Triangulate Library

Generated by Doxygen 1.8.13

Contents

1 Namespace Index **1**

1.1 Namespace List 1

2 Class Index **3**

2.1 Class List 3

3 File Index **5**

3.1 File List 5

4 Namespace Documentation **7**

4.1 math Namespace Reference 7

4.2 obj Namespace Reference 7

4.2.1 Function Documentation 7

4.2.1.1 triangulate() 8

5 Class Documentation **9**

5.1 obj::Face Class Reference 9

5.1.1 Detailed Description 9

5.1.2 Constructor & Destructor Documentation 10

5.1.2.1 Face() [1/2] 10

5.1.2.2 Face() [2/2] 10

5.1.3 Member Data Documentation 10

5.1.3.1 n0 10

5.1.3.2 n1 11

5.1.3.3 n2 11

5.1.3.4 tc0 . . . 11
5.1.3.5 tc1 . . . 11
5.1.3.6 tc2 . . . 11
5.1.3.7 v0 . . . 11
5.1.3.8 v1 . . . 11
5.1.3.9 v2 . . . 12
5.2 obj::OBJ Class Reference . . . 12
5.2.1 Detailed Description . . . 12
5.2.2 Constructor & Destructor Documentation . . . 13
5.2.2.1 OBJ() [1/2] . . . 13
5.2.2.2 OBJ() [2/2] . . . 13
5.2.2.3 ~OBJ() . . . 13
5.2.3 Member Function Documentation . . . 13
5.2.3.1 addVertex() . . . 13
5.2.3.2 getFaces() . . . 14
5.2.3.3 getIndices() . . . 14
5.2.3.4 getNormals() . . . 14
5.2.3.5 getUVs() . . . 14
5.2.3.6 getVertices() . . . 15
5.2.3.7 isLoaded() . . . 15
5.2.3.8 loadModel() . . . 15
5.2.3.9 saveModel() . . . 15
5.2.3.10 updateFaces() . . . 16
5.3 math::Vector2 Class Reference . . . 16
5.3.1 Detailed Description . . . 17
5.3.2 Constructor & Destructor Documentation . . . 17
5.3.2.1 Vector2() [1/2] . . . 17
5.3.2.2 Vector2() [2/2] . . . 17
5.3.2.3 ~Vector2() . . . 18
5.3.3 Member Function Documentation . . . 18

5.3.3.1 dot() . . . 18
5.3.3.2 magnitude() . . . 18
5.3.3.3 magnitudeSquared() . . . 19
5.3.3.4 normal() . . . 19
5.3.3.5 normalise() . . . 19
5.3.3.6 operator∗() . . . 19
5.3.3.7 operator∗=() . . . 20
5.3.3.8 operator+() . . . 20
5.3.3.9 operator+=() . . . 20
5.3.3.10 operator-() . . . 20
5.3.3.11 operator-=() . . . 21
5.3.3.12 operator/() . . . 21
5.3.3.13 operator/=() . . . 21
5.3.3.14 operator=() . . . 22
5.3.3.15 operator==() . . . 22
5.3.4 Member Data Documentation . . . 22
5.3.4.1 x . . . 22
5.3.4.2 y . . . 23
5.4 math::Vector3 Class Reference . . . 23
5.4.1 Detailed Description . . . 24
5.4.2 Constructor & Destructor Documentation . . . 24
5.4.2.1 Vector3() [1/2] . . . 24
5.4.2.2 Vector3() [2/2] . . . 24
5.4.2.3 ∼Vector3() . . . 25
5.4.3 Member Function Documentation . . . 25
5.4.3.1 cross() . . . 25
5.4.3.2 dot() . . . 25
5.4.3.3 magnitude() . . . 26
5.4.3.4 magnitudeSquared() . . . 26
5.4.3.5 normal() . . . 26

5.4.3.6 normalise() . . . 26
5.4.3.7 operator∗() . . . 26
5.4.3.8 operator∗=() . . . 27
5.4.3.9 operator+() . . . 27
5.4.3.10 operator+=() . . . 27
5.4.3.11 operator-() . . . 28
5.4.3.12 operator-=() . . . 28
5.4.3.13 operator/() . . . 28
5.4.3.14 operator/=() . . . 29
5.4.3.15 operator=() . . . 29
5.4.3.16 operator==() . . . 29
5.4.4 Member Data Documentation . . . 30
5.4.4.1 x . . . 30
5.4.4.2 y . . . 30
5.4.4.3 z . . . 30

6 File Documentation **31**

6.1 src/lib/obj-loader/src/Face.hpp File Reference . . . 31
6.2 src/lib/obj-loader/src/lib/math-library/src/MathLibrary.hpp File Reference . . . 31
6.3 src/lib/obj-loader/src/lib/math-library/src/Vector2.hpp File Reference . . . 31
6.4 src/lib/obj-loader/src/lib/math-library/src/Vector3.hpp File Reference . . . 32
6.5 src/lib/obj-loader/src/OBJ.cpp File Reference . . . 32
6.6 src/lib/obj-loader/src/OBJ.h File Reference . . . 32
6.7 src/Main.cpp File Reference . . . 33
6.7.1 Function Documentation . . . 33
6.7.1.1 main() . . . 33
6.8 test/main.cpp File Reference . . . 33
6.8.1 Function Documentation . . . 33
6.8.1.1 main() . . . 33
6.9 src/Triangulate.hpp File Reference . . . 33
6.10 test/src/TriangulateTests.cpp File Reference . . . 34
6.10.1 Function Documentation . . . 34
6.10.1.1 TEST() . . . 34

Index **35**

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

math 7
obj 7

Chapter 2

Class Index

2.1 Class List

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

obj::Face
: Class used to represent a face. A face is represented by 3 points. Each point is composed of a position, a texture coordinate and a normal. The data is not saved here but only the index . . . 9

obj::OBJ
: Class for OBJ. Loads and OBJ loader form file, saves a model to path, adds a vertex, updates the faces and gets the data. The data is saved as vectors in an std::vector container id is accessed by index . . . 12

math::Vector2
: 2D Vector class containing 2D vector operations. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.↩ normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare() . . . 16

math::Vector3
: Vector3 structure. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.cross(Vec), Vec.normal(), Vec.↩ normalise(), Vec.magnitude(), Vec.magnitudeSquare() . . . 23

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

src/Main.cpp . . . 33
src/Triangulate.hpp . . . 33
src/lib/obj-loader/src/Face.hpp . . . 31
src/lib/obj-loader/src/OBJ.cpp . . . 32
src/lib/obj-loader/src/OBJ.h . . . 32
src/lib/obj-loader/src/lib/math-library/src/MathLibrary.hpp . . . 31
src/lib/obj-loader/src/lib/math-library/src/Vector2.hpp . . . 31
src/lib/obj-loader/src/lib/math-library/src/Vector3.hpp . . . 32
test/main.cpp . . . 33
test/src/TriangulateTests.cpp . . . 34

Chapter 4

Namespace Documentation

4.1 math Namespace Reference

Classes

- class Vector2

  2D Vector class containing 2D vector operations. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare()
- class Vector3

  Vector3 structure. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.cross(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.↩ magnitudeSquare()

4.2 obj Namespace Reference

Classes

- class Face

  Class used to represent a face. A face is represented by 3 points. Each point is composed of a position, a texture coordinate and a normal. The data is not saved here but only the index.
- class OBJ

  Class for OBJ. Loads and OBJ loader form file, saves a model to path, adds a vertex, updates the faces and gets the data. The data is saved as vectors in an std::vector container id is accessed by index.

Functions

- void triangulate (OBJ &model)

  Triangulate and subdivide an OBJ file. Uses a 3D model loaded and loops through the faces ABC. For each face, computes 3 new points (D, E and F). D = (A + B)/2, E = (B + C)/2 and F = (C + A)/2. Adds D, E and F to the model and retrieved their indices. Delete the current face and create 4 faces: ADF, DBE, DEF and ECF.

4.2.1 Function Documentation

4.2.1.1 triangulate()

```
void obj::triangulate (
            OBJ & model )
```

Triangulate and subdivide an OBJ file. Uses a 3D model loaded and loops through the faces ABC. For each face, computes 3 new points (D, E and F). D = (A + B)/2, E = (B + C)/2 and F = (C + A)/2. Adds D, E and F to the model and retrieved their indices. Delete the current face and create 4 faces: ADF, DBE, DEF and ECF.

Parameters

model	The model to be subdivided

Chapter 5

Class Documentation

5.1 obj::Face Class Reference

Class used to represent a face. A face is represented by 3 points. Each point is composed of a position, a texture coordinate and a normal. The data is not saved here but only the index.

`#include <Face.hpp>`

Public Member Functions

- Face ()

  Constructs the object. Empty constructor sets vertices, texture coordinates and normals indices to 0.
- Face (unsigned v0, unsigned tc0, unsigned n0, unsigned v1, unsigned tc1, unsigned n1, unsigned v2, unsigned tc2, unsigned n2)

  Constructs the object.

Public Attributes

- unsigned v0
- unsigned v1
- unsigned v2
- unsigned tc0
- unsigned tc1
- unsigned tc2
- unsigned n0
- unsigned n1
- unsigned n2

5.1.1 Detailed Description

Class used to represent a face. A face is represented by 3 points. Each point is composed of a position, a texture coordinate and a normal. The data is not saved here but only the index.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Face() [1/2]

```
obj::Face::Face ( )  [inline]
```

Constructs the object. Empty constructor sets vertices, texture coordinates and normals indices to 0.

5.1.2.2 Face() [2/2]

```
obj::Face::Face (
            unsigned v0,
            unsigned tc0,
            unsigned n0,
            unsigned v1,
            unsigned tc1,
            unsigned n1,
            unsigned v2,
            unsigned tc2,
            unsigned n2 )  [inline]
```

Constructs the object.

Parameters

in	*v0*	First vertex index
in	*tc0*	First vertex texture coordinate index
in	*n0*	First vertex normal index
in	*v1*	First vertex index
in	*tc1*	First vertex texture coordinate index
in	*n1*	First vertex normal index
in	*v2*	First vertex index
in	*tc2*	First vertex texture coordinate index
in	*n2*	First vertex normal index

5.1.3 Member Data Documentation

5.1.3.1 n0

```
unsigned obj::Face::n0
```

5.1.3.2 n1

```
unsigned obj::Face::n1
```

5.1.3.3 n2

```
unsigned obj::Face::n2
```

5.1.3.4 tc0

```
unsigned obj::Face::tc0
```

5.1.3.5 tc1

```
unsigned obj::Face::tc1
```

5.1.3.6 tc2

```
unsigned obj::Face::tc2
```

5.1.3.7 v0

```
unsigned obj::Face::v0
```

5.1.3.8 v1

```
unsigned obj::Face::v1
```

5.1.3.9 v2

```
unsigned obj::Face::v2
```

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

- src/lib/obj-loader/src/Face.hpp

5.2 obj::OBJ Class Reference

Class for OBJ. Loads and OBJ loader form file, saves a model to path, adds a vertex, updates the faces and gets the data. The data is saved as vectors in an std::vector container id is accessed by index.

```
#include <OBJ.h>
```

Public Member Functions

- OBJ ()
  Constructs the object. Empty constructor.
- OBJ (const char *modelPath)
  Constructs the object. Loads a 3D model from path.
- ~OBJ ()
  Destroys the object.
- void loadModel (const char *filename)
  Loads a model. Loads a 3D model given a path.
- void saveModel (const char *filename)
  Saves a model. Saves the OBJ to disk given a path.
- bool isLoaded ()
  Determines if loaded. Checks if the OBJ is successfully loaded.
- std::vector< math::Vector3 > getVertices ()
  Gets the vertices. Fetches the list of vertices in the OBJ.
- std::vector< math::Vector3 > getNormals ()
  Gets the normals. Fetches the list of normals in the OBJ.
- std::vector< math::Vector2 > getUVs ()
  Gets the texture coordinates. Fetches the list of UV coordinates in the OBJ.
- std::vector< Face > getFaces ()
  Gets the faces. Fetches the list of faces in the OBJ.
- std::vector< unsigned > getIndices ()
  Gets the indices. Fetches the list of indices in the OBJ.
- unsigned int addVertex (math::Vector3 vertex)
  Adds a vertex. Adds a vertex to the OBJ.
- void updateFaces (std::vector< Face > newFaces)
  Updates the faces. Updates the list of faces.

5.2.1 Detailed Description

Class for OBJ. Loads and OBJ loader form file, saves a model to path, adds a vertex, updates the faces and gets the data. The data is saved as vectors in an std::vector container id is accessed by index.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 OBJ() [1/2]

```
OBJ::OBJ ( )
```

Constructs the object. Empty constructor.

5.2.2.2 OBJ() [2/2]

```
OBJ::OBJ (
            const char * modelPath )
```

Constructs the object. Loads a 3D model from path.

Parameters

in	*modelPath*	The model path to load

5.2.2.3 ~OBJ()

```
obj::OBJ::~OBJ ( )  [inline]
```

Destroys the object.

5.2.3 Member Function Documentation

5.2.3.1 addVertex()

```
unsigned int OBJ::addVertex (
            math::Vector3 vertex )
```

Adds a vertex. Adds a vertex to the OBJ.

Parameters

in	*vertex*	The vertex

Returns

Vertex index.

5.2.3.2 getFaces()

```
std::vector<Face> obj::OBJ::getFaces ( )  [inline]
```

Gets the faces. Fetches the list of faces in the OBJ.

Returns

The faces list.

5.2.3.3 getIndices()

```
std::vector< unsigned > OBJ::getIndices ( )
```

Gets the indices. Fetches the list of indices in the OBJ.

Returns

The indices list.

5.2.3.4 getNormals()

```
std::vector<math::Vector3> obj::OBJ::getNormals ( )  [inline]
```

Gets the normals. Fetches the list of normals in the OBJ.

Returns

The normals list.

5.2.3.5 getUVs()

```
std::vector<math::Vector2> obj::OBJ::getUVs ( )  [inline]
```

Gets the texture coordinates. Fetches the list of UV coordinates in the OBJ.

Returns

The texture coordinates list.

5.2.3.6 getVertices()

```
std::vector<math::Vector3> obj::OBJ::getVertices ( )  [inline]
```

Gets the vertices. Fetches the list of vertices in the OBJ.

Returns

The vertices list.

5.2.3.7 isLoaded()

```
bool obj::OBJ::isLoaded ( )  [inline]
```

Determines if loaded. Checks if the OBJ is successfully loaded.

Returns

True if loaded, False otherwise.

5.2.3.8 loadModel()

```
void OBJ::loadModel (
            const char * filename )
```

Loads a model. Loads a 3D model given a path.

Parameters

in	*filename*	The filename

5.2.3.9 saveModel()

```
void OBJ::saveModel (
            const char * filename )
```

Saves a model. Saves the OBJ to disk given a path.

Parameters

in	*filename*	The filename

5.2.3.10 updateFaces()

```
void obj::OBJ::updateFaces (
            std::vector< Face > newFaces )  [inline]
```

Updates the faces. Updates the list of faces.

Parameters

in	*newFaces*	The new faces

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

- src/lib/obj-loader/src/OBJ.h
- src/lib/obj-loader/src/OBJ.cpp

5.3 math::Vector2 Class Reference

2D Vector class containing 2D vector operations. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare()

```
#include <Vector2.hpp>
```

Public Member Functions

- Vector2 ()

  Constructs the object. Empty construcor sets x=0 and y=0.
- Vector2 (double x, double y)

  Constructs the object. Uses the x and y provided to build the 2D vector.
- ∼Vector2 ()

  Destroys the object.
- Vector2 operator+ (const Vector2 &other)

  Overloads + operator. Vec+Vec, sum of the x and y components of the 2D vector.
- void operator+= (const Vector2 &other)

  Overloads += operator. Vec+=Vec, sum of the x and y components.
- Vector2 operator- (const Vector2 &other)

  Overloads - operator. Vec-Vec, subtraction of x and y components.
- void operator-= (const Vector2 &other)

  Overloads -= operator. Vec-=Vec, subtraction of x and y components.
- void operator= (const Vector2 &other)

  Overloads = operator. Vec=Vec, assign x and y components to left hand side 2D vector.
- bool operator== (const Vector2 &other)

  Overloads == operator. Vec==Vec, compares if x and y components are the same.
- Vector2 operator∗ (double value)

Overloads ∗ operator. Vec∗Vec, multiplies the x and y components of the 2D vectors.

- void operator∗= (double value)

  Overloads ∗= operator. Vec∗=Vec, multiplies the x and y components of the 2D vectors.
- Vector2 operator/ (double value)

  Overloads / operator. Vec/Vec, divide left vector by right vector.
- void operator/= (double value)

  Overloads /= operator. Vec/=Vec, divide left vector by right vector.
- double dot (Vector2 &other)

  Dot product. Vec.dot(Vec), result of the dot product operation. V1 is first vector and V2 second vector, DotProduct Result = V1.x ∗ V2.x + V1.y ∗ V2.y.
- Vector2 normal ()

  Computes the vector mormal. Vec.normal(), divides x and y components by magnitude.
- void normalise ()

  Normalises the vector. Vec.normal(), divides x and y components by magnitude.
- double magnitude ()

  Computes the magnitude. Magnitude = sqrt(x∗x + y∗y).
- double magnitudeSquared ()

  Computes the magnitude squared. MagnitudeSquared = x∗x + y∗y.

Public Attributes

- double x
- double y

5.3.1 Detailed Description

2D Vector class containing 2D vector operations. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare()

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Vector2() [1/2]

```
math::Vector2::Vector2 ( )  [inline]
```

Constructs the object. Empty construcor sets x=0 and y=0.

5.3.2.2 Vector2() [2/2]

```
math::Vector2::Vector2 (
            double x,
            double y )  [inline]
```

Constructs the object. Uses the x and y provided to build the 2D vector.

Parameters

in	*x*	x component of the 2D vector
in	*y*	y component of the 2D vector

5.3.2.3 ~Vector2()

```
math::Vector2::~Vector2 ( )  [inline]
```

Destroys the object.

5.3.3 Member Function Documentation

5.3.3.1 dot()

```
double math::Vector2::dot (
            Vector2 & other )  [inline]
```

Dot product. Vec.dot(Vec), result of the dot product operation. V1 is first vector and V2 second vector, DotProduct↩ Result = V1.x ∗ V2.x + V1.y ∗ V2.y.

Parameters

other	The other vector used in the operation

Returns

Result of the dot product.

5.3.3.2 magnitude()

```
double math::Vector2::magnitude ( )  [inline]
```

Computes the magnitude. Magnitude = sqrt(x∗x + y∗y).

Returns

The vector magnitude.

5.3.3.3 magnitudeSquared()

```
double math::Vector2::magnitudeSquared ( )  [inline]
```

Computes the magnitude squared. MagnitudeSquared = x∗x + y∗y.

Returns

The magnitude squared.

5.3.3.4 normal()

```
Vector2 math::Vector2::normal ( )  [inline]
```

Computes the vector mormal. Vec.normal(), divides x and y components by magnitude.

Returns

Normal 2D vector.

5.3.3.5 normalise()

```
void math::Vector2::normalise ( )  [inline]
```

Normalises the vector. Vec.normal(), divides x and y components by magnitude.

5.3.3.6 operator∗()

```
Vector2 math::Vector2::operator* (
            double value )  [inline]
```

Overloads ∗ operator. Vec∗Vec, multiplies the x and y components of the 2D vectors.

Parameters

in	*value*	The double value used in the operation

Returns

Result of the multiplication.

5.3.3.7 operator*=()

```
void math::Vector2::operator*= (
            double value )  [inline]
```

Overloads *= operator. Vec*=Vec, multiplies the x and y components of the 2D vectors.

Parameters

in	*value*	The double value used in the operation

5.3.3.8 operator+()

```
Vector2 math::Vector2::operator+ (
            const Vector2 & other )  [inline]
```

Overloads + operator. Vec+Vec, sum of the x and y components of the 2D vector.

Parameters

other	The other vector used in the operation

Returns

Result of the addition.

5.3.3.9 operator+=()

```
void math::Vector2::operator+= (
            const Vector2 & other )  [inline]
```

Overloads += operator. Vec+=Vec, sum of the x and y components.

Parameters

other	The other vector used in the operation

5.3.3.10 operator-()

```
Vector2 math::Vector2::operator- (
            const Vector2 & other )  [inline]
```

Overloads - operator. Vec-Vec, subtraction of x and y components.

Parameters

other	The other vector used in the operation

Returns

Result of the subtraction.

5.3.3.11 operator-=()

```
void math::Vector2::operator-= (
            const Vector2 & other )  [inline]
```

Overloads -= operator. Vec-=Vec, subtraction of x and y components.

Parameters

other	The other vector used in the operation

5.3.3.12 operator/()

```
Vector2 math::Vector2::operator/ (
            double value )  [inline]
```

Overloads / operator. Vec/Vec, divide left vector by right vector.

Parameters

in	*value*	The double value used in the operation

Returns

Result of the division.

5.3.3.13 operator/=()

```
void math::Vector2::operator/= (
            double value )  [inline]
```

Overloads /= operator. Vec/=Vec, divide left vector by right vector.

Parameters

in	*value*	The double value used in the operation

5.3.3.14 operator=()

```
void math::Vector2::operator= (
            const Vector2 & other )  [inline]
```

Overloads = operator. Vec=Vec, assign x and y components to left hand side 2D vector.

Parameters

in	*other*	The other vector used in the operation

5.3.3.15 operator==()

```
bool math::Vector2::operator== (
            const Vector2 & other )  [inline]
```

Overloads == operator. Vec==Vec, compares if x and y components are the same.

Parameters

other	The other vector used in the operation

Returns

True if vectors are the same, False otherwise.

5.3.4 Member Data Documentation

5.3.4.1 x

```
double math::Vector2::x
```

5.3.4.2 y

```
double math::Vector2::y
```

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

- src/lib/obj-loader/src/lib/math-library/src/Vector2.hpp

5.4 math::Vector3 Class Reference

Vector3 structure. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.cross(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.↩ magnitudeSquare()

```
#include <Vector3.hpp>
```

Public Member Functions

- Vector3 ()
- Vector3 (double x, double y, double z)

  Constructor. Uses the x, y and z provided to build the 3D vector.
- ∼Vector3 ()

  Destroys the object.
- Vector3 operator+ (const Vector3 &other)

  Overloads + operator. Vec+Vec, sum of the x, y and z components of the 3D vector.
- void operator+= (const Vector3 &other)

  Overloads += operator. Vec+=Vec, sum of the x, y and z components.
- Vector3 operator- (const Vector3 &other)

  Overloads - operator. Vec-Vec, subtraction of x, y and z components.
- void operator-= (const Vector3 &other)

  Overloads -= operator. Vec-=Vec, subtraction of x, y and z components.
- void operator= (const Vector3 &other)

  Overloads equal operator. Vec=Vec, assign x, y and z components to left hand side 3D vector.
- bool operator== (const Vector3 &other)

  Overloads == operator. Vec==Vec, compares if x, y and z components are the same.
- Vector3 operator∗ (double other)

  Overloads multiplication with double. Vec∗Vec, multiplies the x, y and z components of the 3D vectors.
- void operator∗= (double other)

  Overloads ∗= operator. Vec∗=Vec, multiplies the x, y and z components of the 3D vectors.
- Vector3 operator/ (double other)

  Overloads division with double. Vec/Vec, divide left vector by right vector.
- void operator/= (double other)

  Overloads /= operator. Vec/=Vec, divide left vector by right vector.
- double dot (const Vector3 &other)

  Dot product. Vec.dot(Vec), result of the dot product operation. V1 is first vector and V2 second vector, DotProduct↩ Result = V1.x ∗ V2.x + V1.y ∗ V2.y + V1.z ∗ V2.z.
- Vector3 cross (const Vector3 &other)

  Cross product. V1 is the first vector and V2 is the second vector, CrossProduct = [V1.y∗V2.z - V1.z∗V2.y, V1.z∗V2.x - V1.x∗V2.z, V1.x∗V2.y-V1.y∗V2.x].

- Vector3 normal ()

  Computes the vector mormal. Vec.normal(), divides x, y and z components by magnitude.
- void normalise ()

  Normalises the vector. Vec.normal(), divides x, y and z components by magnitude.
- double magnitude ()

  Computes the vector magnitude. Magnitude = sqrt(x∗x + y∗y + z∗z).
- double magnitudeSquared ()

  Computes the magnitude squared. MagnitudeSquared = x∗x + y∗y + z∗z.

Public Attributes

- double x
- double y
- double z

5.4.1 Detailed Description

Vector3 structure. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.cross(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.↩ magnitudeSquare()

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Vector3() [1/2]

```
math::Vector3::Vector3 ( )  [inline]
```

Default constructor. Empty construcor sets x=0, y=0 and z=0.

5.4.2.2 Vector3() [2/2]

```
math::Vector3::Vector3 (
            double x,
            double y,
            double z )  [inline]
```

Constructor. Uses the x, y and z provided to build the 3D vector.

Parameters

in	*x*	x value
in	*y*	y value
in	*z*	z value

5.4.2.3 ~Vector3()

```
math::Vector3::~Vector3 ( )  [inline]
```

Destroys the object.

5.4.3 Member Function Documentation

5.4.3.1 cross()

```
Vector3 math::Vector3::cross (
            const Vector3 & other )  [inline]
```

Cross product. V1 is the first vector and V2 is the second vector, CrossProduct = [V1.y∗V2.z - V1.z∗V2.y, V1.z∗V2.x - V1.x∗V2.z, V1.x∗V2.y-V1.y∗V2.x].

Parameters

in	*other*	The other Vector3 used in the operation

Returns

Result of the cross product.

5.4.3.2 dot()

```
double math::Vector3::dot (
            const Vector3 & other )  [inline]
```

Dot product. Vec.dot(Vec), result of the dot product operation. V1 is first vector and V2 second vector, DotProduct↩ Result = V1.x ∗ V2.x + V1.y ∗ V2.y + V1.z ∗ V2.z.

Parameters

other	The other vector

Returns

Result of the dot product.

5.4.3.3 magnitude()

```
double math::Vector3::magnitude ( )  [inline]
```

Computes the vector magnitude. Magnitude = sqrt(x∗x + y∗y + z∗z).

Returns

Magnitude computed.

5.4.3.4 magnitudeSquared()

```
double math::Vector3::magnitudeSquared ( )  [inline]
```

Computes the magnitude squared. MagnitudeSquared = x∗x + y∗y + z∗z.

Returns

Magnitude squared.

5.4.3.5 normal()

```
Vector3 math::Vector3::normal ( )  [inline]
```

Computes the vector mormal. Vec.normal(), divides x, y and z components by magnitude.

Returns

Normal 3D vector.

5.4.3.6 normalise()

```
void math::Vector3::normalise ( )  [inline]
```

Normalises the vector. Vec.normal(), divides x, y and z components by magnitude.

5.4.3.7 operator∗()

```
Vector3 math::Vector3::operator* (
            double other )  [inline]
```

Overloads multiplication with double. Vec∗Vec, multiplies the x, y and z components of the 3D vectors.

Parameters

in	*other*	Double value used in the operation

Returns

Multiplication result.

5.4.3.8 operator*=()

```
void math::Vector3::operator*= (
            double other )  [inline]
```

Overloads *= operator. Vec*=Vec, multiplies the x, y and z components of the 3D vectors.

Parameters

in	*other*	Double value used in the operation

5.4.3.9 operator+()

```
Vector3 math::Vector3::operator+ (
            const Vector3 & other )  [inline]
```

Overloads + operator. Vec+Vec, sum of the x, y and z components of the 3D vector.

Parameters

in	*other*	The other Vector3 used in the operation

Returns

Sum of the two vectors.

5.4.3.10 operator+=()

```
void math::Vector3::operator+= (
            const Vector3 & other )  [inline]
```

Overloads += operator. Vec+=Vec, sum of the x, y and z components.

Parameters

in	*other*	The other Vector3 used in the operation

5.4.3.11 operator-()

```
Vector3 math::Vector3::operator- (
            const Vector3 & other )  [inline]
```

Overloads - operator. Vec-Vec, subtraction of x, y and z components.

Parameters

in	*other*	The other Vector3 used in the operation

Returns

Substraction of the two vectors.

5.4.3.12 operator-=()

```
void math::Vector3::operator-= (
            const Vector3 & other )  [inline]
```

Overloads -= operator. Vec-=Vec, subtraction of x, y and z components.

Parameters

in	*other*	The other Vector3 used in the operation

5.4.3.13 operator/()

```
Vector3 math::Vector3::operator/ (
            double other )  [inline]
```

Overloads division with double. Vec/Vec, divide left vector by right vector.

Parameters

in	*other*	Double value used in the operation

Returns

Division result.

5.4.3.14 operator/=()

```
void math::Vector3::operator/= (
            double other )  [inline]
```

Overloads /= operator. Vec/=Vec, divide left vector by right vector.

Parameters

in	*other*	Double value used in the operation

5.4.3.15 operator=()

```
void math::Vector3::operator= (
            const Vector3 & other )  [inline]
```

Overloads equal operator. Vec=Vec, assign x, y and z components to left hand side 3D vector.

Parameters

in	*other*	The other Vector3 used in the operation

5.4.3.16 operator==()

```
bool math::Vector3::operator== (
            const Vector3 & other )  [inline]
```

Overloads == operator. Vec==Vec, compares if x, y and z components are the same.

Parameters

other	The other vector

Returns

True if vectors are the same, False otherwise.

5.4.4 Member Data Documentation

5.4.4.1 x

```
double math::Vector3::x
```

5.4.4.2 y

```
double math::Vector3::y
```

5.4.4.3 z

```
double math::Vector3::z
```

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

- src/lib/obj-loader/src/lib/math-library/src/Vector3.hpp

Chapter 6

File Documentation

6.1 src/lib/obj-loader/src/Face.hpp File Reference

Classes

- class obj::Face

  Class used to represent a face. A face is represented by 3 points. Each point is composed of a position, a texture coordinate and a normal. The data is not saved here but only the index.

Namespaces

- obj

6.2 src/lib/obj-loader/src/lib/math-library/src/MathLibrary.hpp File Reference

```
#include "Vector2.hpp"
#include "Vector3.hpp"
```

6.3 src/lib/obj-loader/src/lib/math-library/src/Vector2.hpp File Reference

```
#include <cmath>
```

Classes

- class math::Vector2

  2D Vector class containing 2D vector operations. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare()

Namespaces

- math

6.4 src/lib/obj-loader/src/lib/math-library/src/Vector3.hpp File Reference

```
#include <cmath>
```

Classes

- class math::Vector3

  Vector3 structure. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.cross(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare()

Namespaces

- math

6.5 src/lib/obj-loader/src/OBJ.cpp File Reference

```
#include "OBJ.h"
```

6.6 src/lib/obj-loader/src/OBJ.h File Reference

```
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
#include "MathLibrary.hpp"
#include "Face.hpp"
```

Classes

- class obj::OBJ

  Class for OBJ. Loads and OBJ loader form file, saves a model to path, adds a vertex, updates the faces and gets the data. The data is saved as vectors in an std::vector container id is accessed by index.

Namespaces

- obj

6.7 src/Main.cpp File Reference

```
#include "Triangulate.hpp"
```

Functions

- int main (int argc, char const ∗argv[ ])

6.7.1 Function Documentation

6.7.1.1 main()

```
int main (
            int argc,
            char const * argv[] )
```

6.8 test/main.cpp File Reference

```
#include "gtest/gtest.h"
#include <iostream>
```

Functions

- int main (int argc, char ∗argv[ ])

6.8.1 Function Documentation

6.8.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

6.9 src/Triangulate.hpp File Reference

```
#include "lib/obj-loader/src/OBJ.h"
#include <vector>
```

Namespaces

- obj

Functions

- void obj::triangulate (OBJ &model)

  Triangulate and subdivide an OBJ file. Uses a 3D model loaded and loops through the faces ABC. For each face, computes 3 new points (D, E and F). D = (A + B)/2, E = (B + C)/2 and F = (C + A)/2. Adds D, E and F to the model and retrieved their indices. Delete the current face and create 4 faces: ADF, DBE, DEF and ECF.

6.10 test/src/TriangulateTests.cpp File Reference

```
#include "gtest/gtest.h"
#include "Triangulate.hpp"
```

Functions

- TEST (Triangulate, UpdateFaces)

6.10.1 Function Documentation

6.10.1.1 TEST()

```
TEST (
            Triangulate ,
            UpdateFaces  )
```

Index

~OBJ
    obj::OBJ, 13
~Vector2
    math::Vector2, 18
~Vector3
    math::Vector3, 25

addVertex
    obj::OBJ, 13

cross
    math::Vector3, 25

dot
    math::Vector2, 18
    math::Vector3, 25

Face
    obj::Face, 10

getFaces
    obj::OBJ, 14
getIndices
    obj::OBJ, 14
getNormals
    obj::OBJ, 14
getUVs
    obj::OBJ, 14
getVertices
    obj::OBJ, 14

isLoaded
    obj::OBJ, 15

loadModel
    obj::OBJ, 15

magnitude
    math::Vector2, 18
    math::Vector3, 25
magnitudeSquared
    math::Vector2, 18
    math::Vector3, 26
main
    src/Main.cpp, 33
    test/Main.cpp, 33
math, 7
math::Vector2, 16
    ~Vector2, 18
    dot, 18
    magnitude, 18
    magnitudeSquared, 18
    normal, 19
    normalise, 19
    operator∗, 19
    operator∗=, 19
    operator+, 20
    operator+=, 20
    operator-, 20
    operator-=, 21
    operator/, 21
    operator/=, 21
    operator=, 22
    operator==, 22
    Vector2, 17
    x, 22
    y, 22
math::Vector3, 23
    ~Vector3, 25
    cross, 25
    dot, 25
    magnitude, 25
    magnitudeSquared, 26
    normal, 26
    normalise, 26
    operator∗, 26
    operator∗=, 27
    operator+, 27
    operator+=, 27
    operator-, 28
    operator-=, 28
    operator/, 28
    operator/=, 29
    operator=, 29
    operator==, 29
    Vector3, 24
    x, 30
    y, 30
    z, 30

n0
    obj::Face, 10
n1
    obj::Face, 10
n2
    obj::Face, 11
normal
    math::Vector2, 19
    math::Vector3, 26
normalise
    math::Vector2, 19

math::Vector3, 26

OBJ
obj::OBJ, 13
obj, 7
triangulate, 7
obj::Face, 9
Face, 10
n0, 10
n1, 10
n2, 11
tc0, 11
tc1, 11
tc2, 11
v0, 11
v1, 11
v2, 11
obj::OBJ, 12
∼OBJ, 13
addVertex, 13
getFaces, 14
getIndices, 14
getNormals, 14
getUVs, 14
getVertices, 14
isLoaded, 15
loadModel, 15
OBJ, 13
saveModel, 15
updateFaces, 16
operator∗
math::Vector2, 19
math::Vector3, 26
operator∗=
math::Vector2, 19
math::Vector3, 27
operator+
math::Vector2, 20
math::Vector3, 27
operator+=
math::Vector2, 20
math::Vector3, 27
operator-
math::Vector2, 20
math::Vector3, 28
operator-=
math::Vector2, 21
math::Vector3, 28
operator/
math::Vector2, 21
math::Vector3, 28
operator/=
math::Vector2, 21
math::Vector3, 29
operator=
math::Vector2, 22
math::Vector3, 29
operator==
math::Vector2, 22
math::Vector3, 29

saveModel
obj::OBJ, 15
src/Main.cpp, 33
main, 33
src/Triangulate.hpp, 33
src/lib/obj-loader/src/Face.hpp, 31
src/lib/obj-loader/src/OBJ.cpp, 32
src/lib/obj-loader/src/OBJ.h, 32
src/lib/obj-loader/src/lib/math-library/src/MathLibrary.↩
hpp, 31
src/lib/obj-loader/src/lib/math-library/src/Vector2.hpp, 31
src/lib/obj-loader/src/lib/math-library/src/Vector3.hpp, 32

TEST
TriangulateTests.cpp, 34
tc0
obj::Face, 11
tc1
obj::Face, 11
tc2
obj::Face, 11
test/Main.cpp
main, 33
test/main.cpp, 33
test/src/TriangulateTests.cpp, 34
triangulate
obj, 7
TriangulateTests.cpp
TEST, 34

updateFaces
obj::OBJ, 16

v0
obj::Face, 11
v1
obj::Face, 11
v2
obj::Face, 11
Vector2
math::Vector2, 17
Vector3
math::Vector3, 24

x
math::Vector2, 22
math::Vector3, 30

y
math::Vector2, 22
math::Vector3, 30

z
math::Vector3, 30