3 &scale=glm::vec3(1, 1, 1), float radii=0.5f, float height=1.f)

  Construct a new Capsule Collider object.
- CapsuleCollider (const Transform &transform=Transform(), float radii=0.5f, float height=1.f)

  Construct a new Capsule Collider object.
- virtual ~CapsuleCollider ()=default

  Destroy the Capsule Collider object.
- virtual bool RaycastCollision (const Ray &ray) override

  Raycast ray and capsule collider collision detection algorithm.
- float GetRadii () const

  Get the Radii of the capsule colldier.
- float GetHeight () const

  Get the Height of the capsule collider.
- void SetRadii (float radius)

  Set the Radii of the capsule collider.
- void SetHeight (float height)

  Set the Height of the capsule collider.
- virtual void SetScale (const glm::vec3 &scale) override

  Set the Scale of the capsule collider Only the X and Y values of the scale are the only applicable values on the collider The Z value is ignored.
- virtual void SetPosition (const glm::vec3 &position) override

  Set the Position of the capsile collider.
- virtual void SetOrientation (const glm::quat &orientation) override

  Set the Orientation of the capsule collider.

Public Attributes

- glm::vec3 **A**
- glm::vec3 **B**

Private Attributes

- float **radii**
- float **height**

Additional Inherited Members

6.4.1 Detailed Description

Implementation of Capsule colliders.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 CapsuleCollider() [1/2]

```
mtrx::CapsuleCollider::CapsuleCollider (
            const glm::vec3 & center = glm::vec3(),
            const glm::quat & orientation = glm::angleAxis(0.f, worldUp),
            const glm::vec3 & scale = glm::vec3(1, 1, 1),
            float radii = 0.5f,
            float height = 1.f )
```

Construct a new Capsule Collider object.

Parameters

center	Center of the capsule collider
orientation	Orientation of the capsule collider
scale	Scale of the capsule collider
radii	Radii of the spheres of the capsule collider
height	Height of the capsule collider

6.4.2.2 CapsuleCollider() [2/2]

```
mtrx::CapsuleCollider::CapsuleCollider (
            const Transform & transform = Transform(),
            float radii = 0.5f,
            float height = 1.f )
```

Construct a new Capsule Collider object.

Parameters

transform	The transform that the capsule collider will use

See also

mtrx::Transform

Parameters

radii	Radii of the spheres of the capsule collider
height	Height of the capsule collider

6.4.2.3 ~CapsuleCollider()

```
virtual mtrx::CapsuleCollider::~CapsuleCollider ( )  [virtual], [default]
```

Destroy the Capsule Collider object.

6.4.3 Member Function Documentation

6.4.3.1 GetHeight()

```
float mtrx::CapsuleCollider::GetHeight ( ) const  [inline]
```

Get the Height of the capsule collider.

Returns

float The height of the capsule collider

6.4.3.2 GetRadii()

```
float mtrx::CapsuleCollider::GetRadii ( ) const  [inline]
```

Get the Radii of the capsule colldier.

Returns

float The radii of the capsule collider

6.4.3.3 RaycastCollision()

```
virtual bool mtrx::CapsuleCollider::RaycastCollision (
            const Ray & ray )  [inline], [override], [virtual]
```

Raycast ray and capsule collider collision detection algorithm.

Parameters

ray	Ray used for raycast check

See also

mtrx::Ray

Returns

true The ray and capsule collider collide

false the ray and the capsule collider do not collide

Implements mtrx::Collider.

Here is the call graph for this function:

6.4.3.4 SetHeight()

```
void mtrx::CapsuleCollider::SetHeight (
            float height )  [inline]
```

Set the Height of the capsule collider.

Parameters

height	The new value of the height

Here is the call graph for this function:

6.4.3.5 SetOrientation()

```
virtual void mtrx::CapsuleCollider::SetOrientation (
            const glm::quat & orientation )  [inline], [override], [virtual]
```

Set the Orientation of the capsule collider.

Parameters

orientation	The new orientation of the capsule collider

Reimplemented from mtrx::Collider.

Here is the call graph for this function:

6.4.3.6 SetPosition()

```
virtual void mtrx::CapsuleCollider::SetPosition (
            const glm::vec3 & position )  [inline], [override], [virtual]
```

Set the Position of the capsile collider.

Parameters

position	The new position of the capsule collider

Reimplemented from mtrx::Collider.

Here is the call graph for this function:

6.4.3.7 SetRadii()

```
void mtrx::CapsuleCollider::SetRadii (
            float radius )  [inline]
```

Set the Radii of the capsule collider.

Parameters

radius	The new value of radius

Here is the call graph for this function:

6.4.3.8 SetScale()

```
virtual void mtrx::CapsuleCollider::SetScale (
            const glm::vec3 & scale )  [inline], [override], [virtual]
```

Set the Scale of the capsule collider Only the X and Y values of the scale are the only applicable values on the collider The Z value is ignored.

Parameters

scale	The new scale of the collider

Reimplemented from mtrx::Collider.

Here is the call graph for this function:

6.5 mtrx::Collider Class Reference

Basic API for all colliders.

`#include <Collider.h>`

Inheritance diagram for mtrx::Collider:

Collaboration diagram for mtrx::Collider:

Public Member Functions

- Collider (const ColliderType &colliderType, const glm::vec3 &center=glm::vec3(), const glm::quat &orientation=glm::angleAxis(0.f, worldUp), const glm::vec3 &scale=glm::vec3(1.f, 1.f, 1.f), bool isConvex=false)

  Construct a new Collider object.
- Collider (const ColliderType &colliderType, const Transform &transform, bool isConvex=false)

  Construct a new Collider object.
- virtual ~Collider ()=default

Destroy the Collider object.

- virtual bool RaycastCollision (const Ray &ray)=0

  Raycasting function that all colliders should have.
- bool CheckCollision (const Collider &collider)

  Check for collision with another collider.
- const glm::vec3 & GetPosition () const

  Get the Position of the collider.
- const glm::vec3 & GetScale () const

  Get the Scale of the collider.
- const glm::quat & GetOrientation () const

  Get the Orientation of the collider.
- const ColliderType & GetColliderType () const

  Get the Collider Type of collider.
- const int GetColliderId () const

  Get the Collider Id of the collider.
- const bool IsConvex () const

  Is this collider convex.
- const glm::vec3 GetForward () const

  Get the Forward axis vector.
- const glm::vec3 GetSide () const

  Get the Side axis vector.
- const glm::vec3 GetUp () const

  Get the Up axis vector.
- virtual void SetPosition (const glm::vec3 &center)

  Set the Position of the collider.
- virtual void SetScale (const glm::vec3 &scale)

  Set the Scale of the collider.
- virtual void SetOrientation (const glm::quat &orientation)

  Set the Orientation of the collider.

Protected Attributes

- int **colliderId**
- ColliderType **type**
- bool **isConvexShape**
- Transform **transform**

Static Private Attributes

- static int **id**

6.5.1 Detailed Description

Basic API for all colliders.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Collider() [1/2]

```
mtrx::Collider::Collider (
            const ColliderType & colliderType,
            const glm::vec3 & center = glm::vec3(),
            const glm::quat & orientation = glm::angleAxis(0.f, worldUp),
            const glm::vec3 & scale = glm::vec3(1.f, 1.f, 1.f),
            bool isConvex = false )
```

Construct a new Collider object.

Parameters

colliderType	The type of the collider
center	Center of the collider
orientation	Orientation of the collider
scale	The scale of the collider
isConvex	Whether collider is a convex shape collider

6.5.2.2 Collider() [2/2]

```
mtrx::Collider::Collider (
            const ColliderType & colliderType,
            const Transform & transform,
            bool isConvex = false )
```

Construct a new Collider object.

Parameters

colliderType	The type of the collider
transform	The transform of the collider

See also

mtrx::Transform

Parameters

isConvex	Whether the collider is a convex shape collider

6.5.2.3 ~Collider()

```
virtual mtrx::Collider::~Collider ( )  [virtual], [default]
```

Destroy the Collider object.

6.5.3 Member Function Documentation

6.5.3.1 CheckCollision()

```
bool mtrx::Collider::CheckCollision (
            const Collider & collider )  [inline]
```

Check for collision with another collider.

Parameters

collider	The collider that we want to check for collision

Returns

true The collider is colliding with this collider

false The collider is not colliding with this collider

Here is the call graph for this function:

6.5.3.2 GetColliderId()

```
const int mtrx::Collider::GetColliderId ( ) const  [inline]
```

Get the Collider Id of the collider.

Returns

const int Value of the Id of the collider

6.5.3.3 GetColliderType()

```
const ColliderType& mtrx::Collider::GetColliderType ( ) const  [inline]
```

Get the Collider Type of collider.

Returns

const ColliderType& Enum collider type value

6.5.3.4 GetForward()

```
const glm::vec3 mtrx::Collider::GetForward ( ) const  [inline]
```

Get the Forward axis vector.

Returns

const glm::vec3 The forward vector of this collider

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.5 GetOrientation()

```
const glm::quat& mtrx::Collider::GetOrientation ( ) const  [inline]
```

Get the Orientation of the collider.

Returns

const glm::quat& The current orientation of the collider

Here is the call graph for this function:

6.5.3.6 GetPosition()

```
const glm::vec3& mtrx::Collider::GetPosition ( ) const  [inline]
```

Get the Position of the collider.

Returns

const glm::vec3& current position of the collider

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.7 GetScale()

```
const glm::vec3& mtrx::Collider::GetScale ( ) const  [inline]
```

Get the Scale of the collider.

Returns

const glm::vec3& The current scale of the collider

Here is the call graph for this function:

6.5.3.8 GetSide()

```
const glm::vec3 mtrx::Collider::GetSide ( ) const  [inline]
```

Get the Side axis vector.

Returns

const glm::vec3 The side axis vector of the collider

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.9 GetUp()

```
const glm::vec3 mtrx::Collider::GetUp ( ) const  [inline]
```

Get the Up axis vector.

Returns

const glm::vec3 the up axis vector of the collider

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.10 IsConvex()

```
const bool mtrx::Collider::IsConvex ( ) const  [inline]
```

Is this collider convex.

Returns

true The collider is a convex shape

false The collider is not a convex shape

6.5.3.11 RaycastCollision()

```
virtual bool mtrx::Collider::RaycastCollision (
            const Ray & ray )  [pure virtual]
```

Raycasting function that all colliders should have.

Parameters

ray	Ray that we want to cast

See also

mtrx::Ray

Returns

true The ray and the collider collide

false The ray and the collider do not collide

Implemented in mtrx::ConvexShapeCollider, mtrx::SphereCollider, mtrx::CapsuleCollider, mtrx::OOBBCollider, and mtrx::AABBCollider.

6.5.3.12 SetOrientation()

```
virtual void mtrx::Collider::SetOrientation (
            const glm::quat & orientation )  [inline], [virtual]
```

Set the Orientation of the collider.

Parameters

orientation	The new orientation of the collider

Reimplemented in mtrx::ConvexShapeCollider, mtrx::CapsuleCollider, and mtrx::AABBCollider.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.13 SetPosition()

```
virtual void mtrx::Collider::SetPosition (
            const glm::vec3 & center )  [inline], [virtual]
```

Set the Position of the collider.

Parameters

center	The center position of this collider

Reimplemented in mtrx::CapsuleCollider, and mtrx::ConvexShapeCollider.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.14 SetScale()

```
virtual void mtrx::Collider::SetScale (
            const glm::vec3 & scale )  [inline], [virtual]
```

Set the Scale of the collider.

Parameters

scale	The new scale of the collider

Reimplemented in mtrx::SphereCollider, mtrx::CapsuleCollider, mtrx::ConvexShapeCollider, mtrx::AABBCollider, and mtrx::OOBBCollider.

Here is the call graph for this function: Here is the caller graph for this function:

6.6 mtrx::CollisionSystem Class Reference

The collision detection system of the engine.

`#include <CollisionSystem.h>`

Collaboration diagram for mtrx::CollisionSystem:

Public Member Functions

- CollisionSystem ()

  Construct a new Collision System object.

Public Attributes

- std::unordered_set< Collider * > **colliders**

6.6.1 Detailed Description

The collision detection system of the engine.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 CollisionSystem()

`mtrx::CollisionSystem::CollisionSystem ( )`

Construct a new Collision System object.

6.7 mtrx::ConvexShapeCollider Class Reference

Implementation of a Convex Shape collider.

`#include <ConvexShapeCollider.h>`

Inheritance diagram for mtrx::ConvexShapeCollider:

Collaboration diagram for mtrx::ConvexShapeCollider:

Public Member Functions

- ConvexShapeCollider (const ColliderType &colliderType=ColliderType::ConvexShape, const glm::vec3 &center=glm::vec3(), const glm::quat &orientation=glm::angleAxis(0.f, worldUp), const glm::vec3 &scale=glm::vec3(1, 1, 1))

  Construct a new Convex Shape Collider object.
- ConvexShapeCollider (const ColliderType &colliderType=ColliderType::ConvexShape, const Transform &transform=Transform())

  Construct a new Convex Shape Collider object.
- ConvexShapeCollider (const ColliderType &colliderType=ColliderType::ConvexShape, const std::vector< glm::vec3 * > &vertices=std::vector< glm::vec3 * >(), const glm::vec3 &center=glm::vec3(), const glm::quat &orientation=glm::angleAxis(0.f, glm::vec3(0, 1, 0)), const glm::vec3 &scale=glm::vec3(1, 1, 1))

  Construct a new Convex Shape Collider object.
- ConvexShapeCollider (const ColliderType &colliderType=ColliderType::ConvexShape, const std::vector< glm::vec3 * > &vertices=std::vector< glm::vec3 * >(), const Transform &transform=Transform())

  Construct a new Convex Shape Collider object.
- virtual ~ConvexShapeCollider ()

  Destroy the Convex Shape Collider object.
- virtual bool RaycastCollision (const Ray &ray) override

  Raycast collision check with convex shape colliders.
- std::vector< glm::vec3 * > * GetVertices () const

  Get the Vertices of the convex shape collider.
- glm::mat4 GetModelMatrix () const

  Get the Model Matrix of the convex shape collider.
- virtual void SetPosition (const glm::vec3 &pos) override

  Set the Position of the convex shape collider.
- virtual void SetScale (const glm::vec3 &scale) override

  Set the Scale of the collider.
- virtual void SetOrientation (const glm::quat &quat) override

  Set the Orientation of the collider.

Protected Attributes

- std::vector< glm::vec3 * > **vertices**
- std::vector< glm::vec3 * > **transformedVertices**
- bool **transformModified**

6.7.1 Detailed Description

Implementation of a Convex Shape collider.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 ConvexShapeCollider() [1/4]

```
mtrx::ConvexShapeCollider::ConvexShapeCollider (
            const ColliderType & colliderType = ColliderType::ConvexShape,
            const glm::vec3 & center = glm::vec3(),
            const glm::quat & orientation = glm::angleAxis(0.f, worldUp),
            const glm::vec3 & scale = glm::vec3(1, 1, 1) )
```

Construct a new Convex Shape Collider object.

Parameters

colliderType	The type of collider we are creating. The default is a convex shape however we can have other type

See also

mtrx::AABBCollider

mtrx::OOBBCollider

Parameters

center	The center of the collider
orientation	The orientation of the collider
scale	The scale of the collider

6.7.2.2 ConvexShapeCollider() [2/4]

```
mtrx::ConvexShapeCollider::ConvexShapeCollider (
            const ColliderType & colliderType = ColliderType::ConvexShape,
            const Transform & transform = Transform() )
```

Construct a new Convex Shape Collider object.

Parameters

colliderType	The type of collider we are creating. The default is a convex shape however we can have other type

See also

mtrx::AABBCollider

mtrx::OOBBCollider

Parameters

transform	The transform of the collider

See also

mtrx::Transform

6.7.2.3 ConvexShapeCollider() [3/4]

```
mtrx::ConvexShapeCollider::ConvexShapeCollider (
            const ColliderType & colliderType = ColliderType::ConvexShape,
            const std::vector< glm::vec3 * > & vertices = std::vector< glm::vec3 * >(),
            const glm::vec3 & center = glm::vec3(),
            const glm::quat & orientation = glm::angleAxis(0.f, glm::vec3(0, 1, 0)),
            const glm::vec3 & scale = glm::vec3(1, 1, 1) )
```

Construct a new Convex Shape Collider object.

Parameters

colliderType	The type of collider we are creating. The default is a convex shape however we can have other type

See also

mtrx::AABBCollider

mtrx::OOBBCollider

Parameters

vertices	The vertices of the convex shape collider
center	The center of the collider
orientation	The orientation of the collider
scale	The scale of the collider

6.7.2.4 ConvexShapeCollider() [4/4]

```
mtrx::ConvexShapeCollider::ConvexShapeCollider (
            const ColliderType & colliderType = ColliderType::ConvexShape,
            const std::vector< glm::vec3 * > & vertices = std::vector< glm::vec3 * >(),
            const Transform & transform = Transform() )
```

Construct a new Convex Shape Collider object.

Parameters

colliderType	The type of collider we are creating. The default is a convex shape however we can have other type

See also

mtrx::AABBCollider

mtrx::OOBBCollider

Parameters

vertices	The vertices of the convex shape collider
transform	The transform of the collider

See also

mtrx::Trnasform

6.7.2.5 ~ConvexShapeCollider()

```
virtual mtrx::ConvexShapeCollider::~ConvexShapeCollider ( )  [virtual]
```

Destroy the Convex Shape Collider object.

6.7.3 Member Function Documentation

6.7.3.1 GetModelMatrix()

```
glm::mat4 mtrx::ConvexShapeCollider::GetModelMatrix ( ) const
```

Get the Model Matrix of the convex shape collider.

Returns

glm::mat4 The model matrix of the convex shape collider

6.7.3.2 GetVertices()

```
std::vector<glm::vec3*>* mtrx::ConvexShapeCollider::GetVertices ( ) const
```

Get the Vertices of the convex shape collider.

Returns

std::vector<glm::vec3*>* A pointer to the transformed vertices of the collider

6.7.3.3 RaycastCollision()

```
virtual bool mtrx::ConvexShapeCollider::RaycastCollision (
            const Ray & ray )  [override], [virtual]
```

Raycast collision check with convex shape colliders.

Parameters

ray	The ray that we want to cast

Returns

true The ray and the colliders collide

false The ray and colliders do not collide

Implements mtrx::Collider.

Reimplemented in mtrx::OOBBCollider, and mtrx::AABBCollider.

6.7.3.4 SetOrientation()

```
virtual void mtrx::ConvexShapeCollider::SetOrientation (
            const glm::quat & quat )  [inline], [override], [virtual]
```

Set the Orientation of the collider.

Parameters

quat	The new orientation of the collider

Reimplemented from mtrx::Collider.

Reimplemented in mtrx::AABBCollider.

Here is the call graph for this function:

6.7.3.5 SetPosition()

```
virtual void mtrx::ConvexShapeCollider::SetPosition (
            const glm::vec3 & pos )  [inline], [override], [virtual]
```

Set the Position of the convex shape collider.

Parameters

pos	The new position of the collider

Reimplemented from mtrx::Collider.

Here is the call graph for this function:

6.7.3.6 SetScale()

```
virtual void mtrx::ConvexShapeCollider::SetScale (
            const glm::vec3 & scale )  [inline], [override], [virtual]
```

Set the Scale of the collider.

Parameters

scale	The new scale of the collider

Reimplemented from mtrx::Collider.

Reimplemented in mtrx::AABBCollider, and mtrx::OOBBCollider.

Here is the call graph for this function:

6.8 mtrx::GameTime Class Reference

Basic implementation of time values mostly used for the calculation of delta time.

`#include <GameTime.h>`

Collaboration diagram for mtrx::GameTime:

Static Public Member Functions

- static double FindTimeDiffSeconds (const std::chrono::high_resolution_clock::time_point &t1, const std::chrono::high_resolution_clock::time_point &t2)
- static void Init ()
- static void Update ()
- static float GetTimeSeconds ()
- static std::chrono::high_resolution_clock::time_point GetTime ()

Static Public Attributes

- static float **deltaTime**

Static Private Member Functions

- static void CalculateDeltaTime ()

Static Private Attributes

- static const std::chrono::high_resolution_clock::time_point **startTime**
- static std::chrono::high_resolution_clock::time_point **currentTime**
- static std::chrono::high_resolution_clock::time_point **prevCurrentTime**

6.8.1 Detailed Description

Basic implementation of time values mostly used for the calculation of delta time.

6.8.2 Member Function Documentation

6.8.2.1 CalculateDeltaTime()

```
static void mtrx::GameTime::CalculateDeltaTime ( )  [inline], [static], [private]
```

Calculate delta time Here is the call graph for this function: Here is the caller graph for this function:

6.8.2.2 FindTimeDiffSeconds()

```
static double mtrx::GameTime::FindTimeDiffSeconds (
            const std::chrono::high_resolution_clock::time_point & t1,
            const std::chrono::high_resolution_clock::time_point & t2 )  [inline], [static]
```

Returns

The time difference between 2 timepoints in seconds its a double for more precision

Here is the caller graph for this function:

6.8.2.3 GetTime()

```
static std::chrono::high_resolution_clock::time_point mtrx::GameTime::GetTime ( )  [inline],
[static]
```

Returns

Wrapper for getting the current time

Here is the caller graph for this function:

6.8.2.4 GetTimeSeconds()

```
static float mtrx::GameTime::GetTimeSeconds ( )  [inline], [static]
```

Returns

time elapsed since the start of the application

Here is the call graph for this function:

6.8.2.5 Init()

```
static void mtrx::GameTime::Init ( )  [inline], [static]
```

Initialization code Here is the call graph for this function:

6.8.2.6 Update()

```
static void mtrx::GameTime::Update ( )  [inline], [static]
```

Update time values for calculating delta time Here is the call graph for this function:

6.9 mtrx::GJKUtil Class Reference

Utility class for the implementation of the GJK algorithm used for collision detection between convex shape colliders.

```
#include <GJKUtil.h>
```

Collaboration diagram for mtrx::GJKUtil:

Static Public Member Functions

- template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator>::value_type>::value>, typename Iterator1 , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator1>::value_type>::value>>
  static bool Collision (const Iterator &startVertices1, const Iterator &endVertices1, const Iterator1 &startVertices2, const Iterator1 &endVertices2)

  Implementation of GJK Collision detection algorithm.

Static Private Member Functions

- static bool UpdateSimplex (Simplex &simplex, glm::vec3 &direction, glm::vec3 &a)

  Update the simplex values or return a collision.
- static bool TriangleSimplexUpdate (Simplex &simplex, glm::vec3 &direction, glm::vec3 &a)

  Updating the simplex values from a triangle simplex.
- static bool TetrahedronSimplexUpdate (Simplex &simplex, glm::vec3 &direction, glm::vec3 &a)

  Update the simplex based upon a tetrahedron simplex.
- static void TetrahedronChecks (Simplex &simplex, glm::vec3 &AO, glm::vec3 &AB, glm::vec3 &AC, glm::vec3 &ABC, glm::vec3 &direction, glm::vec3 &a)

  Utility function for updating tetrahedron simplex knowing that there is no collision.
- template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator>::value_type>::value>, typename Iterator1 , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator1>::value_type>::value>>
  static glm::vec3 Support (const Iterator &startVertices1, const Iterator &endVertices1, const Iterator1 &startVertices2, const Iterator1 &endVertices2, const glm::vec3 &direction)

  Support functionality for convex shape collider gives us a point in Minkowski space that is closest to the direction vector given.
- template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator>::value_type>::value>>
  static glm::vec3 * FarthestPointInDirection (const Iterator &startVertices, const Iterator &endVertices, const glm::vec3 &direction)

  Get a point in the vertex list that is closest (direction-wise) to the direction vector.

6.9.1 Detailed Description

Utility class for the implementation of the GJK algorithm used for collision detection between convex shape colliders.

See also

mtrx::ConvexShapeCollider

Some resources on the algorithm:

http://www.dyn4j.org/2010/04/gjk-gilbert-johnson-keerthi/#gjk-minkowski http://vec3.ca/

6.9.2 Member Function Documentation

6.9.2.1 Collision()

```
template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename
std::iterator_traits<Iterator>::value_type>::value>, typename Iterator1 , typename = std↩
::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator1>::value_↩
type>::value>>
static bool mtrx::GJKUtil::Collision (
            const Iterator & startVertices1,
            const Iterator & endVertices1,
            const Iterator1 & startVertices2,
            const Iterator1 & endVertices2 )  [inline], [static]
```

Implementation of GJK Collision detection algorithm.

See also

mtrx::ConvexShapeCollider

Template Parameters

Iterator	The type of iterator we will use for vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator>::value_type>::value> Iterator template checks
Iterator1	The type of iterator we will use for vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator1>::value_type>::value> Iterator template checks

Parameters

startVertices1	Iterator to the beginning of the first vertex list for first convex shape
endVertices1	Iterator to the end of the first vertex list of the first convex shape
startVertices2	Iterator to the beginning of the second vertex list for the second convex shape
endVertices2	Iterator to the end of the second vertex list of the second convex shape

Returns

true The 2 convex shape colliders collide

false The 2 convex shape colliders do not collide

Here is the call graph for this function: Here is the caller graph for this function:

6.9.2.2 FarthestPointInDirection()

```
template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename
std::iterator_traits<Iterator>::value_type>::value>>
static glm::vec3* mtrx::GJKUtil::FarthestPointInDirection (
            const Iterator & startVertices,
            const Iterator & endVertices,
            const glm::vec3 & direction )  [inline], [static], [private]
```

Get a point in the vertex list that is closest (direction-wise) to the direction vector.

Template Parameters

Iterator	The type of iterator we will use for vertices
std::enable_if_t<std::is_same<glm::vec3,typename*	std::iterator_traits<Iterator>::value_type>::value> Iterator template checks

Parameters

startVertices	Iterator to the beginning of the vertex list
endVertices	Iterator to the end of the vertex list
direction	The direction vector used to find the farthest point

Returns

glm::vec3∗ The farthest point with the give direction

Here is the caller graph for this function:

6.9.2.3 Support()

```
template<typename Iterator , typename = std::enable_if_t<std::is_same<glm::vec3*, typename
std::iterator_traits<Iterator>::value_type>::value>, typename Iterator1 , typename = std↩
::enable_if_t<std::is_same<glm::vec3*, typename std::iterator_traits<Iterator1>::value_↩
type>::value>>
static glm::vec3 mtrx::GJKUtil::Support (
            const Iterator & startVertices1,
            const Iterator & endVertices1,
            const Iterator1 & startVertices2,
            const Iterator1 & endVertices2,
            const glm::vec3 & direction )  [inline], [static], [private]
```

Support functionality for convex shape collider gives us a point in Minkowski space that is closest to the direction vector given.

Template Parameters

Iterator	The type of iterator we will use for vertices
std::enable_if_t<std::is_same<glm::vec3∗,typename	std::iterator_traits<Iterator>::value_type>::value> Iterator template checks
Iterator1	The type of iterator we will use for vertices
std::enable_if_t<std::is_same<glm::vec3∗,typename	std::iterator_traits<Iterator1>::value_type>::value> Iterator template checks

Parameters

startVertices1	Iterator to the beginning of the first vertex list for first convex shape
endVertices1	Iterator to the end of the first vertex list of the first convex shape
startVertices2	Iterator to the beginning of the second vertex list for the second convex shape
endVertices2	Iterator to the end of the second vertex list of the second convex shape
direction	The direction that we want to find a vertex in the direction of

Returns

glm::vec3 A vertex in minkowski space in the direction given to be added to the simplex

Here is the call graph for this function: Here is the caller graph for this function:

6.9.2.4 TetrahedronChecks()

```
static void mtrx::GJKUtil::TetrahedronChecks (
            Simplex & simplex,
```

```
            glm::vec3 & AO,
            glm::vec3 & AB,
            glm::vec3 & AC,
            glm::vec3 & ABC,
            glm::vec3 & direction,
            glm::vec3 & a )  [static], [private]
```

Utility function for updating tetrahedron simplex knowing that there is no collision.

Parameters

simplex	The simplex that we want to modify
AO	Line of the triangle 'AO'
AB	Line of the triangle 'AB'
AC	Line of the triangle 'AC'
ABC	normal of the the plane ABC of the triangle
direction	The direction vector of the search
a	The new vector from the support functionality

See also

mtrx::GJKUtil::Support

6.9.2.5 TetrahedronSimplexUpdate()

```
static bool mtrx::GJKUtil::TetrahedronSimplexUpdate (
            Simplex & simplex,
            glm::vec3 & direction,
            glm::vec3 & a )  [static], [private]
```

Update the simplex based upon a tetrahedron simplex.

Parameters

simplex	The simplex we want to update
direction	The direction that we are searching with
a	The new vertex generated from the support function

See also

mtrx::GJKUtil::Support

Returns

true The tetrahedron encapsulate the origin signifying a collision

false The tetrahedron does not encapsulate the origin and thus the simplex needs to be modified or recalculated

6.9.2.6 TriangleSimplexUpdate()

```
static bool mtrx::GJKUtil::TriangleSimplexUpdate (
            Simplex & simplex,
            glm::vec3 & direction,
            glm::vec3 & a )  [static], [private]
```

Updating the simplex values from a triangle simplex.

Parameters

simplex	The simplex we want to update
direction	The direction for the search
a	The new vector that we got from the support function

See also

mtrx::GJKUtil::Support

Returns

true N/A as a triangle simplex cannot encapsulate a point in 3d space

false Always returned

6.9.2.7 UpdateSimplex()

```
static bool mtrx::GJKUtil::UpdateSimplex (
            Simplex & simplex,
            glm::vec3 & direction,
            glm::vec3 & a )  [static], [private]
```

Update the simplex values or return a collision.

Parameters

simplex	The simplex generated to be modified according the the new vertices
direction	The direction that we will be finding points using
a	The newest point to be added to the simplex

Returns

true The simplex encapsulates the origin and thus there exists a collision

false The simplex does not encapsule the origin and thus modified the simplex to continue search

Here is the caller graph for this function:

6.10 mtrx::IBoundingVolume Class Reference

Interface used to define a bounding volume that can be used in a Bounding volume Hierarchy.

`#include <IBoundingVolume.h>`

Inheritance diagram for mtrx::IBoundingVolume:

Collaboration diagram for mtrx::IBoundingVolume:

Public Member Functions

- virtual float **GetSize** ()=0

6.10.1 Detailed Description

Interface used to define a bounding volume that can be used in a Bounding volume Hierarchy.

See also

mtrx::BVHNode PS: This infrastructure is being reworked

6.11 mtrx::IIntegratable Class Reference

Interface for integration PS: Not used that much really.

`#include <IIntegratable.h>`

Inheritance diagram for mtrx::IIntegratable:

Collaboration diagram for mtrx::IIntegratable:

Public Member Functions

- virtual void Integrate (float deltaTime)=0
  Integration functionality.

6.11.1 Detailed Description

Interface for integration PS: Not used that much really.

6.11.2 Member Function Documentation

6.11.2.1 Integrate()

```
virtual void mtrx::IIntegratable::Integrate (
            float deltaTime )  [pure virtual]
```

Integration functionality.

Parameters

deltaTime	Time elapsed from previous frame

Implemented in mtrx::Body, and mtrx::Rigidbody.

6.12 mtrx::IRigidbodyForceGenerator Class Reference

Force generation interface used to apply forces on rigidbodies.

`#include <IRigidbodyForceGenerator.h>`

Inheritance diagram for mtrx::IRigidbodyForceGenerator:

Collaboration diagram for mtrx::IRigidbodyForceGenerator:

Public Member Functions

- virtual void UpdateForces (Rigidbody *rb, float deltaTime)=0
  Apply a certain force onto the given rigidbodies.

6.12.1 Detailed Description

Force generation interface used to apply forces on rigidbodies.

6.12.2 Member Function Documentation

6.12.2.1 UpdateForces()

```
virtual void mtrx::IRigidbodyForceGenerator::UpdateForces (
            Rigidbody * rb,
            float deltaTime )  [pure virtual]
```

Apply a certain force onto the given rigidbodies.

Parameters

rb	The rigidbody we want to apply the force onto
deltaTime	The time elapsed since the previous frame

Implemented in mtrx::rb_BuoyancyForceGenerator, and mtrx::rb_GravityForceGenerator.

6.13 mtrx::LogManager Class Reference

Logger wrapper for logging functionality in the engine.

`#include <LogManager.h>`

Collaboration diagram for mtrx::LogManager:

Public Member Functions

- template<typename T >
  void warn (const T &msg)
  Wrapper for logging a warning.
- template<typename T >
  void info (const T &msg)
  Wrapper for logging an info level message.
- template<typename T >
  void trace (const T &msg)
  Wrapper for logging a trace level message.
- template<typename T >
  void error (const T &msg)
  Wrapper for logging an error level message.
- template<typename T >
  void critical (const T &msg)
  Wrapper for logging a critical level message.

Static Public Member Functions

- static LogManager & GetInstance ()
  Get the singleton logger instance.

Private Member Functions

- void CreateLogDirectory ()
  Create The log directory that we will use for logging if it does not exist.
- LogManager ()
  Construct a new Log Manager object. This object is a singleton so it will be constructed once.
- **LogManager** (const LogManager &)=delete
- **LogManager** (const LogManager &&)=delete
- LogManager & **operator=** (const LogManager &)=delete
- LogManager & **operator=** (const LogManager &&)=delete

Private Attributes

- std::shared_ptr< spdlog::logger > **mtrxLogger**

6.13.1 Detailed Description

Logger wrapper for logging functionality in the engine.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 LogManager()

```
mtrx::LogManager::LogManager ( )  [private]
```

Construct a new Log Manager object. This object is a singleton so it will be constructed once.

6.13.3 Member Function Documentation

6.13.3.1 CreateLogDirectory()

```
void mtrx::LogManager::CreateLogDirectory ( )  [private]
```

Create The log directory that we will use for logging if it does not exist.

6.13.3.2 critical()

```
template<typename T >
void mtrx::LogManager::critical (
            const T & msg )  [inline]
```

Wrapper for logging a critical level message.

Template Parameters

T	Type of warning that we will be logging

Parameters

msg	The message that we log

6.13.3.3 error()

```
template<typename T >
void mtrx::LogManager::error (
            const T & msg )  [inline]
```

Wrapper for logging an error level message.

Template Parameters

T	Type of warning that we will be logging

Parameters

msg	The message that we log

6.13.3.4 GetInstance()

```
static LogManager& mtrx::LogManager::GetInstance ( )  [inline], [static]
```

Get the singleton logger instance.

Returns

LogManager& The instance of the logger that we will be using

6.13.3.5 info()

```
template<typename T >
void mtrx::LogManager::info (
            const T & msg )  [inline]
```

Wrapper for logging an info level message.

Template Parameters

T	Type of warning that we will be logging

Parameters

msg	The message that we log

6.13.3.6 trace()

```
template<typename T >
void mtrx::LogManager::trace (
            const T & msg )  [inline]
```

Wrapper for logging a trace level message.

Template Parameters

T	Type of warning that we will be logging

Parameters

msg	The message that we log

6.13.3.7 warn()

```
template<typename T >
void mtrx::LogManager::warn (
             const T & msg )  [inline]
```

Wrapper for logging a warning.

Template Parameters

T	Type of warning that we will be logging

Parameters

msg	The message that we log

6.14 mtrx::mtrxDynamicWorld Class Reference

Entry point of the user to the engine. This class defines the API that user will be using to interface with the engine.

`#include <mtrxDynamicWorld.h>`

Collaboration diagram for mtrx::mtrxDynamicWorld:

Public Member Functions

- mtrxDynamicWorld ()

  Construct a new mtrx Dynamic World object.
- ~mtrxDynamicWorld ()=default

  Destroy the mtrx Dynamic World object.
- void Update (float dt)

  Update the game world.
- void AddRigidbody (Rigidbody *rb)

  Add a rigidbody to the rigidbody manager.
- void RemoveRigidbody (Rigidbody *rb)

Remove a rigidbody from the rigidbody manager.

- void AddForceGenerator (Rigidbody *rb, const std::shared_ptr< IRigidbodyForceGenerator > &forceGenerator)

  Add a force generator to the rigidbody manager.
- void RemoveForceGenerator (Rigidbody *rb, const std::shared_ptr< IRigidbodyForceGenerator > &generator)

  Remove a rigidbody from the rigidbody manager.
- void AddCollider (Collider *col)

  Add a collider to the collision system.
- void RemoveCollider (Collider *col)

  Remove a collider to the collision system.

Public Attributes

- float **accumulator**

Private Attributes

- RigidbodyManager **m_rbManager**
- CollisionSystem **m_CollisionSystem**

6.14.1 Detailed Description

Entry point of the user to the engine. This class defines the API that user will be using to interface with the engine.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 mtrxDynamicWorld()

```
mtrx::mtrxDynamicWorld::mtrxDynamicWorld ( )
```

Construct a new mtrx Dynamic World object.

6.14.2.2 ~mtrxDynamicWorld()

```
mtrx::mtrxDynamicWorld::~mtrxDynamicWorld ( )  [default]
```

Destroy the mtrx Dynamic World object.

6.14.3 Member Function Documentation

6.14.3.1 AddCollider()

```
void mtrx::mtrxDynamicWorld::AddCollider (
            Collider * col )  [inline]
```

Add a collider to the collision system.

Parameters

col	The collider that we want to add

6.14.3.2 AddForceGenerator()

```
void mtrx::mtrxDynamicWorld::AddForceGenerator (
            Rigidbody * rb,
            const std::shared_ptr< IRigidbodyForceGenerator > & forceGenerator )  [inline]
```

Add a force generator to the rigidbody manager.

Parameters

rb	The rigidbody we want to add the force generator to
forceGenerator	The force generator we want to add

6.14.3.3 AddRigidbody()

```
void mtrx::mtrxDynamicWorld::AddRigidbody (
            Rigidbody * rb )  [inline]
```

Add a rigidbody to the rigidbody manager.

Parameters

rb	The rigidbody we want to add to the manager

6.14.3.4 RemoveCollider()

```
void mtrx::mtrxDynamicWorld::RemoveCollider (
            Collider * col )  [inline]
```

Remove a collider to the collision system.

Parameters

col	The collider that we want to remove

6.14.3.5 RemoveForceGenerator()

```
void mtrx::mtrxDynamicWorld::RemoveForceGenerator (
            Rigidbody * rb,
            const std::shared_ptr< IRigidbodyForceGenerator > & generator )  [inline]
```

Remove a rigidbody from the rigidbody manager.

Parameters

rb	The rigidbody we want to remove the rigidbody from
generator	The force generator that we want to remove

6.14.3.6 RemoveRigidbody()

```
void mtrx::mtrxDynamicWorld::RemoveRigidbody (
            Rigidbody * rb )  [inline]
```

Remove a rigidbody from the rigidbody manager.

Parameters

rb	The rigidbody we want to remove

6.14.3.7 Update()

```
void mtrx::mtrxDynamicWorld::Update (
            float dt )
```

Update the game world.

Parameters

dt	The time elapsed since the last frame

6.15 mtrx::ObjectAxes Struct Reference

The axes that define an objects world.

`#include <Defs.h>`

Collaboration diagram for mtrx::ObjectAxes:

Public Member Functions

- glm::vec3 & **operator[ ]** (int index)
- const glm::vec3 & **operator[ ]** (int index) const
- **ObjectAxes** (const glm::vec3 &forward=glm::vec3(0, 0, -1), const glm::vec3 &up=glm::vec3(0, 1, 0), const glm::vec3 &side=glm::vec3(1, 0, 0))

Public Attributes

-
```
union {
  struct {
    glm::vec3 side
    glm::vec3 up
    glm::vec3 forward
  }
  glm::vec3 axes [3]
};
```

6.15.1 Detailed Description

The axes that define an objects world.

6.16 mtrx::OOBBCollider Class Reference

Implementation of OOBBs.

`#include <OOBBCollider.h>`

Inheritance diagram for mtrx::OOBBCollider:

Collaboration diagram for mtrx::OOBBCollider:

Public Member Functions

- OOBBCollider (const glm::vec3 &center=glm::vec3(), const glm::quat &orientation=glm::angleAxis(0.f, glm↩ ::vec3(0, 1, 0)), const glm::vec3 &scale=glm::vec3(1, 1, 1))

  Construct a new OOBBCollider object.
- OOBBCollider (const Transform &transform=Transform())

  Construct a new OOBBCollider object.
- virtual ∼OOBBCollider ()=default

  Destroy the OOBBCollider object.
- virtual bool RaycastCollision (const Ray &ray) override

  Raycast OOBB collision detection algorithm.
- const glm::vec3 ∗ GetAxes () const

  Get the Axes of the collider.
- const float ∗ GetHalfExtents () const

  Get the Half Extents of the OOBB collider.
- float ∗ GetHalfExtents ()

  Get the Half Extents of the OOBB collider.
- virtual void SetScale (const glm::vec3 &scale) override

  Set the Scale of the collider.

Private Attributes

- float **halfExtents** [3]
- ObjectAxes **axes**

Additional Inherited Members

6.16.1 Detailed Description

Implementation of OOBBs.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 OOBBCollider() [1/2]

```
mtrx::OOBBCollider::OOBBCollider (
            const glm::vec3 & center = glm::vec3(),
            const glm::quat & orientation = glm::angleAxis(0.f, glm::vec3(0, 1, 0)),
            const glm::vec3 & scale = glm::vec3(1, 1, 1) )
```

Construct a new OOBBCollider object.

Parameters

center	Center of the collider
orientation	Orientation of the collider
scale	The scale of the collider

6.16.2.2 OOBBCollider() [2/2]

```
mtrx::OOBBCollider::OOBBCollider (
            const Transform & transform = Transform() )
```

Construct a new OOBBCollider object.

Parameters

transform	The transform of the collider @mtrx::Transform

6.16.2.3 ~OOBBCollider()

```
virtual mtrx::OOBBCollider::~OOBBCollider ( )  [virtual], [default]
```

Destroy the OOBBCollider object.

6.16.3 Member Function Documentation

6.16.3.1 GetAxes()

```
const glm::vec3* mtrx::OOBBCollider::GetAxes ( ) const  [inline]
```

Get the Axes of the collider.

Returns

const glm::vec3* Array of the axes of the OOBB

Here is the call graph for this function: Here is the caller graph for this function:

6.16.3.2 GetHalfExtents() [1/2]

```
float* mtrx::OOBBCollider::GetHalfExtents ( )  [inline]
```

Get the Half Extents of the OOBB collider.

Returns

float* The half extents of the collider

6.16.3.3 GetHalfExtents() [2/2]

```
const float* mtrx::OOBBCollider::GetHalfExtents ( ) const  [inline]
```

Get the Half Extents of the OOBB collider.

Returns

const float* The half extents of the collider

6.16.3.4 RaycastCollision()

```
virtual bool mtrx::OOBBCollider::RaycastCollision (
            const Ray & ray )  [inline], [override], [virtual]
```

Raycast OOBB collision detection algorithm.

Parameters

ray	Ray that we want to check collision for

See also

mtrx::Ray

Returns

true The ray and collider collide

false The ray and collider do not collide

Reimplemented from mtrx::ConvexShapeCollider.

Here is the call graph for this function:

6.16.3.5 SetScale()

```
virtual void mtrx::OOBBCollider::SetScale (
            const glm::vec3 & scale )  [inline], [override], [virtual]
```

Set the Scale of the collider.

Parameters

scale	The scale of the collider

Reimplemented from mtrx::ConvexShapeCollider.

Here is the call graph for this function:

6.17 mtrx::PotentialCollision Struct Reference

Potential collisions struct.

`#include <BVHNode.h>`

Collaboration diagram for mtrx::PotentialCollision:

Public Attributes

- Body * **bodies** [2]

6.17.1 Detailed Description

Potential collisions struct.

6.18 mtrx::Ray Class Reference

Implementation of a ray.

`#include <Ray.h>`

Collaboration diagram for mtrx::Ray:

Public Member Functions

- Ray (const glm::vec3 &startPos=glm::vec3(), const glm::vec3 &rayDirection=glm::vec3())

  Construct a new Ray object.
- ~Ray ()=default

  Destroy the Ray object.

Public Attributes

- glm::vec3 **startPosition**
- glm::vec3 **direction**

6.18.1 Detailed Description

Implementation of a ray.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Ray()

```
mtrx::Ray::Ray (
            const glm::vec3 & startPos = glm::vec3(),
            const glm::vec3 & rayDirection = glm::vec3() )  [inline]
```

Construct a new Ray object.

Parameters

startPos	The start position of the ray
rayDirection	A normalized vector representing the direction of the ray

6.18.2.2 ~Ray()

```
mtrx::Ray::~Ray ( )  [default]
```

Destroy the Ray object.

6.19 mtrx::rb_BuoyancyForceGenerator Class Reference

Implementation of a buoyancy force generator for rigidbodies.

`#include <rb_BuoyancyForceGenerator.h>`

Inheritance diagram for mtrx::rb_BuoyancyForceGenerator:

Collaboration diagram for mtrx::rb_BuoyancyForceGenerator:

Public Member Functions

- rb_BuoyancyForceGenerator (const glm::vec3 &gravity, float volumeBody, float bodyHalfExtent, float liquid↩ Height=0.f, float density=1000.f)

  Construct a new rb BuoyancyForceGenerator object.
- ∼rb_BuoyancyForceGenerator ()

  Destroy the rb BuoyancyForceGenerator object.
- virtual void UpdateForces (Rigidbody ∗rb, float deltaTime) override

  The implementation of the buoyancy force.

Public Attributes

- float **liquidDensity**
- float **liquidLevel**
- float **bodyHalfExtent**
- float **volumeBody**
- glm::vec3 **gravitationalAcceleration**
- glm::vec3 **centerOfBuoyancy**

6.19.1 Detailed Description

Implementation of a buoyancy force generator for rigidbodies.

This force generator generally assumes that the objects behaves more like a AABB We do not calculate the point of collision with the liquid plane and how much of the object is submerged if it is not perpendicular to the liquid plane

6.19.2 Constructor & Destructor Documentation

6.19.2.1 rb_BuoyancyForceGenerator()

```
mtrx::rb_BuoyancyForceGenerator::rb_BuoyancyForceGenerator (
            const glm::vec3 & gravity,
            float volumeBody,
            float bodyHalfExtent,
            float liquidHeight = 0.f,
            float density = 1000.f )
```

Construct a new rb BuoyancyForceGenerator object.

Parameters

gravity	The gravitational acceleration used by the generator
volumeBody	The volume of the body
bodyHalfExtent	The half extent of the body
liquidHeight	The height of the liquid that would represent the liquid's plane
density	The density of the liquid

6.19.2.2 ~rb_BuoyancyForceGenerator()

```
mtrx::rb_BuoyancyForceGenerator::~rb_BuoyancyForceGenerator ( )
```

Destroy the rb BuoyancyForceGenerator object.

6.19.3 Member Function Documentation

6.19.3.1 UpdateForces()

```
virtual void mtrx::rb_BuoyancyForceGenerator::UpdateForces (
            Rigidbody * rb,
            float deltaTime )  [override], [virtual]
```

The implementation of the buoyancy force.

Parameters

rb	The rigidbody to apply the buoyancy force
deltaTime	The time elapsed since the last frame

Implements mtrx::IRigidbodyForceGenerator.

6.20 mtrx::rb_ForceGenerationRegistry Class Reference

Registry for force generators used to map a rigidbody to all of the force generators that are to be applied to said rigidbody.

```
#include <rb_ForceGenerationRegistry.h>
```

Collaboration diagram for mtrx::rb_ForceGenerationRegistry:

Public Member Functions

- rb_ForceGenerationRegistry ()=default

  Construct a new rb ForceGenerationRegistry object.
- ~rb_ForceGenerationRegistry ()=default

  Destroy the rb ForceGenerationRegistry object.
- void AddForceGenerator (const std::shared_ptr< IRigidbodyForceGenerator > &forceGenerator)

  Add a force generator to the registry.
- void RemoveForceGenerator (const int index)

  Remove a force generator from registry by index.
- void RemoveForceGenerator (const std::shared_ptr< IRigidbodyForceGenerator > &forceGenerator)

  Remove force generator.
- void UpdateForceGenerators (Rigidbody *rb, float deltaTime)

  Apply the force generators onto the rigidbody.

Public Attributes

- std::vector< std::shared_ptr< IRigidbodyForceGenerator > > **forceGenerators**

6.20.1 Detailed Description

Registry for force generators used to map a rigidbody to all of the force generators that are to be applied to said rigidbody.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 rb_ForceGenerationRegistry()

```
mtrx::rb_ForceGenerationRegistry::rb_ForceGenerationRegistry ( )  [default]
```

Construct a new rb ForceGenerationRegistry object.

6.20.2.2 ~rb_ForceGenerationRegistry()

```
mtrx::rb_ForceGenerationRegistry::~rb_ForceGenerationRegistry ( )  [default]
```

Destroy the rb ForceGenerationRegistry object.

6.20.3 Member Function Documentation

6.20.3.1 AddForceGenerator()

```
void mtrx::rb_ForceGenerationRegistry::AddForceGenerator (
            const std::shared_ptr< IRigidbodyForceGenerator > & forceGenerator )  [inline]
```

Add a force generator to the registry.

Parameters

forceGenerator	The force generator to be added

6.20.3.2 RemoveForceGenerator() [1/2]

```
void mtrx::rb_ForceGenerationRegistry::RemoveForceGenerator (
            const int index ) [inline]
```

Remove a force generator from registry by index.

Parameters

index	The position of the force generator

6.20.3.3 RemoveForceGenerator() [2/2]

```
void mtrx::rb_ForceGenerationRegistry::RemoveForceGenerator (
            const std::shared_ptr< IRigidbodyForceGenerator > & forceGenerator ) [inline]
```

Remove force generator.

Parameters

forceGenerator	The force generator that we want to remove

6.20.3.4 UpdateForceGenerators()

```
void mtrx::rb_ForceGenerationRegistry::UpdateForceGenerators (
            Rigidbody * rb,
            float deltaTime ) [inline]
```

Apply the force generators onto the rigidbody.

Parameters

rb	The rigidbody we want to apply the forces to
deltaTime	The time elapsed from last frame

6.21 mtrx::rb_GravityForceGenerator Class Reference

Implementation of gravitational force generator.

`#include <rb_GravityForceGenerator.h>`

Inheritance diagram for mtrx::rb_GravityForceGenerator:

Collaboration diagram for mtrx::rb_GravityForceGenerator:

Public Member Functions

- virtual void UpdateForces (Rigidbody *rb, float deltaTime)
  Implementation of the force generator.
- rb_GravityForceGenerator (const glm::vec3 &gravity)
  Construct a new rb GravityForceGenerator object.
- ~rb_GravityForceGenerator ()
  Destroy the rb GravityForceGenerator object.

Public Attributes

- glm::vec3 **gravitationalAcceleration**

6.21.1 Detailed Description

Implementation of gravitational force generator.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 rb_GravityForceGenerator()

```
mtrx::rb_GravityForceGenerator::rb_GravityForceGenerator (
            const glm::vec3 & gravity )
```

Construct a new rb GravityForceGenerator object.

Parameters

gravity	The gravitational acceleration value that we want to use

6.21.2.2 ~rb_GravityForceGenerator()

```
mtrx::rb_GravityForceGenerator::~rb_GravityForceGenerator ( )
```

Destroy the rb GravityForceGenerator object.

6.21.3 Member Function Documentation

6.21.3.1 UpdateForces()

```
virtual void mtrx::rb_GravityForceGenerator::UpdateForces (
            Rigidbody * rb,
            float deltaTime )  [virtual]
```

Implementation of the force generator.

Parameters

rb	The rigidbody to apply the force onto
deltaTime	The time elapsed since the last frame

Implements mtrx::IRigidbodyForceGenerator.

6.22 mtrx::Rigidbody Class Reference

Implementation of rigidbodies with rigidbody dynamics newtonian force integration and rotational forces.

```
#include <Rigidbody.h>
```

Inheritance diagram for mtrx::Rigidbody:

Collaboration diagram for mtrx::Rigidbody:

Public Member Functions

- Rigidbody (float mass=MAX_MASS, bool iskinematic=false, const glm::vec3 &position=glm::vec3(), const glm::quat &orientation=glm::angleAxis(0.f, glm::vec3(0, 1, 0)), const glm::vec3 &scale=glm::vec3(1, 1, 1), const glm::mat3 &inertiaTensor=glm::mat3(1.0f))

  Construct a new Rigidbody object.
- ~Rigidbody ()=default

  Destroy the Rigidbody object.
- void SetInverseInertiaTensor (const glm::mat3 &inertiaTensor)

  Set the Inverse Inertia Tensor of the body.
- void SetAngularDamping (float angularDamping)

Set the Angular Damping of the rigidbody Angular damping allows us to dampen torque forces applied to the body as a function of time.

- void SetOrientation (glm::quat &orientation)

  Set the Orientation of the body.
- void SetRotation (glm::vec3 &rotation)

  Set the Rotation value of the rigidbody This rotation value is used to generate torque forces.
- void SetIsKinematic (bool kinematic)

  Set the kinematic value.
- glm::mat3 & GetInverseInertiaTensor ()

  Get the Inverse Inertia Tensor of the body.
- float GetAngularDamping () const

  Get the Angular Damping of the body.
- glm::mat3x4 GetObjToWorldMat ()

  Get the Obj To World Mat.
- glm::vec3 & GetRotation ()

  Get the Rotation of the body.
- bool GetIsKinematic ()

  Get whether the body is kinematic.
- glm::mat3 CalculateIITWorld ()

  Calculating the inverse inertia tensor in world space.
- void AddTorque (const glm::vec3 &torque)

  Add a torque force.
- void ClearAccumulators () override

  Clear acculmulators of rigidbody.
- void Integrate (float deltaTime) override

  Integrate the values of the rigidbody.
- void CalculateObjToWorldMat ()

  Calculate object space to world space matrix.
- void AddForceAtPoint (const glm::vec3 &force, const glm::vec3 &point)

  Add a force at a certain point in space.
- void CalculateBodyData ()

  Update some of the information of the rigidbody.

Private Attributes

- glm::vec3 **rotation**
- glm::mat3 **inverseInertiaTensor**
- glm::mat3x4 **objToWorldMat**
- glm::vec3 **accumTorque**
- float **angularDamping**
- bool **isKinematic**
- ObjectAxes **axes**

Additional Inherited Members

6.22.1 Detailed Description

Implementation of rigidbodies with rigidbody dynamics newtonian force integration and rotational forces.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 Rigidbody()

```
mtrx::Rigidbody::Rigidbody (
            float mass = MAX_MASS,
            bool iskinematic = false,
            const glm::vec3 & position = glm::vec3(),
            const glm::quat & orientation = glm::angleAxis(0.f, glm::vec3(0, 1, 0)),
            const glm::vec3 & scale = glm::vec3(1, 1, 1),
            const glm::mat3 & inertiaTensor = glm::mat3(1.0f) )
```

Construct a new Rigidbody object.

Parameters

mass	The mass of the rigidbody
iskinematic	Whether the rigidbody is kinematic. A kinematic object does not affect other rigidbodies in the simulation
position	The position of the rigidbody
orientation	The orientation of the rigidbody
scale	The scale of the rigidbody
inertiaTensor	The inertia tensor of the rigidbody

6.22.2.2 ~Rigidbody()

```
mtrx::Rigidbody::~Rigidbody ( )  [default]
```

Destroy the Rigidbody object.

6.22.3 Member Function Documentation

6.22.3.1 AddForceAtPoint()

```
void mtrx::Rigidbody::AddForceAtPoint (
            const glm::vec3 & force,
            const glm::vec3 & point )
```

Add a force at a certain point in space.

Parameters

force	The force that we want to add
point	The point upon which to apply the force

6.22.3.2 AddTorque()

```
void mtrx::Rigidbody::AddTorque (
            const glm::vec3 & torque )  [inline]
```

Add a torque force.

Parameters

torque	The torque force that we want to add

6.22.3.3 CalculateBodyData()

```
void mtrx::Rigidbody::CalculateBodyData ( )
```

Update some of the information of the rigidbody.

6.22.3.4 CalculateIITWorld()

```
glm::mat3 mtrx::Rigidbody::CalculateIITWorld ( )  [inline]
```

Calculating the inverse inertia tensor in world space.

Returns

glm::mat3 Inverse inertia tensor in world space

6.22.3.5 CalculateObjToWorldMat()

```
void mtrx::Rigidbody::CalculateObjToWorldMat ( )
```

Calculate object space to world space matrix.

6.22.3.6 ClearAccumulators()

```
void mtrx::Rigidbody::ClearAccumulators ( )  [override], [virtual]
```

Clear acculmulators of rigidbody.

Implements mtrx::Body.

6.22.3.7 GetAngularDamping()

```
float mtrx::Rigidbody::GetAngularDamping ( ) const  [inline]
```

Get the Angular Damping of the body.

Returns

float The value of the angular damping

6.22.3.8 GetInverseInertiaTensor()

```
glm::mat3& mtrx::Rigidbody::GetInverseInertiaTensor ( )  [inline]
```

Get the Inverse Inertia Tensor of the body.

Returns

glm::mat3& The inver inertia tensor

6.22.3.9 GetIsKinematic()

```
bool mtrx::Rigidbody::GetIsKinematic ( )  [inline]
```

Get whether the body is kinematic.

Returns

true The body is kinematic

false The body is not kinematic

6.22.3.10 GetObjToWorldMat()

```
glm::mat3x4 mtrx::Rigidbody::GetObjToWorldMat ( )  [inline]
```

Get the Obj To World Mat.

Returns

glm::mat3x4 The object space to world matrix

6.22.3.11 GetRotation()

```
glm::vec3& mtrx::Rigidbody::GetRotation ( )  [inline]
```

Get the Rotation of the body.

Returns

glm::vec3& The rotation of the body

6.22.3.12 Integrate()

```
void mtrx::Rigidbody::Integrate (
            float deltaTime )  [override], [virtual]
```

Integrate the values of the rigidbody.

Parameters

deltaTime	The time elapsed by previous frame

Implements mtrx::Body.

6.22.3.13 SetAngularDamping()

```
void mtrx::Rigidbody::SetAngularDamping (
            float angularDamping )  [inline]
```

Set the Angular Damping of the rigidbody Angular damping allows us to dampen torque forces applied to the body as a function of time.

Parameters

angularDamping	The new value of angular damping

6.22.3.14 SetInverseInertiaTensor()

```
void mtrx::Rigidbody::SetInverseInertiaTensor (
            const glm::mat3 & inertiaTensor )
```

Set the Inverse Inertia Tensor of the body.

Parameters

inertiaTensor	The inertia tensor that we will calculate the inverse of

6.22.3.15 SetIsKinematic()

```
void mtrx::Rigidbody::SetIsKinematic (
            bool kinematic )  [inline]
```

Set the kinematic value.

Parameters

kinematic	the new value of whether the body is kinematic

6.22.3.16 SetOrientation()

```
void mtrx::Rigidbody::SetOrientation (
            glm::quat & orientation )  [inline]
```

Set the Orientation of the body.

Parameters

orientation	The new orientation of the body

Here is the call graph for this function:

6.22.3.17 SetRotation()

```
void mtrx::Rigidbody::SetRotation (
            glm::vec3 & rotation )  [inline]
```

Set the Rotation value of the rigidbody This rotation value is used to generate torque forces.

Parameters

rotation	The new value of the rotation

6.23 mtrx::RigidbodyManager Class Reference

Manager of rigidbodies.

`#include <RigidbodyManager.h>`

Collaboration diagram for mtrx::RigidbodyManager:

Public Member Functions

- RigidbodyManager ()=default

  Construct a new Rigidbody Manager object.
- ~RigidbodyManager ()

  Destroy the Rigidbody Manager object.
- void Integrate (float deltaTime)

  Integrate all entities of the game world.
- void IntegrateRigidbodies (float deltaTime)

  Integrate rigidbodies.
- void UpdateForces (float deltaTime)

  Update the forces applied to the rigidbodies by updating the force generators.

Public Attributes

- std::unordered_map< Rigidbody *, rb_ForceGenerationRegistry > **rigidbodyRegistry**

6.23.1 Detailed Description

Manager of rigidbodies.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 RigidbodyManager()

```
mtrx::RigidbodyManager::RigidbodyManager ( )  [default]
```

Construct a new Rigidbody Manager object.

6.23.2.2 ~RigidbodyManager()

```
mtrx::RigidbodyManager::~RigidbodyManager ( )
```

Destroy the Rigidbody Manager object.

6.23.3 Member Function Documentation

6.23.3.1 Integrate()

```
void mtrx::RigidbodyManager::Integrate (
            float deltaTime )
```

Integrate all entities of the game world.

Parameters

deltaTime	The time elapsed between frames

6.23.3.2 IntegrateRigidbodies()

```
void mtrx::RigidbodyManager::IntegrateRigidbodies (
            float deltaTime )
```

Integrate rigidbodies.

Parameters

deltaTime	The time elapsed between frames

6.23.3.3 UpdateForces()

```
void mtrx::RigidbodyManager::UpdateForces (
```

```
            float deltaTime )
```

Update the forces applied to the rigidbodies by updating the force generators.

Parameters

deltaTime	The time elapsed between frames

6.24 mtrx::Simplex Struct Reference

Simplest shape that can encapsulate a point in 3d space Used within GJK collision detection algorithm.

```
#include <Defs.h>
```

Collaboration diagram for mtrx::Simplex:

Public Attributes

- glm::vec3 **b**
- glm::vec3 **c**
- glm::vec3 **d**
- unsigned int **size**

6.24.1 Detailed Description

Simplest shape that can encapsulate a point in 3d space Used within GJK collision detection algorithm.

6.25 mtrx::SphereCollider Class Reference

Implmentation of a Sphere collider used in collision systems.

```
#include <SphereCollider.h>
```

Inheritance diagram for mtrx::SphereCollider:

Collaboration diagram for mtrx::SphereCollider:

Public Member Functions

- SphereCollider (const glm::vec3 &center=glm::vec3(), const glm::quat &orientation=glm::angleAxis(0.f, worldUp), const glm::vec3 &scale=glm::vec3(1, 1, 1), float radius=0.5)

  Construct a new Sphere Collider object.
- SphereCollider (const Transform &transform=Transform(), float radius=0.5)

  Construct a new Sphere Collider object.
- SphereCollider (const SphereCollider &collider1, const SphereCollider &collider2)

  Construct a new Sphere Collider object that encompass 2 sphere colliders (used within the context of BVH construction mainly)
- virtual ~SphereCollider ()=default

  Destroy the Sphere Collider object.
- virtual bool RaycastCollision (const Ray &ray) override

  Raycast collision check for sphere colliders.
- virtual float GetSize () override

  Get the Size of the sphere (4/3 ∗ PI ∗ radius^ 3)
- virtual float GetGrowth (const SphereCollider &sphereCollider)

  Get the Growth when creating a sphere collider between this collider and another sphere collider (for BVH construction)
- float GetRadius () const

  Get the Radius of the collider.
- void SetRadius (float radius)

  Set the Radius of the collider.
- virtual void SetScale (const glm::vec3 &scale) override

  Set the Scale of the sphere collider. Only the X coordinate of the scale affects the collider, other coordinates are ignored.

Private Attributes

- float **radius**

Additional Inherited Members

6.25.1 Detailed Description

Implmentation of a Sphere collider used in collision systems.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 SphereCollider() [1/3]

```
mtrx::SphereCollider::SphereCollider (
            const glm::vec3 & center = glm::vec3(),
            const glm::quat & orientation = glm::angleAxis(0.f, worldUp),
            const glm::vec3 & scale = glm::vec3(1, 1, 1),
            float radius = 0.5 )
```

Construct a new Sphere Collider object.

Parameters

center	Center position of the sphere
orientation	The orientation of the sphere
scale	The scale of the sphere (influences the radius of the collider)
radius	The default radius of the sphere

Here is the caller graph for this function:

6.25.2.2 SphereCollider() [2/3]

```
mtrx::SphereCollider::SphereCollider (
            const Transform & transform = Transform(),
            float radius = 0.5 )
```

Construct a new Sphere Collider object.

Parameters

transform	The transform of the sphere (position, orientation, scale)

See also

mtrx::Transform

Parameters

radius	the default radius of the sphere

6.25.2.3 SphereCollider() [3/3]

```
mtrx::SphereCollider::SphereCollider (
            const SphereCollider & collider1,
            const SphereCollider & collider2 )
```

Construct a new Sphere Collider object that encompass 2 sphere colliders (used within the context of BVH construction mainly)

Parameters

collider1	First sphere collider
collider2	Second sphere collider

6.25.2.4 ~SphereCollider()

```
virtual mtrx::SphereCollider::~SphereCollider ( )  [virtual], [default]
```

Destroy the Sphere Collider object.

6.25.3 Member Function Documentation

6.25.3.1 GetGrowth()

```
virtual float mtrx::SphereCollider::GetGrowth (
            const SphereCollider & sphereCollider )  [inline], [virtual]
```

Get the Growth when creating a sphere collider between this collider and another sphere collider (for BVH construction)

Parameters

sphereCollider	the sphere collider that we want to add

Returns

float The growth value of the resulting sphere collider

Here is the call graph for this function:

6.25.3.2 GetRadius()

```
float mtrx::SphereCollider::GetRadius ( ) const  [inline]
```

Get the Radius of the collider.

Returns

float Radius of collider

6.25.3.3 GetSize()

```
virtual float mtrx::SphereCollider::GetSize ( )  [inline], [override], [virtual]
```

Get the Size of the sphere (4/3 * PI * radius^3)

Returns

float

Implements mtrx::IBoundingVolume.

6.25.3.4 RaycastCollision()

```
virtual bool mtrx::SphereCollider::RaycastCollision (
            const Ray & ray )  [inline], [override], [virtual]
```

Raycast collision check for sphere colliders.

Parameters

ray	The ray that we are check for collision on

See also

mtrx::Ray

Returns

true The ray and the sphere collide

false The ray and the sphere do not collide

Implements mtrx::Collider.

Here is the call graph for this function:

6.25.3.5 SetRadius()

```
void mtrx::SphereCollider::SetRadius (
            float radius )  [inline]
```

Set the Radius of the collider.

Parameters

radius	The value we want to set the radius to

Here is the call graph for this function:

6.25.3.6 SetScale()

```
virtual void mtrx::SphereCollider::SetScale (
            const glm::vec3 & scale )  [inline], [override], [virtual]
```

Set the Scale of the sphere collider. Only the X coordinate of the scale affects the collider, other coordinates are ignored.

Parameters

scale	The value of the scale that we want to set the collider to

Reimplemented from mtrx::Collider.

Here is the call graph for this function:

6.26 mtrx::Transform Class Reference

Wrapper implementation of a transform that holds position, orientation, scale values and some functionality based on these values.

`#include <Transform.h>`

Collaboration diagram for mtrx::Transform:

Public Member Functions

- Transform (const glm::vec3 &position=glm::vec3(), const glm::quat &orientation=glm::angleAxis(0.f, worldUp), const glm::vec3 &scale=glm::vec3(1, 1, 1))

  Construct a new Transform object.
- ~Transform ()=default

  Destroy the Transform object.
- void Translate (const glm::vec3 &translationVec)

  Translate the position vector.
- void Rotate (const glm::quat &rotation)

  Rotate the orientation value.
- void SetPosition (const glm::vec3 &pos)

  Set the Position vector.
- void SetOrientation (const glm::quat &orientation)

  Set the Orientation quaternion.
- void SetScale (const glm::vec3 &scale)

  Set the Scale vector.
- const glm::vec3 & GetPosition () const

  Get the Position vector.
- const glm::quat & GetOrientation () const

  Get the Orientation quaternion.
- const glm::vec3 & GetScale () const

  Get the Scale vector.
- glm::vec3 & GetPosition ()

  Get the Position vector.
- glm::quat & GetOrientation ()

  Get the Orientation quaternion.
- glm::vec3 & GetScale ()

  Get the Scale vctor.

Private Attributes

- glm::vec3 **position**
- glm::quat **orientation**
- glm::vec3 **scale**

6.26.1 Detailed Description

Wrapper implementation of a transform that holds position, orientation, scale values and some functionality based on these values.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 Transform()

```
mtrx::Transform::Transform (
            const glm::vec3 & position = glm::vec3(),
            const glm::quat & orientation = glm::angleAxis(0.f, worldUp),
            const glm::vec3 & scale = glm::vec3(1, 1, 1) )  [inline]
```

Construct a new Transform object.

Parameters

position	
orientation	
scale	

6.26.2.2 ~Transform()

```
mtrx::Transform::~Transform ( )  [default]
```

Destroy the Transform object.

6.26.3 Member Function Documentation

6.26.3.1 GetOrientation() [1/2]

```
glm::quat& mtrx::Transform::GetOrientation ( )  [inline]
```

Get the Orientation quaternion.

Returns

glm::quat& Orientation value

6.26.3.2 GetOrientation() [2/2]

```
const glm::quat& mtrx::Transform::GetOrientation ( ) const  [inline]
```

Get the Orientation quaternion.

Returns

const glm::quat& Orientation value

Here is the caller graph for this function:

6.26.3.3 GetPosition() [1/2]

```
glm::vec3& mtrx::Transform::GetPosition ( )  [inline]
```

Get the Position vector.

Returns

glm::vec3& Position value

6.26.3.4 GetPosition() [2/2]

```
const glm::vec3& mtrx::Transform::GetPosition ( ) const  [inline]
```

Get the Position vector.

Returns

const glm::vec3& Position value

Here is the caller graph for this function:

6.26.3.5 GetScale() [1/2]

```
glm::vec3& mtrx::Transform::GetScale ( )  [inline]
```

Get the Scale vctor.

Returns

glm::vec3& Scale vector

6.26.3.6 GetScale() [2/2]

```
const glm::vec3& mtrx::Transform::GetScale ( ) const  [inline]
```

Get the Scale vector.

Returns

const glm::vec3& Scale value

Here is the caller graph for this function:

6.26.3.7 Rotate()

```
void mtrx::Transform::Rotate (
            const glm::quat & rotation )  [inline]
```

Rotate the orientation value.

Parameters

rotation	The rotation quaternion that we want to rotate the orientation with

6.26.3.8 SetOrientation()

```
void mtrx::Transform::SetOrientation (
            const glm::quat & orientation )  [inline]
```

Set the Orientation quaternion.

Parameters

orientation	The new orientation value

Here is the caller graph for this function:

6.26.3.9 SetPosition()

```
void mtrx::Transform::SetPosition (
            const glm::vec3 & pos )  [inline]
```

Set the Position vector.

Parameters

pos	The new position value

Here is the caller graph for this function:

6.26.3.10 SetScale()

```
void mtrx::Transform::SetScale (
            const glm::vec3 & scale )  [inline]
```

Set the Scale vector.

Parameters

scale	The new scale value

Here is the caller graph for this function:

6.26.3.11 Translate()

```
void mtrx::Transform::Translate (
            const glm::vec3 & translationVec )  [inline]
```

Translate the position vector.

Parameters

translationVec	Vector used to translate the position of the transform

6.27 mtrx::Triangle Struct Reference

Struct for storing a triangle.

`#include <Defs.h>`

Collaboration diagram for mtrx::Triangle:

Public Member Functions

- **Triangle** (glm::vec3 ∗a, glm::vec3 ∗b, glm::vec3 ∗c)

Public Attributes

-
  union {
    struct {
      glm::vec3 ∗ **a**
      glm::vec3 ∗ **b**
      glm::vec3 ∗ **c**
    }
    glm::vec3 ∗ **pts** [3]
  };

6.27.1 Detailed Description

Struct for storing a triangle.

Index