

Triton

Generated by Doxygen 1.7.3

Sat May 5 2012 09:26:19

Contents

1	Triton(tm) Software Development Kit User's Manual	1
2	Obtaining a license for Triton, or how to evaluate it for free	3
3	Triton System Requirements	5
4	Getting started with Triton	7
4.1	Overview of the sample projects	7
4.2	Configuring your project	7
4.2.1	Linking with Triton under Windows	7
4.2.2	Linking with Triton under Linux	8
4.2.3	Using Triton's headers	9
4.3	Intializing Triton	9
4.4	Rendering each frame	11
4.4.1	Updating Triton's lighting conditions	11
4.4.2	Updating Triton's camera matrices	11
4.4.3	Rendering the Ocean	12
4.5	Integrating environment cube maps with Triton	12
4.6	Shutting down	13
5	Simulating ship wakes with Triton	15
6	Spray effects and breaking waves	17
7	Integrating Triton with terrain	19
8	Synchronizing Triton across multiple viewports or channels	21
9	Intersection tests with Triton	23
10	Underwater rendering with Triton	25
11	Performance tuning tips	27
12	Troubleshooting tips	29
13	Redistributing Triton with your application	31
14	Obtaining support	33
15	Advanced topics	35

15.1	Integrating with your own resource manager	35
15.2	Integrating with your own memory manager	35
15.3	Adding your own effects to the water	36
15.4	Building Triton from source	36
15.5	Integrating planar reflection maps with Triton	37
16	Third-party license notices	39
16.1	FFTSS: A Fast Fourier Transform Library	39
16.2	Intel's Integrated Performance Primitives	40
16.3	Cgtextures.com	40
16.4	AMD Accelerated Parallel Processing Math Libraries	40
17	Class Index	43
17.1	Class Hierarchy	43
18	Class Index	45
18.1	Class List	45
19	File Index	47
19.1	File List	47
20	Class Documentation	49
20.1	Triton::Allocator Class Reference	49
20.1.1	Detailed Description	50
20.1.2	Member Function Documentation	50
20.1.2.1	SetAllocator	50
20.2	Triton::Environment Class Reference	50
20.2.1	Detailed Description	56
20.2.2	Constructor & Destructor Documentation	56
20.2.2.1	Environment	56
20.2.2.2	~Environment	56
20.2.3	Member Function Documentation	56
20.2.3.1	AddWindFetch	56
20.2.3.2	ClearWindFetches	56
20.2.3.3	CullSphere	57
20.2.3.4	GetAboveWaterVisibility	57
20.2.3.5	GetAmbientLightColor	57
20.2.3.6	GetBelowWaterVisibility	57
20.2.3.7	GetCameraMatrix	58
20.2.3.8	GetCameraPosition	58
20.2.3.9	GetConfigOption	58
20.2.3.10	GetCoordinateSystem	58
20.2.3.11	GetDevice	58
20.2.3.12	GetDirectionalLightColor	58
20.2.3.13	GetEnvironmentMap	59
20.2.3.14	GetEnvironmentMapMatrix	59
20.2.3.15	GetLightDirection	59
20.2.3.16	GetPlanarReflectionDisplacementScale	59
20.2.3.17	GetPlanarReflectionMap	59
20.2.3.18	GetPlanarReflectionMapMatrix	59

20.2.3.19	GetProjectionMatrix	59
20.2.3.20	GetRandomNumberGenerator	60
20.2.3.21	GetRenderer	60
20.2.3.22	GetResourceLoader	60
20.2.3.23	GetRightVector	60
20.2.3.24	GetSeaLevel	60
20.2.3.25	GetUpVector	60
20.2.3.26	GetWind	60
20.2.3.27	GetWorldUnits	61
20.2.3.28	Initialize	61
20.2.3.29	IsDirectX	62
20.2.3.30	IsGeocentric	62
20.2.3.31	IsOpenGL	62
20.2.3.32	SetAboveWaterVisibility	62
20.2.3.33	SetAmbientLight	62
20.2.3.34	SetBelowWaterVisibility	62
20.2.3.35	SetCameraMatrix	63
20.2.3.36	SetConfigOption	63
20.2.3.37	SetDirectionalLight	63
20.2.3.38	SetEnvironmentMap	64
20.2.3.39	SetLicenseCode	64
20.2.3.40	SetPlanarReflectionMap	64
20.2.3.41	SetProjectionMatrix	66
20.2.3.42	SetRandomNumberGenerator	66
20.2.3.43	SetSeaLevel	66
20.2.3.44	SetWorldUnits	66
20.2.3.45	SimulateSeaState	67
20.3	Triton::Matrix3 Class Reference	67
20.3.1	Detailed Description	69
20.3.2	Constructor & Destructor Documentation	69
20.3.2.1	Matrix3	69
20.3.2.2	Matrix3	69
20.3.2.3	Matrix3	69
20.3.2.4	~Matrix3	70
20.3.3	Member Function Documentation	70
20.3.3.1	FromRx	70
20.3.3.2	FromRy	70
20.3.3.3	FromRz	70
20.3.3.4	FromXYZ	70
20.3.3.5	operator*	70
20.3.3.6	operator*	70
20.3.3.7	ToFloatArray	70
20.3.3.8	Transpose	70
20.3.4	Friends And Related Function Documentation	71
20.3.4.1	operator*	71
20.4	Triton::Matrix4 Class Reference	71
20.4.1	Detailed Description	73
20.4.2	Constructor & Destructor Documentation	73
20.4.2.1	Matrix4	73
20.4.2.2	Matrix4	73

20.4.2.3	Matrix4	73
20.4.2.4	~Matrix4	74
20.4.3	Member Function Documentation	74
20.4.3.1	GetRow	74
20.4.3.2	InverseCramers	74
20.4.3.3	operator*	74
20.4.3.4	operator*	74
20.4.3.5	operator*	74
20.4.3.6	ToFloatArray	74
20.4.3.7	Transpose	74
20.4.4	Friends And Related Function Documentation	74
20.4.4.1	operator*	74
20.5	Triton::MemObject Class Reference	75
20.5.1	Detailed Description	77
20.6	Triton::Ocean Class Reference	77
20.6.1	Detailed Description	80
20.6.2	Constructor & Destructor Documentation	80
20.6.2.1	~Ocean	80
20.6.3	Member Function Documentation	80
20.6.3.1	Create	80
20.6.3.2	D3D9DeviceLost	80
20.6.3.3	D3D9DeviceReset	81
20.6.3.4	Draw	81
20.6.3.5	EnableSpray	81
20.6.3.6	EnableWireframe	81
20.6.3.7	GetChoppiness	82
20.6.3.8	GetDepth	82
20.6.3.9	GetFFTName	82
20.6.3.10	GetHeight	82
20.6.3.11	GetNumTriangles	83
20.6.3.12	GetRefractionColor	83
20.6.3.13	GetShaderObject	83
20.6.3.14	GetWaveHeading	83
20.6.3.15	SetChoppiness	84
20.6.3.16	SetDepth	84
20.6.3.17	SetRefractionColor	84
20.6.3.18	SprayEnabled	85
20.7	Triton::RandomNumberGenerator Class Reference	85
20.7.1	Detailed Description	86
20.7.2	Member Function Documentation	86
20.7.2.1	GetRandomDouble	86
20.7.2.2	GetRandomInt	87
20.8	Triton::ResourceLoader Class Reference	87
20.8.1	Detailed Description	88
20.8.2	Member Function Documentation	89
20.8.2.1	FreeResource	89
20.8.2.2	LoadResource	89
20.8.2.3	SetResourceDirPath	89
20.9	Triton::Utils Class Reference	90
20.9.1	Detailed Description	90

20.9.2	Member Function Documentation	91
20.9.2.1	ClearGLErrors	91
20.9.2.2	GetDLLExtension	91
20.9.2.3	GetDLLPath	91
20.9.2.4	GetDLLSuffix	91
20.9.2.5	GetDX9Macros	91
20.9.2.6	PrintGLErrors	91
20.10	Triton::Vector3 Class Reference	92
20.10.1	Detailed Description	94
20.10.2	Constructor & Destructor Documentation	94
20.10.2.1	Vector3	94
20.10.2.2	Vector3	94
20.10.3	Member Function Documentation	94
20.10.3.1	Cross	94
20.10.3.2	Dot	94
20.10.3.3	Length	94
20.10.3.4	Normalize	94
20.10.3.5	operator!=	94
20.10.3.6	operator*	95
20.10.3.7	operator*	95
20.10.3.8	operator+	95
20.10.3.9	operator+	95
20.10.3.10	operator-	95
20.10.3.11	operator==	95
20.10.3.12	Serialize	95
20.10.3.13	SquaredLength	95
20.10.3.14	Unserialize	95
20.10.4	Member Data Documentation	96
20.10.4.1	x	96
20.11	Triton::Vector3f Class Reference	96
20.11.1	Detailed Description	97
20.11.2	Constructor & Destructor Documentation	97
20.11.2.1	Vector3f	97
20.11.2.2	Vector3f	98
20.11.2.3	Vector3f	98
20.11.3	Member Function Documentation	98
20.11.3.1	Dot	98
20.11.3.2	Length	98
20.11.3.3	Normalize	98
20.11.3.4	operator+	98
20.11.3.5	operator-	98
20.11.4	Member Data Documentation	98
20.11.4.1	x	98
20.12	Triton::Vector4 Class Reference	99
20.12.1	Detailed Description	100
20.12.2	Constructor & Destructor Documentation	100
20.12.2.1	Vector4	100
20.12.2.2	Vector4	100
20.12.3	Member Function Documentation	101
20.12.3.1	Dot	101

20.12.3.2 operator*	101
20.12.3.3 operator*	101
20.12.3.4 operator+	101
20.12.3.5 operator+	101
20.12.3.6 operator-	101
20.12.4 Member Data Documentation	101
20.12.4.1 x	101
20.13 Triton::WakeGenerator Class Reference	101
20.13.1 Detailed Description	103
20.13.2 Constructor & Destructor Documentation	103
20.13.2.1 WakeGenerator	103
20.13.3 Member Function Documentation	104
20.13.3.1 GetPosition	104
20.13.3.2 GetVelocity	104
20.13.3.3 Update	104
20.14 Triton::WindFetch Class Reference	104
20.14.1 Detailed Description	106
20.14.2 Constructor & Destructor Documentation	106
20.14.2.1 WindFetch	106
20.14.3 Member Function Documentation	106
20.14.3.1 ClearLocalization	106
20.14.3.2 GetWindAtLocation	106
20.14.3.3 SetLocalization	107
20.14.3.4 SetWind	107
21 File Documentation	109
21.1 C:/triton/trunk/Public Headers/Environment.h File Reference	109
21.1.1 Detailed Description	111
21.2 C:/triton/trunk/Public Headers/Matrix3.h File Reference	111
21.2.1 Detailed Description	113
21.3 C:/triton/trunk/Public Headers/Matrix4.h File Reference	113
21.3.1 Detailed Description	115
21.4 C:/triton/trunk/Public Headers/MemAlloc.h File Reference	115
21.4.1 Detailed Description	116
21.5 C:/triton/trunk/Public Headers/Ocean.h File Reference	117
21.5.1 Detailed Description	118
21.6 C:/triton/trunk/Public Headers/RandomNumberGenerator.h File Reference	118
21.6.1 Detailed Description	120
21.7 C:/triton/trunk/Public Headers/ResourceLoader.h File Reference	120
21.7.1 Detailed Description	122
21.8 C:/triton/trunk/Public Headers/Triton.h File Reference	122
21.8.1 Detailed Description	123
21.9 C:/triton/trunk/Public Headers/TritonCommon.h File Reference	123
21.9.1 Detailed Description	125
21.10 C:/triton/trunk/Public Headers/Vector3.h File Reference	125
21.10.1 Detailed Description	126
21.11 C:/triton/trunk/Public Headers/Vector4.h File Reference	126
21.11.1 Detailed Description	128
21.12 C:/triton/trunk/Public Headers/WakeGenerator.h File Reference	128

21.12.1 Detailed Description	130
21.13C:/triton/trunk/Public Headers/WindFetch.h File Reference	130
21.13.1 Detailed Description	131

Chapter 1

Triton(tm) Software Development Kit User's Manual

Thanks for using Triton from Sundog Software! Triton is a C++ library for real-time water rendering, featuring easy integration with OpenGL 2.0+, DirectX9, and DirectX11 based Windows applications. It provides realistic visuals of waves for any given wind conditions or a specified Beaufort scale, and takes advantage of general-purpose GPU (GPGPU) computing on systems that support OpenCL, CUDA, or DirectX11 Compute Shaders (DirectCompute.) This allows us to compute Fast Fourier Transforms on thousands of waves at once at rates of over 100 frames per second on most modern systems.

Triton also features the ability to seamlessly integrate with geocentric coordinate systems in addition to flat coordinate systems. The surface of Triton's sea may be configured to conform to a WGS84 ellipsoid, or a spherical model of the Earth.

This manual will get you started - you should be surprised at how quickly you can be up and running. We also provide a detailed class reference for our public interfaces; licensed users of Triton will receive Triton's full source and full documentation of our internal classes as well.

- [Obtaining a license for Triton, or how to evaluate it for free](#)
- [Triton System Requirements](#)
- [Getting started with Triton](#)
 - [Overview of the sample projects](#)
 - [Configuring your project](#)
 - [Intializing Triton](#)
 - [Integrating environment cube maps with Triton](#)
 - [Rendering each frame](#)
 - [Shutting down](#)
- [Simulating ship wakes with Triton](#)

- [Spray effects and breaking waves](#)
- [Integrating Triton with terrain](#)
- [Intersection tests with Triton](#)
- [Underwater rendering with Triton](#)
- [Synchronizing Triton across multiple viewports or channels](#)
- [Performance tuning tips](#)
- [Troubleshooting tips](#)
- [Redistributing Triton with your application](#)
- [Obtaining support](#)
- [Advanced topics](#)
 - [Integrating with your own resource manager](#)
 - [Integrating with your own memory manager](#)
 - [Adding your own effects to the water](#)
 - [Building Triton from source](#)
 - [Integrating planar reflection maps with Triton](#)
- [Third-party license notices](#)

Chapter 2

Obtaining a license for Triton, or how to evaluate it for free

License codes for Triton may be purchased online at <http://www.sundog-soft.com/> using a credit card; for other payment options, contact sales@sundog-soft.com.

Once you've received your license name and code, all you have to do is pass them into `Triton::Environment::SetLicenseCode()` after you've created your Environment object, and all restrictions on Triton will be removed.

We do make Triton freely available in an "evaluation mode" if you don't pass it a valid license code. This lets you confirm you can integrate Triton with your application before committing to a purchase. Without a license code, Triton will display a warning dialog at startup, and terminate your application after five minutes of runtime - but this should be sufficient to see if you can get Triton up and running with your project.

Our license terms are simple; a single price lets you distribute as many copies or channels of your executable, and incremental updates to it, that links in Triton. No royalties, no per-channel costs, no per-developer costs. Licensees also receive Triton's full source, 3 months of technical support, and free updates.

If you have any questions about Triton's licensing (or about anything, really,) contact us at sales@sundog-soft.com.

Chapter 3

Triton System Requirements

Although Triton is able to take advantage of the latest general purpose GPU technologies and graphics SDK's, it remains compatible with lower end systems as well.

Any system capable of running OpenGL 2.0, or DirectX9 with Shader Model 3.0, is supported. Triton may also be used with OpenGL 3.x, OpenGL 4.x, and DirectX 11. Support for DirectX9Ex is also provided.

It is possible to run Triton on older systems with Shader Model 2.0 pixel shaders and 3.0 vertex shaders, but typically these systems are not powerful enough to run Triton reliably.

If you encounter any trouble with compatibility or performance on a supported system profile, don't hesitate to contact us at support@sundog-soft.com.

Chapter 4

Getting started with Triton

Triton includes over 60,000 lines of code, but it will only take a few to integrate it into your application.

Here's how.

4.1 Overview of the sample projects

Examining the "Samples" directory of the SDK is a quick way to get started. You'll find simple applications illustrating the use of Triton in OpenGL, DirectX9, and DirectX11 applications, with some handy functions for initializing, updating, and shutting down Triton that you can use in your own app. Each sample also includes a sky box class, which is used not just to make the sample look prettier, but also illustrates the integration of environment cube maps with Triton for more realistic reflections from the sky.

Samples for Ogre and OpenSceneGraph are also included.

The OpenGL sample is built on OpenGL 2.0 functionality. However, the core code of the OpenGL sample avoids use of the fixed function pipeline, and should be illustrative for developers working under OpenGL 3, 4, and beyond as well.

4.2 Configuring your project

4.2.1 Linking with Triton under Windows

Triton provides libraries for win32 and x64 applications created with Microsoft Visual Studio 2005, 2008, or 2010 (with the latest service packs applied and security patches) for every runtime library flavor. You'll find the libraries in the lib directory of the SDK. In your project properties, add the appropriate library to your linker inputs.

The Triton SDK installer defines the environment variable TRITON_PATH that you may use when referencing Triton's libraries and headers in your project properties.

For example, for a Win32 application developed with Visual Studio 2010 and using the "multi-threaded DLL" runtime, you'd link against

```
$(TRITON_PATH)/lib/vc10/win32/Triton-MT-DLL.lib.
```

Visual Studio 2005 libraries are found in "lib/vc8", Visual Studio 2008 libraries are found in "lib/vc9", and Visual Studio 2010 libraries in "lib/vc10". Refer to the following table for matching the appropriate library file with the runtime your project is using (which you can find under the "C/C++ / Code Generation" property page in your project.)

Runtime flavor	Triton library
Multi-threaded	Triton-MT.dll
Multi-threaded Debug	Triton-MTD.dll
Multi-threaded DLL	Triton-MT-DLL.dll
Multi-threaded Debug DLL	Triton-MTD-DLL.dll

Even though we provide specific builds for individual compilers and runtimes, some of the third party DLL's we incorporate do not. If you run into trouble linking while using runtimes other than multi-threaded DLL, try adding MSVCRT to the "Ignore specific default libraries" field of the "Linker / Input" property page of your project.

4.2.2 Linking with Triton under Linux

We provide pre-built binaries for Ubuntu, OpenSUSE, and RHEL6 32 and 64 bit OS's on our website; running the Triton installation script (for example, `install-triton-eval.run`) will automatically extract Triton's own libraries as well as several third party libraries it depends on. Libraries for AMD's APPML libraries, Intel's Integrated Performance Primitives, and CUFFT will be installed into your `/usr/local/lib/triton` directory. Remember to ensure these are installed on any target systems when you distribute your application, or else Triton will fall back to CPU-based FFT methods with much lower performance.

You will also need to have the latest graphics drivers installed. Notably, the open-source ATI drivers provided with Ubuntu are, as of this writing, known to not properly support OpenCL/OpenGL interoperability which will cause Triton to fail on ATI cards. You'll need to uninstall the `fglrx` drivers and install the latest from AMD's website if this happens.

An example project is found inside the `Samples/OpenGLSample` directory of the SDK. Just run "make" to produce a binary within the `OpenGLSample` directory. Examining the Makefile provided here is a good starting point. You'll need to have the `freelut` package installed as well in order to successfully link.

If you have purchased a license for Triton and have the full source distribution, building Triton from source requires several third-party development kits to be installed first:

FFTSS: <http://www.ssisc.org/fftss/download.html> AMD's APP SDK:

<http://developer.amd.com/sdks/AMDAPPSDK/downloads/Pages/default.aspx>

NVidia's CUDA Toolkit: <http://developer.nvidia.com/cuda-toolkit-40>

AMD's APPML: <http://developer.amd.com/libraries/appmathlibs/Pages/default.aspx>

Intel's Integrated Performance Primitives: <http://software.intel.com/en-us/articles/intel-ipp>

Use CMake to generate the makefiles to build Triton from source. From the root SDK directory, just running `cmake ./` followed by `make` should rebuild Triton and the OpenGL sample application (which will be built inside the `Samples/OpenGLSample` directory.) If CMake gives you errors about missing variables such as `GLUT_Xi_LIBRARY` or `GLUT_Xmu_LIBRARY`, you may need to install the following packages first:

```
sudo apt-get install libxmu-dev libxi-dev
```

For distributions other than the ones we explicitly support, contact support@sundog-soft.com. We have most popular flavors of Linux installed and can produce custom builds on request, or provide you with the resources you need to produce a build tailored for your system.

4.2.3 Using Triton's headers

Under the "Additional include directories" field in your project's "C/C++" property page, add the following:

```
$(TRITON_PATH)/Public Headers
```

With this in place, you can simply

```
#include "Triton.h"
```

to gain access to Triton's capabilities.

4.3 Intializing Triton

There are three main objects you'll need to create at startup in order to use Triton.

The first thing you'll need is a [Triton::ResourceLoader](#). This class lets Triton access its shaders, DLL's, and texture resources, and it's what you'll use to tell Triton where these resources are located. The SDK includes a "resources" directory that you may redistribute, and you just need to provide an absolute or relative path to where you installed this directory in the [Triton::ResourceLoader](#)'s constructor. If you're interested in integrating Triton with your own resource manager, see [Integrating with your own resource manager](#).

Next, you'll need to create and initialize a [Triton::Environment](#) object, using the [Triton::ResourceLoader](#) you just made. The [Triton::Environment](#) class lets you specify the coordinate system, rendering system, and environmental conditions affecting Triton's water. For example, to integrate Triton with a DirectX11-based simulation application in a geocentric coordinate system using the WGS84 ellipsoid with the Z axis pointing through the poles, you'd call [Triton::Environment::Initialize\(\)](#) with the parameters `Triton::WGS84_ZUP` and `Triton::DIRECTX_11` as well as the [ResourceLoader](#) you created. You'll find support for flat-earth coordinate systems and spherical systems as

well, with "up" on the Z or Y axes, and for OpenGL 2.x, 3.x, 4.x, DirectX9, and DirectX11. Be sure to check for error codes from `Triton::Environment::Initialize()` - the most likely problem is that the resource path used to create your ResourceLoader isn't quite right.

If you have purchased a license for Triton, call `Triton::Environment::SetLicenseCode()` to remove the evaluation restrictions on the SDK.

You'll then need to add some wind to Triton's simulation, or there won't be any waves. Create one or more `Triton::WindFetch` objects and add them via `Triton::Environment::AddWindFetch()`. A wind fetch represents a region of a specific wind speed and direction, which may be localized to an ellipsoidal area. If more than one wind fetch is present at a location, they'll be added together. Alternately, you can use the `Triton::Environment::SimulateSeaState()` method to quickly simulate a given state on the Beaufort scale.

Finally, you'll create the `Triton::Ocean` object itself, using the `Triton::Environment` you've created. There are two `Triton::WaterModelTypes` you can select from when constructing an Ocean: TESSENDORF and GERSTNER. You'll almost always want to use TESSENDORF which uses Fast Fourier Transforms to simulate thousands of waves at once, as it looks the best. However, if performance on low-end systems is important and you only need to simulate a handful of waves in a small body of water with light wind, you may want to give GERSTNER a try. If `Triton::Ocean::Create` returns a null pointer, see [Troubleshooting tips](#) to figure out what's going on.

A complete example of initializing Triton's objects follows.

```
// Create the Triton objects at startup, once we have a valid GL context in place

bool InitTriton()
{
    // We use the default resource loader that just loads files from disk. You'll
    // need
    // to redistribute the resources folder if using this. You can also extend th
    // e
    // ResourceLoader class to hook into your own resource manager if you wish.
    resourceLoader = new Triton::ResourceLoader("../..\\resources\\");

    // Create an environment for the water, with a flat-Earth coordinate system w
    // ith Y
    // pointing up and using an OpenGL 2.0 capable context.
    environment = new Triton::Environment();
    Triton::EnvironmentError err = environment->Initialize(Triton::FLAT_YUP,
        Triton::OPENGL_2_0, resourceLoader);
    if (err != Triton::SUCCEEDED) {
        ::MessageBoxA(NULL, "Failed to initialize Triton - is the resource path p
        ased in to "
            "the ResouceLoader constructor valid?", "Triton error", MB_OK | MB_IC
            ONEXCLAMATION);
        return false;
    }

    // Substitute your own license name and code, otherwise the app will terminat
    // e after
    // 5 minutes. Visit www.sundog-soft.com to purchase a license if you're so in
    // clined.
    environment->SetLicenseCode("Your license name", "Your license code");

    // Set up wind of 10 m/s blowing North
    Triton::WindFetch wf;
```

```
wf.SetWind(10.0, 0.0);
environment->AddWindFetch(wf);

// Finally, create the Ocean object using the environment we've created.
// If NULL is returned, something's wrong - enable the enable-debug-messages
// option
// in resources/triton.config to get more details on the problem.
ocean = Triton::Ocean::Create(environment, Triton::TESSENDORF);

return (ocean != NULL);
}
```

4.4 Rendering each frame

If your camera includes a region of your scene containing water, you'll need to render your Ocean object as part of your scene. Doing involves three steps:

4.4.1 Updating Triton's lighting conditions

Call `Triton::Environment::SetDirectionalLight()` to specify the color and direction of sunlight (or moonlight.) This information will be used to create specular reflections of the sun or moon in the water.

`Triton::Environment::SetAmbientLight()` provides the ambient skylight used to light the seafoam and the water itself, if not environment map is provided (see [Integrating environment cube maps with Triton](#))

You might use Triton in conjunction with a system that provides scene lighting from dynamic time of day effects, such as Sundog Software's SilverLining library (see <http://www.sundog-soft.com/>) Or, these lighting values might be derived from the skybox texture you're using.

For example:

```
// Position the sun 45 degrees up in the sky at full brightness:
Triton::Vector3 lightPosition(0, 1.0 / sqrt(2.0), 1.0 / sqrt(2.0));
environment->SetDirectionalLight(lightPosition, Triton::Vector3(1.0, 1.0, 1.0));

// Ambient color of the sky:
environment->SetAmbientLight(Triton::Vector3(0.6, 0.9, 0.9));
```

4.4.2 Updating Triton's camera matrices

Since Triton does not rely on fixed function pipelines, you'll need to tell it explicitly what your camera matrices are so we may render the ocean consistently with the rest of your scene. Just pass in the modelview and projection matrices for your scene using `Triton::Environment::SetCameraMatrix()` and `Triton::Environment::SetProjectionMatrix()`. Both methods take in an array of 16 doubles. For example:

```
double projection[16];
double modelview[16];
```

```
// How you retrieve your camera matrices will vary depending on the engine
// you're using, so we'll just postulate the existence of these methods:
GetProjectionMatrix(projection);
GetModelviewMatrix(modelview);

environment->SetCameraMatrix(modelview);
environment->SetProjectionMatrix(projection);
```

Engines sometimes vary on whether they expose row-major or column-major matrices. If your ocean isn't rendering properly, try transposing the matrix before passing it in. If all else fails, try constructing these matrices from scratch using the camera's position, field of view, clip planes, and aspect ratio. Refer to the sample code included with Triton for examples.

4.4.3 Rendering the Ocean

With the lighting and camera information in place, we can now draw our ocean. Just call `Triton::Ocean::Draw()` and you're done. For example:

```
// Draw the ocean for the current time sample
if (ocean) {
    DWORD millis = timeGetTime();
    ocean->Draw((double)millis * 0.001);
}
```

Note that an explicit time sample is passed in, so you may manipulate the passage of time however you wish.

4.5 Integrating environment cube maps with Triton

By default, Triton will just reflect whatever color you passed in with `Triton::Environment::SetAmbientLight()` in the water. For more realistic reflections, you'll want to set an environmental cube map using `Triton::Environment::SetEnvironmentMap()`. Doing so is optional, but it makes a big difference.

The type of texture parameter passed into this method will vary depending on what renderer you're using. OpenGL users should pass in a `GLuint` representing the texture ID of the cube map. DirectX9 users should pass in a `LPDIRECT3DCUBETEXTURE9`. DirectX11 users should pass a `ID3D11ShaderResourceView` pointer.

Examples of loading and constructing properly constructed cube maps from disk may be found in the SkyBox classes in the SDK's sample code. You might also dynamically generate these cube maps based on a physical simulation of the sky such as Sundog Software's SilverLining library. The demo application for Triton found on our website does exactly that.

4.6 Shutting down

At shutdown time, just delete your Triton objects in the reverse order in which you created them - first your [Triton::Ocean](#), then your [Triton::Environment](#), and finally your [Triton::ResourceLoader](#). We'll clean up our memory and resources. For example:

```
// Clean up our resources
void Destroy()
{
    if (ocean) delete ocean;
    if (environment) delete environment;
    if (resourceLoader) delete resourceLoader;
}
```


Chapter 5

Simulating ship wakes with Triton

Triton includes the ability to displace the ocean surface in 3D with wakes generated by multiple ships, or more generally objects moving through the water.

In addition to the standard "Kelvin wake" of 19.46 degrees found behind ships moving at constant velocity in a straight line and a realistically modeled turbulent wake behind the propellers, Triton also properly handles ships that are accelerating, decelerating, or moving along arbitrary paths - all at constant framerates.

Including wakes in your simulation is simple. Just create a [Triton::WakeGenerator](#) object for every ship or object you wish to simulate in your scene, associated with the [Triton::Ocean](#) you'll attach it to.

```
Triton::WakeGenerator *ship = new Triton::WakeGenerator(ocean);
```

To take advantage of more advanced wake simulation features, you may pass additional information to the WakeGenerator constructor. For example, let's set up a wake for a ship that has a length of 100 meters with particle-based spray effects at the bow (100 meters ahead of the source of the wake), and a beam width of 20 meters. With this information, the turbulent wake behind the ship will expand at a realistic rate given the length and beam width of the ship, which may be useful for training purposes.

```
Triton::WakeGenerator *ship = new Triton::WakeGenerator(ocean, true, 100.0, 100.0, 20.0);
```

Then, each frame, update the ship propellers' position, direction, and velocity using [Triton::WakeGenerator::Update\(\)](#). The position is in world units, the velocity in world units per second, and the timestamp is in seconds. The direction is a normalized vector in world space.

```
ship->Update(Triton::Vector3(shipX, shipY, shipZ), shipDirection, shipVelocity, now);
```

That's all there is to it! Once you stop calling Update on the WakeGenerator, any existing wakes will dissipate over time. Just delete the WakeGenerator object when

you're done with it, and make sure the `Tirton::Ocean` it's attached to is initialized and is not deleted during the `WakeGenerator`'s lifetime. Make sure the velocity is realistic and accurate, in order to receive realistic and accurate wakes.

You may simulate as many `WakeGenerators` as you'd like in a scene. As the wakes are applied inside Triton's vertex programs, there are a finite amount of individual wake waves that may be drawn in any specific frame. Triton will select the waves closest to the camera if there are more waves being simulated that can be drawn at once. You may adjust this upper limit using the `max-wake-waves` settings in `resources/Triton.config`.

Note that the turbulent wake simulation may be taxing on some systems, as it is performed almost entirely on the GPU. If you experience poor performance when wakes are in the scene, try setting the value `"wake-propeller-backwash"` to `"no"` inside the file `resources/Triton.config`, and the propeller backwash behind ships will be disabled. Similarly, the 3D Kelvin wakes may also be disabled using the `"wake-kelvin-wakes"` setting. Disabling one or the other wake effects will significantly improve performance when wakes are visible, as will adjusting the `max-wake-waves` parameter for the renderer you are using.

Chapter 6

Spray effects and breaking waves

Triton looks for sharp wave crests and automatically creates particle-based spray effects near the camera.

As the wind increases and the waves become higher, or as the choppiness is increased, the spray effects will become more pronounced.

These particle systems are highly optimized, but still come at a performance cost. If you'd like to disable the spray effects to maximize Triton's performance, just call `Triton::Ocean::EnableSpray()` to turn it on or off at runtime. To disable spray effects entirely, see the `fft-enable-spray` setting in the `resources/Triton.config` file.

Fine control over the appearance of the spray effects is available in `Triton.config`; see the "Spray Settings" section in that file to learn how to adjust the appearance and placement of the spray effects to your liking.

Chapter 7

Integrating Triton with terrain

If your application includes terrain as well as open ocean, there are some best practices to follow.

Be sure to call [Triton::Ocean::SetDepth\(\)](#) if your camera is near the shoreline. This method allows you to specify the depth and surface normal of the seafloor at the camera position. At shallow depths, this information is used to make the waves more pronounced near the shore, and to make the water more transparent near the shoreline. If your terrain includes textured geometry extending under the water for some distance offshore, your ocean floor will influence the color of the ocean near the shore. The Triton demo application available from our website illustrates this effect when the "Beach" checkbox is enabled.

For handling the water/land boundary, the simplest approach is to rely on the depth buffer. You'll need to ensure your depth buffer has adequate precision for this to work well. Use a 32 bit depth buffer if your system supports it, and ensure your near and far clip planes are set as close together as is practical.

If your existing terrain database includes geometry for the ocean surface, you'll want to remove it in order to prevent depth fighting between Triton's oceans and your existing ocean plane. Ideally, your terrain geometry will include bathymetry data near the shore, and include the seafloor sloping away from the shoreline.

If your terrain includes areas of land that are below the simulated sea level, you'll need to prevent Triton from rendering waves in these areas. A simple solution is to render your terrain while writing to the stencil buffer, and then enabling the stencil test surrounding the call to [Triton::Ocean::Draw\(\)](#).

You may adjust the sea level of Triton using [Triton::Environment::SetSeaLevel\(\)](#).

Chapter 8

Synchronizing Triton across multiple viewports or channels

In multi-channel simulators that render several viewports at once using multiple computers, you need to ensure that the ocean is rendered consistently across the different systems.

Fortunately, [Triton::Ocean::Draw\(\)](#) takes an explicit time sample as its parameter. As long as you're synchronizing the time passed in across the channels, the display should be consistent.

There is a random component to the wave generation, that just uses the standard `rand()` function. Initialize each channel consistently with `srand()` and you should be fine. Alternately, refer to the documentation for the [Triton::RandomNumberGenerator](#) class - you may subclass this interface, and provide your own random number generator to Triton using the `Environment::SetRandomNumberGenerator()` method.

Chapter 9

Intersection tests with Triton

Triton allows you to query the height of the ocean surface at any given position; this allows you to simulate floating object in your application consistently with Triton's waves.

In order to use intersection tests, you must create your Ocean object with the `enableHeightTests` parameter of `Triton::Ocean::Create()` set to true. By default, intersection tests are disabled, which allows Triton to avoid the performance hit of reading data back from the GPU on some systems.

With a properly constructed `Triton::Ocean`, simply call the `Triton::Ocean::GetHeight()` method at runtime to query the height of the ocean surface at the intersection of a given ray and the ocean surface. For example:

```
Triton::Vector3 testPos(0.0, 100.0, 0.0);
Triton::Vector3 down(0.0, -1.0, 0.0);
float height = 0;
if (ocean && ocean->GetHeight(testPos, down, height)) {
    // do something with the height at this point...
}
```

The height returned will be accurate to within one grid vertex of the ocean mesh.

Chapter 10

Underwater rendering with Triton

Triton provides support for rendering the water surface from above or below sea level.

Rendering the rest of the scene's objects underwater is the responsibility of the application, but we provide hooks to ensure the sea surface is rendered consistently with the rest of your scene.

You'll want to take a look at the method [Triton::Environment::SetBelowWaterVisibility\(\)](#). In addition to rendering the water surface properly from below the water, this method will allow you to fog the water surface to a given visibility and fog color.

When underwater, setting a fog color that's consistent with the background and fog used for the rest of the underwater scene will yield good results. Clearing the back-buffer to match the color specified, or using a skybox with a specific color below the horizon, will work well.

Note that these methods take in a visibility value; this will be translated into an exponential fog extinction value using the Koschmieder equation: $\text{visibility} = 3.912 / \text{extinction}$.

Chapter 11

Performance tuning tips

Triton will use as many special capabilities as it can find of your graphics hardware, taking advantage of CUDA, OpenCL, and the parallel computing capabilities of your main CPU whenever possible.

If you need even more performance, there are some things you can tweak.

If you believe vertex processing may be your bottleneck, you can decrease the number of polygons used in the projected grid used to render the ocean. Open up the file `resources/Triton.config` in a text editor, and reduce the value of `default-grid-resolution`. As the number of polygons increases with the square of this value, reducing it can have a big impact on some systems.

Computing the Fast Fourier Transforms associated with the TESSENDORF wave model can also be expensive. You can reduce the cost of these computations by reducing the values of `fft-grid-dimension-x` and `fft-grid-dimension-y` to a smaller power of 2. You may want to also reduce the `fft-grid-size` settings to match in order to avoid a loss in resolution (at the cost of increased tiling.)

As a last resort, you may want to consider using the GERSTNER wave model instead of TESSENDORF to avoid the FFT's altogether. However, the Gerstner model is limited to only 16 waves, and as such won't look realistic unless you're simulating calm conditions in a small body of water.

If you're feeling adventurous, you can also edit the shaders used by Triton to simplify them. You'll find them in the resources folder; eliminating the sections that apply foam and noise may help performance slightly, but in our experience the small cost of these effects is well worth the visual quality.

Chapter 12

Troubleshooting tips

Since Triton draws its waves using a fine-grained screen-aligned grid, it's particularly vulnerable to aliasing artifacts.

Make sure you've got anti-aliasing enabled in your application and your drivers have good texture filtering options enabled.

What if nothing shows up?

Make sure you're checking for errors returned from [Triton::Environment::Initialize\(\)](#). If this method fails, you'll get an informative error code back - which will most likely tell you that the path to Triton's resources directory specified in the constructor of [Triton::ResourceLoader](#) was invalid.

Also ensure you're getting back a valid [Triton::Ocean](#) object from [Triton::Ocean::Create\(\)](#). This method may return NULL, which likely indicates a driver compatibility issue we haven't encountered before. The first thing to try is updating your graphics drivers to the latest release, but if that fails, we provide a way to get more information about what's going on with Triton under the hood.

Open up the file `resources/Triton.config` in a text editor, and change the setting `enable-debug-messages` to "yes". Now, when you run your application in Debug mode, you'll get information in the Output window of Visual Studio prefaced with "TRITON:" that tells you more about how it selected its underlying FFT method, and error information.

If these messages implicate a specific FFT implementation as running into problems, the simplest thing is to disable the culprit. The settings `disable-cuda`, `disable-ipp`, `disable-opencl`, and `disable-compute-shader` will let you force Triton to not use a FFT method that's potentially problematic on your system. You may want to try enabling `fft-force-cpu` to force Triton to use its built-in CPU-based FFT transform, which has no special system dependencies or DLL dependencies at all.

If you receive the message "Failed to initialize projected grid" after calling [Triton::Ocean::Create\(\)](#) with no other messages before it, you may be creating the Ocean from a different thread than you used to create the GL context. Make sure all of your Triton calls are done in the same thread as the one your context was created within.

Another likely culprit is the matrices passed into Triton via [Triton::Environment::SetCameraMatrix\(\)](#) and [Triton::Environment::SetProjectionMatrix\(\)](#). Double check that these matrices

contain what you expect, and try transposing them in case your engine's conventions differ from ours. If you just can't get the transforms right, try creating these matrices "from scratch" from your camera properties, as illustrated in the sample code provided with the SDK.

There are some known issues with Intel integrated graphics and how writes to floating point textures are handled, which can lead to missing or garbled wave heights. Try updating your drivers.

If you're still running into trouble, or you believe you've encountered a system compatibility issue we should know about, please send us a note at support@sundog-soft.com. We're happy to provide limited pre-sales support to you, and we're always very interested in identifying and fixing any new bugs we haven't come across before.

Chapter 13

Redistributing Triton with your application

Windows developers must ensure that the DirectX end-user runtimes are installed on your target systems, from the June 2010 DirectX SDK or newer.

Triton's runtime dependencies are contained within the "resources" directory of the SDK, which you are free to redistribute with your application. You're also free to roll the contents of this directory into your own resource manager if you wish; see [Integrating with your own resource manager](#) for more information. Linux users must also ensure that the IPP, clAmdFft, and cufft shared objects that we installed into your /usr/local/lib/triton directory are installed on your target systems in locations that are part of the library search path.

If Windows developers want to trim down the size of the resources directory, it's safe to remove any of the DLL's in the resources/vc9 and resources/vc10 directories that you're not using. If you built your solution with Visual Studio 2008, it's safe to delete the entire vc10 directory. If your application's built for win32 instead of x64, the x64 subdirectory can go. Then, within your surviving directory, any DLL's for runtime libraries you aren't using are safe to remove as well. You'll also find two directories containing runtime dependencies for triton: resources/dll and resources/dll64. If your application is built for x64, it's safe to remove the dll folder. If it's built for Win32, it's safe to remove dll64.

You may also remove shaders from the resources directory that you aren't using. If you're application is for DirectX only, all the .glsl files can go, for example - or, OpenGL users may safely remove the .fx files.

As with any Windows applications, if you link against the DLL runtime libraries, you'll also need to ensure that the Visual C++ runtimes for your compiler and architecture are installed on your target systems as well.

Chapter 14

Obtaining support

We're happy to provide limited email-based pre-sales technical support if you're evaluating SilverLining, and licensed customers receive 3 months of support as well.

Just contact support@sundog-soft.com with your questions. Be sure to include which renderer you're using, what your graphics card and driver version is, and what kind of CPU you're using so we can better help you.

Chapter 15

Advanced topics

15.1 Integrating with your own resource manager

It's possible to derive your own [Triton::ResourceLoader](#) class to hook into your own resource manager. Implement your own [Triton::ResourceLoader::LoadResource\(\)](#) and [Triton::ResourceLoader::FreeResource\(\)](#) methods to grab Triton's shader and graphical resources, located in the resources directory of the SDK, from any resource database you might have. Then, pass your derived [Triton::ResourceLoader](#) into [Triton::Environment::Initialize\(\)](#) and it will be called instead of our default one.

The DLL's within our resources directory will require special care, however. You'll need to keep the vc9 and/or vc10 subdirectories of the resources directory in place on disk, and continue to pass a valid path to the resources directory to the [Triton::ResourceLoader](#) constructor so Triton will know where to find our FFT DLL's. Failure to do so will force Triton to fall back on a CPU-only FFT transform, which will hurt performance in a big way.

You'll also need to ensure the third-party DLL's located in the resources/dll directory are distributed in a place where Triton will be able to load them as part of the DLL search path. Our default implementation of [Triton::ResourceLoader::SetResourceDirPath\(\)](#) calls the Windows function `SetDllDirectory` to add this directory to the DLL search path. You'll want to emulate this behavior, or redistribute these DLL's alongside your application's executable or working directory so Triton will find them when loaded.

15.2 Integrating with your own memory manager

If you'd like to hook into your own memory manager instead of using ours (which is just based on the Windows functions `HeapAlloc()` and `HeapFree()`), it's possible to do so.

Extend your own [Triton::Allocator](#) class from the one defined in [MemAlloc.h](#). Then, pass it in via the static [Triton::Allocator::SetAllocator\(\)](#) method prior to calling any other Triton methods or instantiating any Triton objects.

Since Triton operates across DLL boundaries, you'll want to take care that a consistent heap is used, as we do. Licensed users have access to Triton's full source, and you may find it informative to examine our own implementation of [Triton::Allocator](#) before creating your own.

We take care to capture every usage of new, delete, malloc, and free within Triton, as well as hooking into any memory allocated from STL objects. Memory may be allocated by system functions and third party libraries (such as CUDA, OpenCL, and FFTSS) that is outside of our control, however.

15.3 Adding your own effects to the water

You'll find Triton's shaders in plain text inside the Resources folder; .glsl files are for OpenGL and .fx are effect files for DirectX9 and DirectX11. You're free to make changes to these shaders as you see fit to customize your application, or integrate more tightly with your graphics engine. For example, you could introduce support for planar reflections or shadows on the water using textures generated by your application.

If you are developing changes to Triton's shaders, it's a good idea to enable the enable-debug-messages setting in resources/Triton.config so you'll see any compilation errors in the shaders.

Your modifications will likely require additional parameters to be passed in to the shaders. To enable this, we offer the [Triton::Ocean::GetShaderObject\(\)](#) method. Depending on the renderer being used, this will return to you a GLhandle, a ID3DXEffect pointer, or a ID3DX11Effect pointer, which you can use to pass in uniform parameters of your own - for example, your own textures or matrices used to transform them as they are combined with the water.

For deeper integration, we offer full source licenses in addition to binary licenses on our website (www.sundog-soft.com). However, even with a binary license you receive the shader sources and the ability to hook into them as described here.

15.4 Building Triton from source

Licensed customers of the full source Triton Ocean SDK will receive a special SDK installer that includes Triton's full source code, enabling you to modify Triton to meet your own special needs.

You'll find solution files for both Visual Studio 2008 and Visual Studio 2010 included at the top level directory of the SDK.

In order to compile the TritonOpenCL DLL project, you'll need to have AMD's AMD Accelerated Parallel Processing (APP) SDK installed.

In order to compile the TritonCUDA DLL project, you'll need NVidia's CUDA Toolkit 4.0 or newer installed.

Both are freely available from AMD's and Nvidia's websites.

You'll find that the documentation included with the full source SDK includes refer-

ences on all of Triton's internal classes, in addition to the public API's.

If you have any trouble building Triton, drop a note to support@sundog-soft.com.

15.5 Integrating planar reflection maps with Triton

In addition to environment cube maps, you may also pass in planar reflection maps for use with Triton. This is useful for generating local reflections from ships and terrain. Use `Triton::Environment::SetPlanarReflectionMap()` for this effect.

The type of texture parameter passed into this method will vary depending on what renderer you're using. OpenGL users should pass in a GLuint representing the texture ID of the planar reflection texture. DirectX9 users should pass in a LPDIRECT3DTEXTURE2D. DirectX11 users should pass a ID3D11ShaderResourceView pointer.

Triton can use planar reflection and environment map textures together. If both are applied, the alpha channel of the planar reflection texture is used to blend between planar reflection and environment cube map reflections. This is a good way to get the "best of both worlds," with the cube map providing environmental reflections from steep wave angles, and the planar reflection map providing local reflections directly above the water surface.

Generation of planar reflection maps can be considered to be an advanced topic. It requires a render to texture pass to produce a proper reflection map. The scene rendered to such a texture has to be mirrored in the reflection plane by scaling the height values by -1. Such scaling flips winding directions, so it's often necessary to turn off backface culling or to swap the order of vertices of front and backfaces. Application of clip planes to cut off pieces of scene models normally hidden below the water surface may be also required. All these topics are described in great detail in various documents available on the Internet. If you seek more info try searching the web for "Water Rendering" and you will surely find plenty of documentation on the topic of rendering mirror textures.

Together with a planar reflection texture, Triton requires passing a texture matrix used to project Triton's computed reflection vector to texture coords. In the most common scenario, this is exactly the View*Projection matrix used to render the reflected scene scaled and translated to 0..1 x 0..1 texture coord space. Usually such a View*Projection matrix will be equal to the main scene's View * Projection matrix. However, more complex scenarios aiming to optimize the use of reflection map space can adopt less intuitive View and Projection matrices. Such unusual TextureMatrices can be also passed to `Triton::Environment::SetEnvironmentMap()`.

Triton's "input" Vector multiplied by the planar reflection TextureMatrix is defined in world space coordinates translated to the view point. In other words, this coordinate space has the same orientation as world space but its origin (point 0,0,0) is moved to the camera location. This is the same coordinate space that the env map texture matrix uses as "input" space. Advanced developers will certainly benefit from browsing the code of Triton's pixel (fragment) shaders available in Resources folder.

Triton's "input" vector multiplied by planar reflection Texture Matrix is a view vector perturbed by normal.xy components. Such a vector only approximates real wave reflection vectors which should not be used directly to access the planar reflection map.

Realtime bumpy surface reflection methods are often based on approximation of reflections from flat mirrors. However, this approach has serious limitations. It assumes that the view vector (and resulting reflection vector) angle is strictly correlated with reflection incident point on the mirror surface. If the surface is not perfectly flat and water waves are of course an example of such bumpy surface, above assumption fails and many points at the ocean surface can reflect vectors in the same direction. If these reflection vectors were directly used to address a planar map texels we could see reflections of the objects at random and often very distant unrelated locations.

To avoid the above effect, some limits are often imposed on reflected vector or reflected texture coords. Triton adopts the classic approach to the above problem which works by perturbing the view vector with `normal.xy` multiplied by a fixed displacement scale constant. Since `normal.xy` components are never larger than the unit value we can be sure that the reflection vector will fit in a finite margin defined by the displacement scale set by the Triton programmer.

Chapter 16

Third-party license notices

Triton incorporates some third-party code, and their required license notices are here.

Triton's own license terms are found in the file `license.txt` installed with your SDK, and were presented to you upon installation. We do not include any GPL software in Triton, and there are no demands to make your product "open source" in any of these terms.

All third party trademarks and logos are the property of their respective owners.

16.1 FFTSS: A Fast Fourier Transform Library

Copyright 2002-2007 Akira Nukada. All rights reserved. Copyright 2002-2007 The Scalable Software Infrastructure Project, supported by "Development of Software Infrastructure for Large Scale Scientific Simulation" Team, CREST, JST. Akira Nishida, Department of Computer Science, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE SCALABLE SOFTWARE INFRASTRUCTURE PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE SCALABLE SOFTWARE INFRASTRUCTURE PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED

TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

16.2 Intel's Integrated Performance Primitives

Decompiling or reverse engineering the DLL's included in the Triton SDK's resources/dll directory is expressly prohibited.

16.3 Cgtextures.com

The sky box textures used in our sample applications are courtesy of cgtextures.com, and may not be redistributed.

16.4 AMD Accelerated Parallel Processing Math Libraries

END USER LICENSE AGREEMENT PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE. BY USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, DO NOT USE THE SOFTWARE.

1. License. The software accompanying this License (hereinafter "Software", regardless of the media on which it is distributed, are licensed to you by Advanced Micro Devices, Inc. ("AMD"). You own the medium on which the Software is recorded, but AMD and AMD's Licensors (referred to collectively as "AMD") retain title to the Software and related documentation. You may: a) use the Software.; and b) make a reasonable number of copies necessary for the purposes of this License. You must reproduce on such copy AMD's copyright notice and any other proprietary legends that were on the original copy of the Software

2. Restrictions. The Software contains copyrighted and patented material, trade secrets and other proprietary material. In order to protect them, and except as permitted by applicable legislation, you may not: a) decompile, reverse engineer, disassemble or otherwise reduce the Software to a human-perceivable form; b) modify, network, rent, lend, loan, distribute or create derivative works based upon the Software in whole or in part; or c) electronically transmit the Software from one computer to another or over a network or otherwise transfer the Software except as permitted by this License.

3. Termination. This License is effective until terminated. You may terminate this License at any time by destroying the Software, related documentation and all copies thereof. This License will terminate immediately without notice from AMD if you fail to comply with any provision of this License. Upon termination you must destroy the Software, related documentation and all copies thereof.

4. Government End Users. If you are acquiring the Software on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees the Software and documentation were developed at private expense and are provided with "RESTRICTED RIGHTS". Use, duplication, or disclosure by the Government is subject to restrictions as set forth in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1) (ii) (Oct 1988), FAR 12.212(a)(1995), FAR 52.227-19, (June 1987) or FAR 52.227-14(ALT III) (June 1987), as amended from time to time. In the event that this License, or any part thereof, is deemed inconsistent with the minimum rights identified in the Restricted Rights provisions, the minimum rights shall prevail.

5. No Other License. No rights or licenses are granted by AMD under this License, expressly or by implication, with respect to any proprietary information or patent, copyright, trade secret or other intellectual property right owned or controlled by AMD, except as expressly provided in this License.

6. Additional Licenses. DISTRIBUTION OR USE OF THE SOFTWARE WITH AN OPERATING SYSTEM MAY REQUIRE ADDITIONAL LICENSES FROM THE OPERATING SYSTEM VENDOR. Additional third party licenses may also be required and you agree that you shall be solely responsible for obtaining such license rights.

7. Disclaimer of Warranty on Software. You expressly acknowledge and agree that use of the Software is at your sole risk. The Software and related documentation are provided "AS IS" and without warranty of any kind and AMD EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, ACCURACY, CONDITION, OWNERSHIP, FITNESS FOR A PARTICULAR PURPOSE, AND/OR OF NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS, AND THOSE ARISING FROM CUSTOM OR TRADE OR COURSE OF USAGE. AMD DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE SOFTWARE WILL BE CORRECTED. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. FURTHERMORE, AMD DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE OR RELATED DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY AMD OR AMD'S AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU (AND NOT AMD OR AMD'S AUTHORIZED REPRESENTATIVE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. THE SOFTWARE IS NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING OR LIFE SUSTAINING APPLICATIONS. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

8. Limitation of Liability. UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL AMD, OR ITS DIRECTORS, OFFICERS, EMPLOYEES OR AGENTS ("AUTHORIZED REPRESENTATIVES"), BE LIABLE TO YOU FOR ANY

PUNITIVE, EXEMPLARY, DIRECT, INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE, MISUSE OR INABILITY TO USE THE SOFTWARE OR RELATED DOCUMENTATION, BREACH OR DEFAULT, INCLUDING THOSE ARISING FROM INFRINGEMENT OR ALLEGED INFRINGEMENT OF ANY PATENT, TRADEMARK, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT, BY AMD, EVEN IF AMD OR AMD'S AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. AMD will not be liable for: 1) loss of, or damage to, your records or data; or 2) any damages claimed by you based on any third party claim. In no event shall AMD's total liability to you for all damages, losses, and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the amount paid by you for the Software.

9. Export Restrictions. You shall adhere to all U.S. and other applicable export laws, including but not limited to the U.S. Export Administration Regulations (EAR), currently found at 15 C.F.R. Sections 730 through 744. Further, pursuant to 15 C.F.R. Section 740.6, You hereby certifies that, except pursuant to a license granted by the United States Department of Commerce Bureau of Industry and Security or as otherwise permitted pursuant to a License Exception under the U.S. Export Administration Regulations ("EAR"), You will not (1) export, re-export or release to a national of a country in Country Groups D:1 or E:2 any restricted technology, software, or source code it receives from AMD, or (2) export to Country Groups D:1 or E:2 the direct product of such technology or software, if such foreign produced direct product is subject to national security controls as identified on the Commerce Control List (currently found in Supplement 1 to Part 774 of EAR). For the most current Country Group listings, or for additional information about the EAR or Recipient's obligations under those regulations, please refer to the U.S. Bureau of Industry and Security's website at <http://www.bis.doc.gov/>. These export requirements shall survive any expiration or termination of this Agreement.

10. Controlling Law and Severability. This Agreement will be governed by and construed under the laws of the State of California without reference to its conflicts of law principles. The rights and obligations under this Agreement shall not be governed by the United Nations Convention on Contracts or the International Sale of Goods, the application of which is expressly excluded. Each party hereto submits to the jurisdiction of the state and federal courts of Santa Clara County and the Northern District of California for the purpose of all legal proceedings arising out of or relating to this Agreement or the subject matter hereof. Each party waives any objection which it may have to contest such forum.

11. Complete Agreement. This License constitutes the entire agreement between the parties with respect to the use of the Software and the related documentation, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of AMD.

Chapter 17

Class Index

17.1 Class Hierarchy

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

Triton::Allocator	49
Triton::MemObject	75
Triton::Environment	50
Triton::Matrix3	67
Triton::Matrix4	71
Triton::Ocean	77
Triton::RandomNumberGenerator	85
Triton::ResourceLoader	87
Triton::Vector3	92
Triton::Vector3f	96
Triton::Vector4	99
Triton::WakeGenerator	101
Triton::WindFetch	104
Triton::Utils	90

Chapter 18

Class Index

18.1 Class List

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

Triton::Allocator (You may extend the Allocator class to hook your own memory management scheme into Triton)	49
Triton::Environment (Triton's public interface for specifying the environmental conditions and camera properties)	50
Triton::Matrix3 (A simple 3x3 matrix class and its operations)	67
Triton::Matrix4 (An implementation of a 4x4 matrix and some simple operations on it)	71
Triton::MemObject (This base class for all Triton objects intercepts the new and delete operators, routing them through Triton::Allocator()) . . .	75
Triton::Ocean (Allows you to configure and draw Triton's water simulation) .	77
Triton::RandomNumberGenerator (An interface for generating random numbers in Triton)	85
Triton::ResourceLoader (This class is used whenever Triton needs to load textures, data files, or shaders from mass storage; you may extend this class to override our default use of POSIX filesystem calls with your own resource management if you wish)	87
Triton::Utils (A collection of static utility methods used by Triton)	90
Triton::Vector3 (A 3D double-precision Vector class and its operations) . . .	92
Triton::Vector3f (A 3D single-precision vector class, and its operations) . . .	96
Triton::Vector4 (A simple double-precision 4D vector class with no operations defined)	99
Triton::WakeGenerator (A WakeGenerator represents an object on the water that generates a wake as it moves, such as a ship)	101
Triton::WindFetch (A localized or global area of wind of given speed and direction)	104

Chapter 19

File Index

19.1 File List

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

mainpage.h	??
C:/triton/trunk/Public Headers/ Environment.h (The public interface for setting Triton's environmental parameters)	109
C:/triton/trunk/Public Headers/ Matrix3.h (Implements a 3x3 matrix and its operations)	111
C:/triton/trunk/Public Headers/ Matrix4.h (An implementation of a 4x4 matrix and some simple operations on it)	113
C:/triton/trunk/Public Headers/ MemAlloc.h (Memory allocation interface for SilverLining)	115
C:/triton/trunk/Public Headers/ Ocean.h (Triton's Ocean model interface)	117
C:/triton/trunk/Public Headers/ RandomNumberGenerator.h (An interface for overriding Triton's generation of random numbers)	118
C:/triton/trunk/Public Headers/ ResourceLoader.h (A class for loading Triton's resources from mass storage, which you may extend)	120
C:/triton/trunk/Public Headers/ Triton.h (A convenience header that includes the main public headers for Triton)	122
C:/triton/trunk/Public Headers/ TritonCommon.h (Common typedefs and defines used within Triton)	123
C:/triton/trunk/Public Headers/ Vector3.h (A 3D Vector class and its operations)	125
C:/triton/trunk/Public Headers/ Vector4.h (A simple 4D vector class)	126
C:/triton/trunk/Public Headers/ WakeGenerator.h (An object that generates a ship Kelvin wake as it moves)	128
C:/triton/trunk/Public Headers/ WindFetch.h (A localized or global area of wind of given speed and direction)	130

Chapter 20

Class Documentation

20.1 Triton::Allocator Class Reference

You may extend the [Allocator](#) class to hook your own memory management scheme into Triton.

```
#include <MemAlloc.h>
```

Collaboration diagram for Triton::Allocator:



Public Member Functions

- virtual void *TRITONAPI [alloc](#) (size_t bytes)
Allocate a block of memory; defaults to malloc()
- virtual void TRITONAPI [dealloc](#) (void *p)
Free a block of memory; defaults to free()

Static Public Member Functions

- static [Allocator](#) *TRITONAPI [GetAllocator](#) ()

Retrieves the static allocator object.

- static void TRITONAPI [SetAllocator](#) ([Allocator](#) *a)

Sets a new static allocator object.

20.1.1 Detailed Description

You may extend the [Allocator](#) class to hook your own memory management scheme into Triton. Instantiate your own implementation of [Allocator](#), and pass it into [Allocator::SetAllocator](#) prior to calling any other Triton methods or instantiating any Triton objects.

Each object in Triton overloads the new and delete operators, and routes memory management through the [Allocator](#) as well.

20.1.2 Member Function Documentation

20.1.2.1 static void TRITONAPI Triton::Allocator::SetAllocator ([Allocator](#) * a) [inline, static]

Sets a new static allocator object.

If this is not called, the default implementation using malloc and free is used.

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

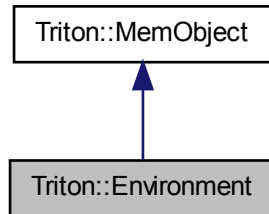
- C:/triton/trunk/Public Headers/[MemAlloc.h](#)

20.2 Triton::Environment Class Reference

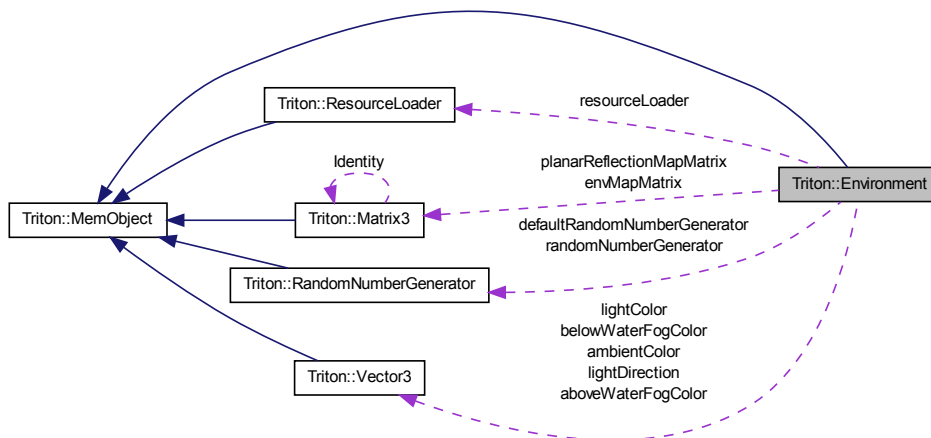
Triton's public interface for specifying the environmental conditions and camera properties.

```
#include <Environment.h>
```

Inheritance diagram for Triton::Environment:



Collaboration diagram for Triton::Environment:



Public Member Functions

- [Environment](#) ()
Constructor.
- EnvironmentError TRITONAPI [Initialize](#) (CoordinateSystem cs, Renderer ren, [ResourceLoader](#) *rl, void *device=NULL)
Initializes the environment prior to use.

- virtual `~Environment ()`
Virtual destructor.
- void TRITONAPI `SetLicenseCode (const char *userName, const char *registrationCode)`

Licensed users must call `SetLicenseCode` with your user name and registration code prior to using the `Environment` object.
- void TRITONAPI `SetRandomNumberGenerator (const RandomNumberGenerator *rng)`

Set a custom `RandomNumberGenerator` - derived random number generator, to override Triton's default use of `stdlib`'s `rand()` function.
- const `RandomNumberGenerator *TRITONAPI GetRandomNumberGenerator ()` const

Returns either the default `RandomNumberGenerator` used for all random numbers in Triton, or a custom subclass of `RandomNumberGenerator` that was passed in via `Environment::SetRandomNumberGenerator()`.
- `ResourceLoader *TRITONAPI GetResourceLoader ()` const

Retrieves the `ResourceLoader` object passed in to the `Environment()` constructor.
- void `*TRITONAPI GetDevice ()` const

Retrieves the DirectX device pointer passed in to the `Environment()` constructor, or NULL for OpenGL users.
- void TRITONAPI `SetDirectionalLight (const Vector3 &direction, const Vector3 &color)`

Sets the color and direction of directional light used to light the water, as from the sun or moon.
- void TRITONAPI `SetAmbientLight (const Vector3 &color)`

Sets the color of ambient light used to light the water, as from skylight.
- const `Vector3 &TRITONAPI GetLightDirection ()` const

Retrieves the vector toward the infinitely distant directional light source passed in via `SetDirectionalLight()`.
- const `Vector3 &TRITONAPI GetDirectionalLightColor ()` const

Retrieves the RGB color of the directional light source passed in via `SetDirectionalLight()`.
- const `Vector3 &TRITONAPI GetAmbientLightColor ()` const

Retrieves the RGB color of the ambient light passed in via `SetAmbientLight()`.
- void TRITONAPI `SetEnvironmentMap (TextureHandle cubeMap, const Matrix3 &textureMatrix=Matrix3::Identity)`

Passes in an optional environment cube map used for rendering reflections in the water.

- TextureHandle TRITONAPI [GetEnvironmentMap](#) () const
Retrieves the environment cube map passed in via [SetEnvironmentMap\(\)](#), which may be a `GLuint`, `LPDIRECT3DCUBETEXTURE9`, or `ID3D11ShaderResourceView` depending on the renderer being used.*
- [Matrix3](#) TRITONAPI [GetEnvironmentMapMatrix](#) () const
Retrieves the texture matrix used to transform the environment map lookups at runtime, which was optionally passed in via [SetEnvironmentMap\(\)](#).
- void TRITONAPI [SetPlanarReflectionMap](#) (TextureHandle textureMap, const [Matrix3](#) &textureMatrix, float normalDisplacementScale=0.125f)
Passes in an optional planar reflection map used for rendering local reflections in the water.
- TextureHandle TRITONAPI [GetPlanarReflectionMap](#) () const
Retrieves the environment cube map passed in via [SetPlanarReflectionMap\(\)](#), which may be a `GLuint`, `LPDIRECT3DTEXTURE9`, or `ID3D11ShaderResourceView` depending on the renderer being used.*
- [Matrix3](#) TRITONAPI [GetPlanarReflectionMapMatrix](#) () const
Retrieves the texture matrix used to transform the planar reflection map lookups at runtime, which was passed in via [SetPlanarReflectionMap\(\)](#).
- float TRITONAPI [GetPlanarReflectionDisplacementScale](#) () const
Retrieves normal displacement scale set for planar reflections via [SetPlanarReflectionMap\(\)](#).
- void TRITONAPI [SimulateSeaState](#) (double beaufortScale, double windDirection)
Simulates a specific sea state on the Beaufort scale, by clearing out any existing wind fetches passed into the [Environment](#) and setting up a new one consistent with the state specified.
- void TRITONAPI [AddWindFetch](#) (const [WindFetch](#) &fetch)
Adds a [Triton::WindFetch](#) to the environment, which specifies an area of wind of a given speed and direction, which may or may not be localized.
- void TRITONAPI [ClearWindFetches](#) ()
Removes all wind fetches or sea state simulations from the simulated environment, resulting in a perfectly calm sea.
- void TRITONAPI [GetWind](#) (const [Vector3](#) &pos, double &windSpeed, double &windDirection) const
Computes the wind conditions at a given location, by evaluating all [WindFetch](#) objects that include the position.

- void TRITONAPI [SetSeaLevel](#) (double altitudeMSL)
If you want to change the mean sea level from a height of 0 in flat-earth coordinates, or from the WGS84 ellipsoid in geocentric coordinates, you may do so here.
- double TRITONAPI [GetSeaLevel](#) () const
Returns the offset for the mean sea level previously set by [SetSeaLevel\(\)](#).
- void TRITONAPI [SetAboveWaterVisibility](#) (double visibility, const [Vector3](#) &fog-Color)
Sets the simulated atmospheric visibility above the water, used to fog out the surface of the water when viewed from above.
- void TRITONAPI [GetAboveWaterVisibility](#) (double &visibility, [Vector3](#) &fog-Color) const
Retrieves the above-water visibility settings previously set with [SetAboveWaterVisibility\(\)](#).
- void TRITONAPI [SetBelowWaterVisibility](#) (double visibility, const [Vector3](#) &fog-Color)
Sets the simulated atmospheric visibility below the water, used to fog out the surface of the water when viewed from below.
- void TRITONAPI [GetBelowWaterVisibility](#) (double &visibility, [Vector3](#) &fog-Color) const
Retrieves the below-water visibility settings previously set with [SetBelowWaterVisibility\(\)](#).
- void TRITONAPI [SetWorldUnits](#) (double worldUnits)
Sets the size of one world unit in meters.
- double TRITONAPI [GetWorldUnits](#) () const
Retrieves the size of one world unit, in meters.
- CoordinateSystem TRITONAPI [GetCoordinateSystem](#) () const
Returns the [CoordinateSystem](#) passed into the [Environment\(\)](#) constructor, indicating the up vector and the presence of a geocentric or flat coordinate system.
- bool TRITONAPI [IsGeocentric](#) () const
Returns whether the [CoordinateSystem](#) passed into the [Environment\(\)](#) constructor is geocentric, indicating an elliptical or spherical coordinate system where all points are relative to the center of the Earth.
- [Renderer](#) TRITONAPI [GetRenderer](#) () const
Returns the [Renderer](#) specified in the [Environment\(\)](#) constructor, telling you what flavor of OpenGL or DirectX is being used to render Triton's water.
- bool TRITONAPI [IsOpenGL](#) () const

Returns whether the *Renderer* specified in the *Environment()* constructor is an *OpenGL* *renderer*.

- `bool TRITONAPI IsDirectX () const`

Returns whether the *Renderer* specified in the *Environment()* constructor is a *DirectX* *renderer*.
- `void TRITONAPI SetCameraMatrix (const double *m)`

Sets the *modelview* matrix used for rendering the *ocean*; this must be called every frame prior to calling *Ocean::Draw()* if your camera orientation or position changes.
- `void TRITONAPI SetProjectionMatrix (const double *p)`

Sets the *projection* matrix used for rendering the *ocean*; this must be called every frame prior to calling *Ocean::Draw()*.
- `const double *TRITONAPI GetCameraMatrix () const`

Retrieves an array of 16 doubles representing the *modelview* matrix passed in via *SetCameraMatrix()*.
- `const double *TRITONAPI GetProjectionMatrix () const`

Retrieves an array of 16 doubles representing the *projection* matrix passed in via *SetProjectionMatrix()*.
- `const double *TRITONAPI GetCameraPosition () const`

Retrieves an array of 3 doubles representing the *X*, *Y*, and *Z* position of the camera, extracted from the *modelview* matrix passed in via *SetCameraMatrix()*.
- `Vector3 TRITONAPI GetUpVector () const`

Retrieves a normalized vector pointing "up", based on the coordinate system specified in *Environment::Initialize()* and the current position from the *modelview* matrix passed in through *Environment::SetCameraMatrix()*.
- `Vector3 TRITONAPI GetRightVector () const`

Retrieves a normalized vector pointing "right", based on the coordinate system specified in *Environment::Initialize()* and the current position from the *modelview* matrix passed in through *Environment::SetCameraMatrix()*.
- `void TRITONAPI SetConfigOption (const char *key, const char *value)`

Sets a configuration setting (defaults in *resources/triton.config*.) Many settings are read at initialization, so call this as early as possible after initializing the *Environment*.
- `const char *TRITONAPI GetConfigOption (const char *key)`

Retrieves the configuration setting for the given configuration key, as set in *resources/triton.config* or via *SetConfigOption()*.
- `bool CullSphere (const Vector3 &position, double radius) const`

Returns true if the given sphere lies within the view frustum, as defined by the modelview - projection matrix passed in via [SetCameraMatrix\(\)](#) and [SetProjectionMatrix\(\)](#).

20.2.1 Detailed Description

Triton's public interface for specifying the environmental conditions and camera properties. The [Ocean](#) constructor requires an [Environment](#) object, so you'll need to create this first.

20.2.2 Constructor & Destructor Documentation

20.2.2.1 Triton::Environment::Environment ()

Constructor.

20.2.2.2 virtual Triton::Environment::~~Environment () [virtual]

Virtual destructor.

20.2.3 Member Function Documentation

20.2.3.1 void TRITONAPI Triton::Environment::AddWindFetch (const WindFetch & fetch)

Adds a [Triton::WindFetch](#) to the environment, which specifies an area of wind of a given speed and direction, which may or may not be localized.

This [WindFetch](#) will be added to any other wind passed in previously, unless [ClearWindFetches\(\)](#) is called first.

All waves in Triton are a result of simulated wind conditions. Without wind, there will be no waves. Stronger winds traveling across longer distances will result in higher waves.

Parameters

<i>fetch</i>	The WindFetch to add to the simulated environment.
--------------	--

20.2.3.2 void TRITONAPI Triton::Environment::ClearWindFetches ()

Removes all wind fetches or sea state simulations from the simulated environment, resulting in a perfectly calm sea.

Call [AddWindFetch\(\)](#) or [SimulateSeaState\(\)](#) to add wind back in and generate waves as a result.

20.2.3.3 `bool Triton::Environment::CullSphere (const Vector3 & position, double radius) const`

Returns true if the given sphere lies within the view frustum, as defined by the modelview - projection matrix passed in via [SetCameraMatrix\(\)](#) and [SetProjectionMatrix\(\)](#).

Parameters

<i>position</i>	The center of the sphere in world coordinates.
<i>radius</i>	The radius of the sphere in world coordinates.

Returns

True if the sphere is not visible and should be culled.

20.2.3.4 `void TRITONAPI Triton::Environment::GetAboveWaterVisibility (double & visibility, Vector3 & fogColor) const [inline]`

Retrieves the above-water visibility settings previously set with [SetAboveWaterVisibility\(\)](#).

Parameters

<i>visibility</i>	Receives the visibility, in world units, above the water.
<i>fogColor</i>	Receives the fog color, in normalized RGB units.

20.2.3.5 `const Vector3& TRITONAPI Triton::Environment::GetAmbientLightColor () const [inline]`

Retrieves the RGB color of the ambient light passed in via [SetAmbientLight\(\)](#).

20.2.3.6 `void TRITONAPI Triton::Environment::GetBelowWaterVisibility (double & visibility, Vector3 & fogColor) const [inline]`

Retrieves the below-water visibility settings previously set with [SetBelowWaterVisibility\(\)](#).

Parameters

<i>visibility</i>	Receives the visibility, in world units, above the water.
<i>fogColor</i>	Receives the fog color, in normalized RGB units.

20.2.3.7 `const double* TRITONAPI Triton::Environment::GetCameraMatrix () const`
`[inline]`

Retrieves an array of 16 doubles representing the modelview matrix passed in via [SetCameraMatrix\(\)](#).

20.2.3.8 `const double* TRITONAPI Triton::Environment::GetCameraPosition () const`
`[inline]`

Retrieves an array of 3 doubles representing the X, Y, and Z position of the camera, extracted from the modelview matrix passed in via [SetCameraMatrix\(\)](#).

20.2.3.9 `const char* TRITONAPI Triton::Environment::GetConfigOption (const char * key)`

Retrieves the configuration setting for the given configuration key, as set in resources/triton.config or via [SetConfigOption\(\)](#).

Parameters

<i>key</i>	The configuration key to retrieve
------------	-----------------------------------

Returns

The value of the key specified, or NULL if not found.

20.2.3.10 `CoordinateSystem TRITONAPI Triton::Environment::GetCoordinateSystem () const`
`[inline]`

Returns the `CoordinateSystem` passed into the [Environment\(\)](#) constructor, indicating the up vector and the presence of a geocentric or flat coordinate system.

20.2.3.11 `void* TRITONAPI Triton::Environment::GetDevice () const` `[inline]`

Retrieves the DirectX device pointer passed in to the [Environment\(\)](#) constructor, or NULL for OpenGL users.

20.2.3.12 `const Vector3& TRITONAPI Triton::Environment::GetDirectionalLightColor () const`
`[inline]`

Retrieves the RGB color of the directional light source passed in via [SetDirectionalLight\(\)](#).

20.2.3.13 `TextureHandle TRITONAPI Triton::Environment::GetEnvironmentMap () const`
[inline]

Retrieves the environment cube map passed in via [SetEnvironmentMap\(\)](#), which may be a `GLuint`, `LPDIRECT3DCUBETEXTURE9`, or `ID3D11ShaderResourceView*` depending on the renderer being used.

20.2.3.14 `Matrix3 TRITONAPI Triton::Environment::GetEnvironmentMapMatrix () const`
[inline]

Retrieves the texture matrix used to transform the environment map lookups at runtime, which was optionally passed in via [SetEnvironmentMap\(\)](#).

20.2.3.15 `const Vector3& TRITONAPI Triton::Environment::GetLightDirection () const`
[inline]

Retrieves the vector toward the infinitely distant directional light source passed in via [SetDirectionalLight\(\)](#).

20.2.3.16 `float TRITONAPI Triton::Environment::GetPlanarReflectionDisplacementScale () const`
[inline]

Retrieves normal displacement scale set for planar reflections via [SetPlanarReflectionMap\(\)](#).

20.2.3.17 `TextureHandle TRITONAPI Triton::Environment::GetPlanarReflectionMap () const`
[inline]

Retrieves the environment cube map passed in via [SetPlanarReflectionMap\(\)](#), which may be a `GLuint`, `LPDIRECT3DTEXTURE9`, or `ID3D11ShaderResourceView*` depending on the renderer being used.

20.2.3.18 `Matrix3 TRITONAPI Triton::Environment::GetPlanarReflectionMapMatrix () const`
[inline]

Retrieves the texture matrix used to transform the planar reflection map lookups at runtime, which was passed in via [SetPlanarReflectionMap\(\)](#).

20.2.3.19 `const double* TRITONAPI Triton::Environment::GetProjectionMatrix () const`
[inline]

Retrieves an array of 16 doubles representing the projection matrix passed in via [SetProjectionMatrix\(\)](#).

20.2.3.20 `const RandomNumberGenerator* TRITONAPI Triton::Environment::GetRandomNumberGenerator () const [inline]`

Returns either the default [RandomNumberGenerator](#) used for all random numbers in Triton, or a custom subclass of [RandomNumberGenerator](#) that was passed in via [Environment::SetRandomNumberGenerator\(\)](#).

Returns

The [RandomNumberGenerator](#) instance in use by Triton.

20.2.3.21 `Renderer TRITONAPI Triton::Environment::GetRenderer () const [inline]`

Returns the [Renderer](#) specified in the [Environment\(\)](#) constructor, telling you what flavor of OpenGL or DirectX is being used to render Triton's water.

20.2.3.22 `ResourceLoader* TRITONAPI Triton::Environment::GetResourceLoader () const [inline]`

Retrieves the [ResourceLoader](#) object passed in to the [Environment\(\)](#) constructor.

20.2.3.23 `Vector3 TRITONAPI Triton::Environment::GetRightVector () const`

Retrieves a normalized vector pointing "right", based on the coordinate system specified in [Environment::Initialize\(\)](#) and the current position from the modelview matrix passed in through [Environment::SetCameraMatrix\(\)](#).

20.2.3.24 `double TRITONAPI Triton::Environment::GetSeaLevel () const [inline]`

Returns the offset for the mean sea level previously set by [SetSeaLevel\(\)](#).

Returns

The offset in world units that the mean sea level is displaced by.

20.2.3.25 `Vector3 TRITONAPI Triton::Environment::GetUpVector () const`

Retrieves a normalized vector pointing "up", based on the coordinate system specified in [Environment::Initialize\(\)](#) and the current position from the modelview matrix passed in through [Environment::SetCameraMatrix\(\)](#).

20.2.3.26 `void TRITONAPI Triton::Environment::GetWind (const Vector3 & pos, double & windSpeed, double & windDirection) const`

Computes the wind conditions at a given location, by evaluating all [WindFetch](#) objects that include the position.

See also

[AddWindFetch\(\)](#)
[SimulateSeaState\(\)](#)

Parameters

<i>pos</i>	A const reference to the position at which wind conditions should be computed.
<i>windSpeed</i>	A reference to a double that will receive the overall wind speed at this location, in world units per second.
<i>windDirection</i>	A reference to a double that will receive the overall wind direction at this location, in radians.

20.2.3.27 `double TRITONAPI Triton::Environment::GetWorldUnits () const [inline]`

Retrieves the size of one world unit, in meters.

See also

[SetWorldUnits\(\)](#)

20.2.3.28 `EnvironmentError TRITONAPI Triton::Environment::Initialize (CoordinateSystem cs, Renderer ren, ResourceLoader * rl, void * device = NULL)`

Initializes the environment prior to use.

Parameters

<i>cs</i>	The CoordinateSystem your application is using, which may be a geocentric system based on an elliptical WGS84 or spherical earth model, or a flat cartesian model, with up pointing along either the Y or Z axes.
<i>ren</i>	Specifies the version of OpenGL or DirectX your application is using. Triton will render the ocean using the same graphics subsystem.
<i>rl</i>	A ResourceLoader object that is used by Triton for loading its graphics, shader, and configuration resources. This may be an instance of Triton's default ResourceLoader class that loads loose files in Triton's resources directory directly from disk, or your own derived class that handles resource management in some other way.
<i>device</i>	Unused for OpenGL contexts. For DirectX users, this must be a pointer to your valid and initialized <code>DIRECT3DDEVICE9</code> or <code>ID3D11Device</code> .

Returns

An error code if the environment failed to initialize, in which case you can't use this environment. To receive more details on why it failed, enable the setting `enable-debug-messages` in `resources/triton.config` which will send more info to your debugger's output. If initialization succeeded, you'll get back `SUCCEEDED`.

20.2.3.29 `bool TRITONAPI Triton::Environment::IsDirectX () const` [inline]

Returns whether the Renderer specified in the [Environment\(\)](#) constructor is a DirectX renderer.

20.2.3.30 `bool TRITONAPI Triton::Environment::IsGeocentric () const` [inline]

Returns whether the CoordinateSystem passed into the [Environment\(\)](#) constructor is geocentric, indicating an elliptical or spherical coordinate system where all points are relative to the center of the Earth.

20.2.3.31 `bool TRITONAPI Triton::Environment::IsOpenGL () const` [inline]

Returns whether the Renderer specified in the Environment() constructor is an OpenGL renderer.

20.2.3.32 `void TRITONAPI Triton::Environment::SetAboveWaterVisibility (double visibility, const Vector3 & fogColor)` [inline]

Sets the simulated atmospheric visibility above the water, used to fog out the surface of the water when viewed from above.

You may use this method to fog the ocean consistently with other objects in your scene.

The visibility specified will be transformed into an exponential fog extinction value using the Koschmieder equation: $visibility = 3.912 / extinction$

Parameters

<i>visibility</i>	The visibility, in world units, above the water.
<i>fogColor</i>	The fog color, in normalized RGB units.

20.2.3.33 `void TRITONAPI Triton::Environment::SetAmbientLight (const Vector3 & color)` [inline]

Sets the color of ambient light used to light the water, as from skylight.

Parameters

<i>color</i>	the RGB color of the ambient light.
--------------	-------------------------------------

20.2.3.34 `void TRITONAPI Triton::Environment::SetBelowWaterVisibility (double visibility, const Vector3 & fogColor)` [inline]

Sets the simulated atmospheric visibility below the water, used to fog out the surface of the water when viewed from below.

You may use this method to fog the ocean consistently with other objects in your scene. While underwater, the application is responsible for clearing the back buffer to the fog color to match the color passed in here.

The visibility specified will be transformed into an exponential fog extinction value using the Koschmieder equation: $\text{visibility} = 3.912 / \text{extinction}$

Parameters

<i>visibility</i>	The visibility, in world units, below the water.
<i>fogColor</i>	The fog color, in normalized RGB units.

20.2.3.35 void TRITONAPI Triton::Environment::SetCameraMatrix (const double * *m*)

Sets the modelview matrix used for rendering the ocean; this must be called every frame prior to calling [Ocean::Draw\(\)](#) if your camera orientation or position changes.

Parameters

<i>m</i>	A pointer to 16 doubles representing a 4x4 modelview matrix.
----------	--

20.2.3.36 void TRITONAPI Triton::Environment::SetConfigOption (const char * *key*, const char * *value*)

Sets a configuration setting (defaults in resources/triton.config.) Many settings are read at initialization, so call this as early as possible after initializing the [Environment](#).

Parameters

<i>key</i>	The configuration entry name to modify
<i>value</i>	The value to set this entry to.

20.2.3.37 void TRITONAPI Triton::Environment::SetDirectionalLight (const Vector3 & *direction*, const Vector3 & *color*)

Sets the color and direction of directional light used to light the water, as from the sun or moon.

Parameters

<i>direction</i>	A normalized vector pointing toward the infinitely distant light source.
<i>color</i>	The RGB color of the light.

20.2.3.38 void TRITONAPI Triton::Environment::SetEnvironmentMap (TextureHandle *cubeMap*, const Matrix3 & *textureMatrix* = Matrix3::Identity) [inline]

Passes in an optional environment cube map used for rendering reflections in the water.

If unused, Triton will instead reflect a constant color based on the ambient light passed in via [SetAmbientLight\(\)](#). The caller is responsible for releasing or deleting this resource at shutdown.

See also

[SetEnvironmentMap\(\)](#)

Parameters

<i>cubeMap</i>	A cube map texture resource, which should be cast to a TextureHandle. Under OpenGL, this must be a GLuint indicating the ID of the GL_TEXTURE_CUBE_MAP returned from glGenTextures. Under DirectX9, this must be a LPDIRECT3DCUBETEXTURE9. Under DirectX11, this must be a ID3D11ShaderResourceView pointer with an underlying ViewDimension of D3D11_SRV_DIMENSION_TEXTURECUBE.
<i>textureMatrix</i>	An optional texture matrix used to transform the 3D coordinates used to access the cube map. If your cube map isn't oriented with the same cartesian axes used by your simulation, you can use this parameter to account for any differences in your cube map's coordinate system and your simulation's.

20.2.3.39 void TRITONAPI Triton::Environment::SetLicenseCode (const char * *userName*, const char * *registrationCode*)

Licensed users must call SetLicenseCode with your user name and registration code prior to using the [Environment](#) object.

Visit <http://www.sundog-soft.com/> to purchase a license. If you don't call SetLicenseCode or pass invalid parameters to it, Triton will run in evaluation mode, which will terminate your application after five minutes of runtime.

Parameters

<i>userName</i>	The user name given to you with your license purchase.
<i>registrationCode</i>	The registration code given to you with your license purchase.

20.2.3.40 void TRITONAPI Triton::Environment::SetPlanarReflectionMap (TextureHandle *textureMap*, const Matrix3 & *textureMatrix*, float *normalDisplacementScale* = 0.125f) [inline]

Passes in an optional planar reflection map used for rendering local reflections in the water.

Triton can use planar reflection map & environment map together. Alpha channel in

planar reflection map is used to blend between planar reflection & environment map. If planar reflection is not used Triton falls back to environment map (it is the same result as if planar reflection had 0 on alpha in every texel).

See also

[SetEnvironmentMap\(\)](#)

Parameters

<i>textureMap</i>	A 2D map texture resource, which should be cast to a TextureHandle. Under OpenGL, this must be a GLuint indicating the ID of the GL_TEXTURE_2D returned from glGenTextures. Under DirectX9, this must be a LPDIRECT3DTEXTURE9. Under DirectX11, this must be a ID3D11ShaderResourceView pointer with an underlying ViewDimension of D3D11_SRV_DIMENSION_TEXTURE2D.
<i>textureMatrix</i>	A required texture matrix used to project the vector computed by triton to reflection map texture coordinates. Triton "Input" is a view vector perturbed by normal.xy components. Such vector approximates wave reflection wiggle and can be used to directly access planar reflection map. See description of parameter normalDisplacementScale to learn why reflection vector cannot be used directly to access the planar reflection map. Input Vector passed to textureMatrix will be defined in world space coordinates translated to view point. In other words this coordinate space has the same orientation as world space but its origin (point 0,0,0) is moved to camera location. This is the same coordinate space that env map projection uses. Advanced users may see how reflection (P) variable is handled in Triton pixel (fragment) shaders.
<i>normalDisplacementScale</i>	A scale factor used to perturb vertex by normal.xy to get more realistic reflection from rough waves. Range of reasonable values is 0..4. Default is 0.125 Realtime bumpy surface reflection approaches are usually based on approximation of reflection from flat surface. However, method of reflection based on planar projection has serious limitation. It assumes that view vector (and reflection vector) angle strictly corresponds to the incident point on the surface where vector was reflected. If surface is not perfectly flat and water waves are of course an example of such surface, above assumption fails and many points at the ocean surface can reflect vectors at the same direction. If such reflection was used to address a planar map texel we could see reflections of the objects at random points on the surface. To avoid such effect, usually some limits are imposed on reflected vector or reflected texture coords. Triton adopts classic approach to the above problem which works by actually using a view vector perturbed by an offset computed from normal.xy scaled by normalDisplacementScale. Since normal.xy components are never larger than unit value we can be sure that reflection vector will fit in finite margin defined by normalDisplacementScale.

20.2.3.41 void TRITONAPI Triton::Environment::SetProjectionMatrix (const double * p)

Sets the projection matrix used for rendering the ocean; this must be called every frame prior to calling [Ocean::Draw\(\)](#).

Parameters

<i>p</i>	A pointer to 16 doubles representing a 4x4 projection matrix.
----------	---

20.2.3.42 void TRITONAPI Triton::Environment::SetRandomNumberGenerator (const RandomNumberGenerator * rng)

Set a custom [RandomNumberGenerator](#) - derived random number generator, to override Triton's default use of `stdlib's rand()` function.

This may be useful for ensuring deterministic behavior across channels (although a simpler approach may be calling `srand()` with a consistent seed from your application.) If this method is not called, a default random number generator will be used automatically.

Parameters

<i>rng</i>	An instance of a class derived from RandomNumberGenerator that will handle all random number generation within Triton.
------------	--

**20.2.3.43 void TRITONAPI Triton::Environment::SetSeaLevel (double altitudeMSL)
[inline]**

If you want to change the mean sea level from a height of 0 in flat-earth coordinates, or from the WGS84 ellipsoid in geocentric coordinates, you may do so here.

See also

[GetSeaLevel\(\)](#)

Parameters

<i>altitudeMSL</i>	The offset in world units to displace mean sea level by.
--------------------	--

**20.2.3.44 void TRITONAPI Triton::Environment::SetWorldUnits (double worldUnits)
[inline]**

Sets the size of one world unit in meters.

By default, one world unit is assumed to mean one meter. If this is not the case for your coordinate system, be sure to call [SetWorldUnits\(\)](#) immediately after instantiating your [Environment](#) class.

20.2.3.45 void TRITONAPI Triton::Environment::SimulateSeaState (double *beaufortScale*, double *windDirection*)

Simulates a specific sea state on the Beaufort scale, by clearing out any existing wind fetches passed into the [Environment](#) and setting up a new one consistent with the state specified.

Any subsequent calls to [AddWindFetch\(\)](#) will create wind additive to that created for the given sea state, so be sure to call [ClearWindFetches\(\)](#) if you intend to mix and match calls to [SimulateSeaState\(\)](#) and [AddWindFetch\(\)](#).

See http://en.wikipedia.org/wiki/Beaufort_scale for detailed descriptions of Beaufort numbers and the wave conditions they specify. At a high level,

0: Calm 1: Light air 2: Light breeze 3: Gentle breeze 4: Moderate breeze 5: Fresh breeze 6: Strong breeze 7: High wind 8: Gale 9: Storm 10: Strong Storm 11: Violent Storm 12: Hurricane

Parameters

<i>beaufortScale</i>	The Beaufort scale number specifying the desired wind and sea conditions. This may be a floating point value.
<i>windDirection</i>	The direction of the wind, specified in radians.

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

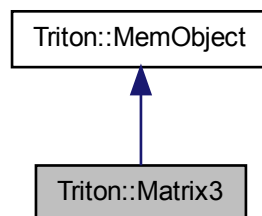
- C:/triton/trunk/Public Headers/[Environment.h](#)

20.3 Triton::Matrix3 Class Reference

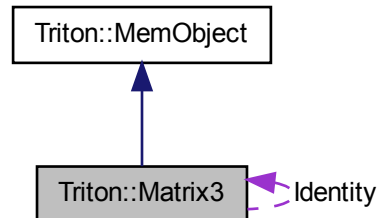
A simple 3x3 matrix class and its operations.

```
#include <Matrix3.h>
```

Inheritance diagram for Triton::Matrix3:



Collaboration diagram for Triton::Matrix3:



Public Member Functions

- [Matrix3](#) ()
Default constructor; performs no initialization for efficiency.
- [Matrix3](#) (double e11, double e12, double e13, double e21, double e22, double e23, double e31, double e32, double e33)
Constructor that instantiates the 3x3 matrix with initial values.
- [Matrix3](#) (double *m)
Constructor that takes an array of 9 doubles in row-major order.
- [~Matrix3](#) ()
Destructor.
- float *TRITONAPI [ToFloatArray](#) ()
Returns a static 3x3 float array in row major order.
- void TRITONAPI [FromRx](#) (double rad)
Populates the matrix to model a rotation about the X axis by a given amount, in radians.
- void TRITONAPI [FromRy](#) (double rad)
Populates the matrix to model a rotation about the Y axis by a given amount, in radians.
- void TRITONAPI [FromRz](#) (double rad)
Populates the matrix to model a rotation about the Z axis by a give amount, in radians.
- void TRITONAPI [FromXYZ](#) (double Rx, double Ry, double Rz)

Populates the matrix as a series of rotations about the X, Y, and Z axes (in that order) by specified amounts in radians.

- **Matrix3** TRITONAPI **operator*** (const **Matrix3** &mat)
Multiplies two matrices together.
- **Vector3** TRITONAPI **operator*** (const **Vector3** &rkVector) const
Multiplies the matrix by a vector, yielding another 3x1 vector.
- **Matrix3** TRITONAPI **Transpose** () const
Calculate the inverse of the matrix.

Public Attributes

- double **elem** [3][3]
The data members are public for convenience.

Friends

- **Vector3** TRITONAPI **operator*** (const **Vector3** &vec, const **Matrix3** &mat)
Multiplies a 1x3 vector by a matrix, yielding a 1x3 vector.

20.3.1 Detailed Description

A simple 3x3 matrix class and its operations.

20.3.2 Constructor & Destructor Documentation

20.3.2.1 Triton::Matrix3::Matrix3 () [inline]

Default constructor; performs no initialization for efficiency.

20.3.2.2 Triton::Matrix3::Matrix3 (double e11, double e12, double e13, double e21, double e22, double e23, double e31, double e32, double e33) [inline]

Constructor that instantiates the 3x3 matrix with initial values.

20.3.2.3 Triton::Matrix3::Matrix3 (double * m) [inline]

Constructor that takes an array of 9 doubles in row-major order.

20.3.2.4 Triton::Matrix3::~~Matrix3 () [inline]

Destructor.

20.3.3 Member Function Documentation

20.3.3.1 void TRITONAPI Triton::Matrix3::FromRx (double *rad*)

Populates the matrix to model a rotation about the X axis by a given amount, in radians.

20.3.3.2 void TRITONAPI Triton::Matrix3::FromRy (double *rad*)

Populates the matrix to model a rotation about the Y axis by a given amount, in radians.

20.3.3.3 void TRITONAPI Triton::Matrix3::FromRz (double *rad*)

Populates the matrix to model a rotation about the Z axis by a give amount, in radians.

20.3.3.4 void TRITONAPI Triton::Matrix3::FromXYZ (double *Rx*, double *Ry*, double *Rz*)

Populates the matrix as a series of rotations about the X, Y, and Z axes (in that order) by specified amounts in radians.

20.3.3.5 Vector3 TRITONAPI Triton::Matrix3::operator* (const Vector3 & *rkVector*) const

Multiplies the matrix by a vector, yielding another 3x1 vector.

20.3.3.6 Matrix3 TRITONAPI Triton::Matrix3::operator* (const Matrix3 & *mat*)

Multiplies two matrices together.

20.3.3.7 float* TRITONAPI Triton::Matrix3::ToFloatArray () [inline]

Returns a static 3x3 float array in row major order.

20.3.3.8 Matrix3 TRITONAPI Triton::Matrix3::Transpose () const

Calculate the inverse of the matrix.

20.3.4 Friends And Related Function Documentation

20.3.4.1 Vector3 TRITONAPI operator* (const Vector3 & *vec*, const Matrix3 & *mat*) [friend]

Multiplies a 1x3 vector by a matrix, yielding a 1x3 vector.

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

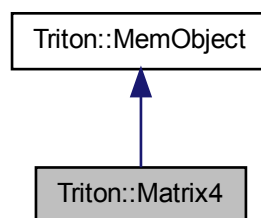
- C:/triton/trunk/Public Headers/[Matrix3.h](#)

20.4 Triton::Matrix4 Class Reference

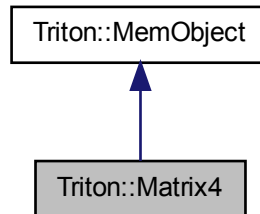
An implementation of a 4x4 matrix and some simple operations on it.

```
#include <Matrix4.h>
```

Inheritance diagram for Triton::Matrix4:



Collaboration diagram for Triton::Matrix4:



Public Member Functions

- [Matrix4](#) ()
Default constructor; initializes the matrix to an identity transform.
- [Matrix4](#) (double e11, double e12, double e13, double e14, double e21, double e22, double e23, double e24, double e31, double e32, double e33, double e34, double e41, double e42, double e43, double e44)
This constructor allows you to initialize the matrix as you please.
- [Matrix4](#) (const double *m)
Initializes the matrix from an array of 16 double-precision values (row-major).
- [~Matrix4](#) ()
Destructor.
- double TRITONAPI [operator](#)() (int x, int y) const
Retrieve a specific matrix element.
- float *TRITONAPI [ToFloatArray](#) ()
Populates a static array of 16 floats with the contents of the matrix.
- [Matrix4](#) TRITONAPI [operator*](#) (const [Matrix4](#) &mat) const
Multiplies two matrices together.
- [Vector4](#) TRITONAPI [operator*](#) (const [Vector4](#) &vec) const
Transform a point by the matrix.
- [Vector3](#) TRITONAPI [operator*](#) (const [Vector3](#) &vec) const

Transform a point by the matrix.

- void TRITONAPI [Transpose](#) ()
Transposes the matrix in-place.
- [Matrix4](#) TRITONAPI [InverseCramers](#) ()
Computes the inverse of the matrix using Cramer's rule.
- double *TRITONAPI [GetRow](#) (int row) const
Retrieves a pointer into the requested row of the matrix.

Public Attributes

- double [elem](#) [4][4]
Data members are public for convenience.

Friends

- [Vector4](#) TRITONAPI [operator*](#) (const [Vector4](#) &vec, const [Matrix4](#) &mat)
Multiplies a 1x3 vector by a matrix, yielding a 1x3 vector.

20.4.1 Detailed Description

An implementation of a 4x4 matrix and some simple operations on it.

20.4.2 Constructor & Destructor Documentation

20.4.2.1 [Triton::Matrix4::Matrix4](#) () `[inline]`

Default constructor; initializes the matrix to an identity transform.

20.4.2.2 [Triton::Matrix4::Matrix4](#) (double *e11*, double *e12*, double *e13*, double *e14*, double *e21*, double *e22*, double *e23*, double *e24*, double *e31*, double *e32*, double *e33*, double *e34*, double *e41*, double *e42*, double *e43*, double *e44*) `[inline]`

This constructor allows you to initialize the matrix as you please.

20.4.2.3 [Triton::Matrix4::Matrix4](#) (const double * *m*) `[inline]`

Initializes the matrix from an array of 16 double-precision values (row-major).

20.4.2.4 `Triton::Matrix4::~~Matrix4 () [inline]`

Destructor.

20.4.3 Member Function Documentation**20.4.3.1** `double* TRITONAPI Triton::Matrix4::GetRow (int row) const [inline]`

Retrieves a pointer into the requested row of the matrix.

20.4.3.2 `Matrix4 TRITONAPI Triton::Matrix4::InverseCramers () [inline]`

Computes the inverse of the matrix using Cramer's rule.

20.4.3.3 `Vector4 TRITONAPI Triton::Matrix4::operator* (const Vector4 & vec) const`

Transform a point by the matrix.

20.4.3.4 `Vector3 TRITONAPI Triton::Matrix4::operator* (const Vector3 & vec) const`

Transform a point by the matrix.

20.4.3.5 `Matrix4 TRITONAPI Triton::Matrix4::operator* (const Matrix4 & mat) const`

Multiplies two matrices together.

20.4.3.6 `float* TRITONAPI Triton::Matrix4::ToFloatArray () [inline]`

Populates a static array of 16 floats with the contents of the matrix.

20.4.3.7 `void TRITONAPI Triton::Matrix4::Transpose () [inline]`

Transposes the matrix in-place.

20.4.4 Friends And Related Function Documentation**20.4.4.1** `Vector4 TRITONAPI operator* (const Vector4 & vec, const Matrix4 & mat)
[friend]`

Multiplies a 1x3 vector by a matrix, yielding a 1x3 vector.

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

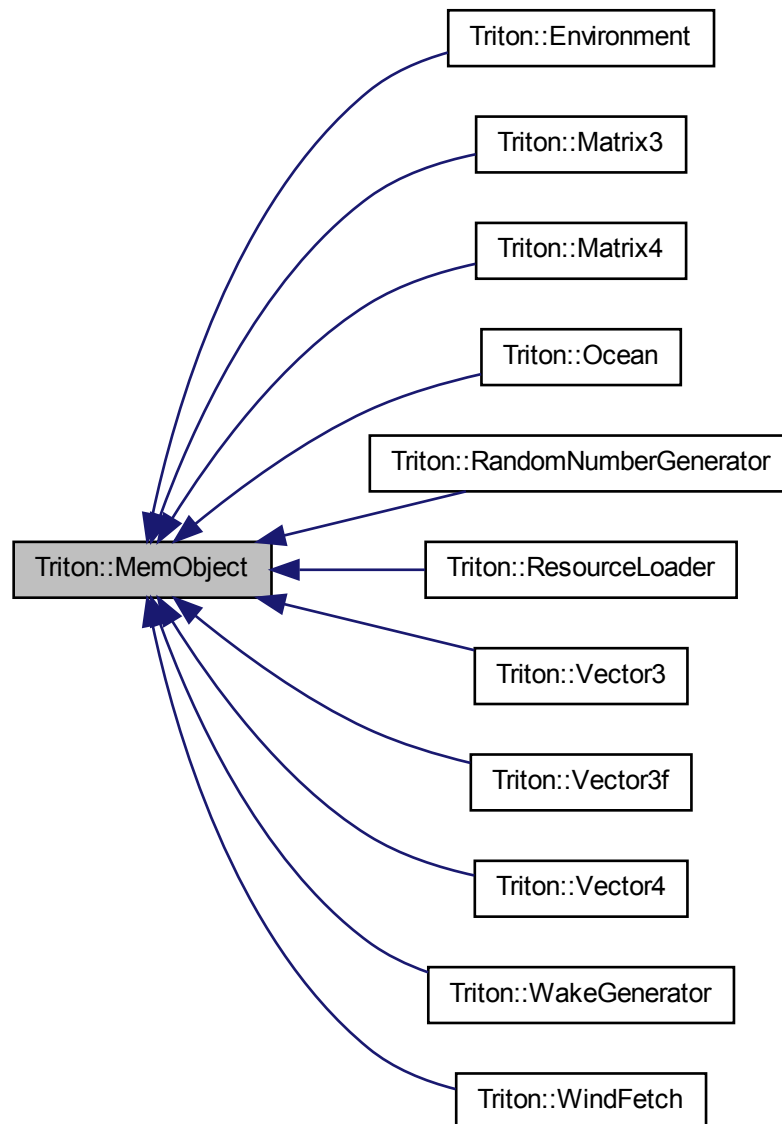
- [C:/triton/trunk/Public Headers/Matrix4.h](#)

20.5 Triton::MemObject Class Reference

This base class for all Triton objects intercepts the new and delete operators, routing them through Triton::Allocator().

```
#include <MemAlloc.h>
```

Inheritance diagram for Triton::MemObject:



20.5.1 Detailed Description

This base class for all Triton objects intercepts the new and delete operators, routing them through Triton::Allocator().

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

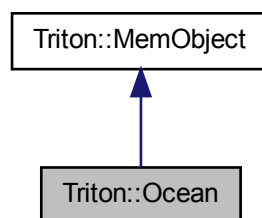
- [C:/triton/trunk/Public Headers/MemAlloc.h](#)

20.6 Triton::Ocean Class Reference

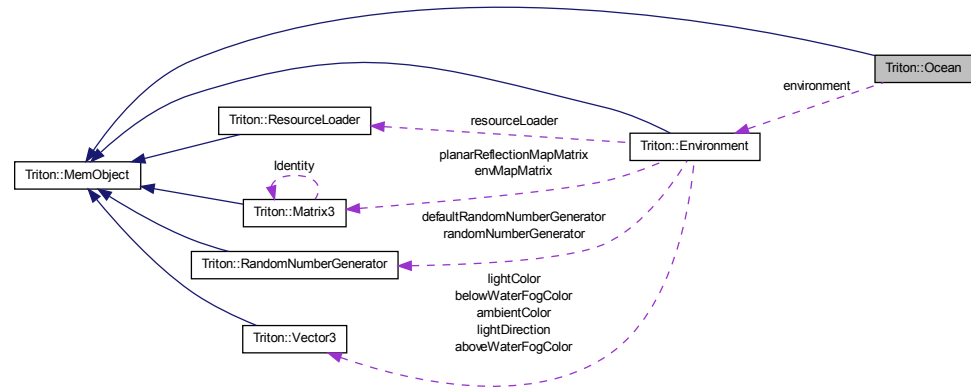
The [Ocean](#) class allows you to configure and draw Triton's water simulation.

```
#include <Ocean.h>
```

Inheritance diagram for Triton::Ocean:



Collaboration diagram for Triton::Ocean:



Public Member Functions

- virtual `~Ocean ()`
Virtual destructor.
- virtual void TRITONAPI `Draw` (double time, bool depthWrites=true)
Draws an infinite ocean surrounding the camera (as specified in the [Environment](#) object) for the simulated conditions at the given time.
- void TRITONAPI `D3D9DeviceLost ()`
DirectX9 users must call `D3D9DeviceLost()` in response to lost devices, prior to re-setting the device.
- void TRITONAPI `D3D9DeviceReset ()`
DirectX9 users must call `D3D9DeviceReset()` in response to device resets done in response to lost devices.
- float TRITONAPI `GetChoppiness ()` const
Retrieves the choppiness setting of the [Ocean](#), which controls how peaked the waves are.
- void TRITONAPI `SetChoppiness` (float chop)
Set the choppiness of the waves, which controls how peaked the waves are.
- float TRITONAPI `GetDepth` (Triton::Vector3 &floorNormal) const
Retrieves the simulated depth of the water in world units, and the surface normal of the sea floor, both at the current camera position.

- void TRITONAPI [SetDepth](#) (float depth, const [Triton::Vector3](#) &floorNormal)
Sets the simulated depth of the water in world units at the camera position, and the slope of the sea floor as specified by its surface normal at the camera position.
- void TRITONAPI [EnableWireframe](#) (bool wireframeOn)
Enables or disables wireframe rendering of the ocean's mesh.
- const char *TRITONAPI [GetFFTName](#) () const
Returns a description of the FFT transform being used, if a FFT water model is active.
- unsigned int TRITONAPI [GetNumTriangles](#) () const
Returns the number of triangles rendered by the underlying projected grid.
- ShaderHandle TRITONAPI [GetShaderObject](#) () const
Retrieves the underlying shader object used to render the water.
- bool TRITONAPI [GetHeight](#) (const [Vector3](#) &point, const [Vector3](#) &direction, float &height, [Vector3](#) &normal, bool visualCorrelation=true)
Retrieves the height and normal of the ocean surface at the intersection point of the given ray.
- float [GetWaveHeading](#) () const
Retrieves the wave direction.
- void [EnableSpray](#) (bool enable)
Enables or disables spray particle effects on breaking waves.
- bool [SprayEnabled](#) () const
Returns if spray particle effects on breaking waves are enabled, which they are by default.
- void [SetRefractionColor](#) (const [Vector3](#) &refractionColor)
Modifies the color used for refracted light rays that go into deep water.
- const [Vector3](#) & [GetRefractionColor](#) () const
Returns the color of light refracted into the water.

Static Public Member Functions

- static [Ocean](#) *TRITONAPI [Create](#) (const [Environment](#) *env, WaterModelTypes type=TESSENDORF, bool enableHeightTests=false)
Creates an [Ocean](#) instance tied to the given [Environment](#), using the specified wave model.

20.6.1 Detailed Description

The [Ocean](#) class allows you to configure and draw Triton's water simulation.

20.6.2 Constructor & Destructor Documentation

20.6.2.1 virtual Triton::Ocean::~~Ocean () [virtual]

Virtual destructor.

20.6.3 Member Function Documentation

20.6.3.1 static Ocean* TRITONAPI Triton::Ocean::Create (const Environment * env, WaterModelTypes type = TESSENDORF, bool enableHeightTests = false) [static]

Creates an [Ocean](#) instance tied to the given [Environment](#), using the specified wave model.

Parameters

<i>env</i>	A pointer to an Environment object created previously, which contains the environmental conditions, coordinate system, camera, and rendering system used by the Ocean . The caller is responsible for deleting this object after the Ocean is deleted.
<i>type</i>	Specifies whether a FFT-based ocean wave model (Tessendorf) should be used, or a simpler but less visually appealing Gerstner wave model that sums up individual waves in the spacial domain.
<i>enable-HeightTests</i>	Specifies whether the application will call Ocean::GetHeight() or not. If false, Triton may be able to keep the ocean simulation entirely on the GPU leading to better performance, but any calls to Ocean::GetHeight() may return 0. Set to true if you need to read back height information from the water surface.

Returns

An instance of [Ocean](#) that may be used for rendering. The caller is responsible for deleting this object when finished. NULL may be returned if the ocean could not initialize itself; in this case, enable the setting `enable-debug-messages` in `resource/triton.config` to get more details on what went wrong sent to your debugger output window. Contact support@sundog-soft.com with this output if necessary.

20.6.3.2 void TRITONAPI Triton::Ocean::D3D9DeviceLost ()

DirectX9 users must call [D3D9DeviceLost\(\)](#) in response to lost devices, prior to resetting the device.

A lost device may occur when the user locks and unlocks a system or changes the monitor resolution, and must be explicitly handled under DX9. Call [D3D9DeviceReset\(\)](#) once the device has been recreated.

20.6.3.3 void TRITONAPI Triton::Ocean::D3D9DeviceReset ()

DirectX9 users must call [D3D9DeviceReset\(\)](#) in response to device resets done in response to lost devices.

You must have called [D3D9DeviceLost\(\)](#) first in response to the lost device.

20.6.3.4 virtual void TRITONAPI Triton::Ocean::Draw (double *time*, bool *depthWrites* = true) [virtual]

Draws an infinite ocean surrounding the camera (as specified in the [Environment](#) object) for the simulated conditions at the given time.

Parameters

<i>time</i>	The simulated point in time to render, in seconds. Note that this is an absolute time which can be relative to any arbitrary point in time. It's not the delta time between frames.
<i>depthWrites</i>	Whether the ocean will write to the depth buffer. One technique for integrating the ocean with terrain is to draw the ocean first with depth writes disabled, then draw the terrain on top of it, thereby avoiding any depth buffer precision issues. Be sure to remove or disable any existing ocean surfaces from your terrain database first if using this technique.

20.6.3.5 void Triton::Ocean::EnableSpray (bool *enable*) [inline]

Enables or disables spray particle effects on breaking waves.

This does incur a performance penalty, so if you need faster performance, try disabling spray effects or even disable it entirely with the `fft-enable-spray` config setting in `resources/Triton.config`.

20.6.3.6 void TRITONAPI Triton::Ocean::EnableWireframe (bool *wireframeOn*)

Enables or disables wireframe rendering of the ocean's mesh.

Parameters

<i>wire-frameOn</i>	Set to true to render in wireframe mode, false to render normally.
---------------------	--

20.6.3.7 float TRITONAPI Triton::Ocean::GetChoppiness () const

Retrieves the choppiness setting of the [Ocean](#), which controls how peaked the waves are.

See also

[SetChoppiness\(\)](#)

20.6.3.8 float TRITONAPI Triton::Ocean::GetDepth (Triton::Vector3 & floorNormal) const

Retrieves the simulated depth of the water in world units, and the surface normal of the sea floor, both at the current camera position.

Only returns the values set by [SetDepth\(\)](#).

See also

[SetDepth\(\)](#)

Parameters

<i>floorNormal</i>	A reference to a Vector3 to retrieve the surface normal of the sea floor as set by SetDepth() .
--------------------	---

Returns

The depth of the sea floor at the camera position, as set by [SetDepth\(\)](#).

20.6.3.9 const char* TRITONAPI Triton::Ocean::GetFFTName () const

Returns a description of the FFT transform being used, if a FFT water model is active.

20.6.3.10 bool TRITONAPI Triton::Ocean::GetHeight (const Vector3 & point, const Vector3 & direction, float & height, Vector3 & normal, bool visualCorrelation = true)

Retrieves the height and normal of the ocean surface at the intersection point of the given ray.

The results of this method are only valid if [Ocean::Create\(\)](#) was called with the parameter `enableHeightReads` set to true. The height returned is relative to sea level, as specified by [Triton::Environment::SetSeaLevel\(\)](#). For example, the crest of a one-meter-high wave will always return a height of one meter, irrespective of the environment's sea level height. Depending on the application, you may want to add in the result of [Triton::Environment::GetSeaLevel\(\)](#) to the height returned.

Parameters

<i>point</i>	The origin of the ray to test the height against.
--------------	---

<i>direction</i>	The normalized direction vector of the ray.
<i>height</i>	Receives the height at the ray's intersection with the ocean, if an intersection was found.
<i>normal</i>	Receives a normalized unit vector pointing in the direction of the normal vector of the sea surface at the intersection point.
<i>visualCorrection</i>	Set to true in order to have the height returned match the visuals, which dampen height offsets with distance to avoid sampling artifacts. To return the true wave height at the given location, set to false.

Returns

True if an intersection was found, false if not.

20.6.3.11 unsigned int TRITONAPI Triton::Ocean::GetNumTriangles () const

Returns the number of triangles rendered by the underlying projected grid.

20.6.3.12 const Vector3& Triton::Ocean::GetRefractionColor () const

Returns the color of light refracted into the water.

See also

[SetRefractionColor\(\)](#);

Returns

The RGB value of the refraction color.

20.6.3.13 ShaderHandle TRITONAPI Triton::Ocean::GetShaderObject () const

Retrieves the underlying shader object used to render the water.

If you make modifications to our shaders to add additional effects, you can use this in order to pass your own uniform variables into the shader. Depending on the renderer you're using, you'll need to cast this to a GLhandleARB, ID3DXEffect, or ID3DX11Effect.

Returns

The GLhandleARB, ID3DXEffect, or ID3DX11Effect representing the shader object used to draw the ocean, or 0 if no shader is loaded.

20.6.3.14 float Triton::Ocean::GetWaveHeading () const [inline]

Retrieves the wave direction.

Normally this is the same as the wind direction, but in shallow water it will align with the slope of the sea floor as specified in [SetDepth\(\)](#).

20.6.3.15 void TRITONAPI Triton::Ocean::SetChoppiness (float *chop*)

Set the choppiness of the waves, which controls how peaked the waves are.

See also

[GetChoppiness\(\)](#)

Parameters

<i>chop</i>	The choppiness parameter; 0.0 yields no chop, 3.0 yields strong chop. Values that are too high may result in wave geometry folding over itself, so take care to set reasonable values.
-------------	--

20.6.3.16 void TRITONAPI Triton::Ocean::SetDepth (float *depth*, const Triton::Vector3 & *floorNormal*)

Sets the simulated depth of the water in world units at the camera position, and the slope of the sea floor as specified by its surface normal at the camera position.

This information is used to interpolate the water depth at various positions in the scene, affecting the transparency of the water as well as the height of the waves. Avoid changing this every frame for performance reasons.

See also

[GetDepth\(\)](#)

Parameters

<i>depth</i>	The depth of the water, in world units, at the camera position. Negative values will be clamped to zero. For open ocean, either do not call this method or set depth to a large number (like 1000).
<i>floorNormal</i>	The surface normal of the sea floor at the camera position. The point defined by the depth parameter under the camera position together with this normal will define a plane that approximates the position of the sea floor surrounding the current location.

20.6.3.17 void Triton::Ocean::SetRefractionColor (const Vector3 & *refractionColor*)

Modifies the color used for refracted light rays that go into deep water.

You can use this to modify the color of the water in areas that are not purely reflective.

Parameters

<i>refraction-Color</i>	the RGB color value of the deep water color; each component should be in the range 0-1.
-------------------------	---

20.6.3.18 `bool Triton::Ocean::SprayEnabled () const [inline]`

Returns if spray particle effects on breaking waves are enabled, which they are by default.

See also

[EnableSpray\(\)](#).

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

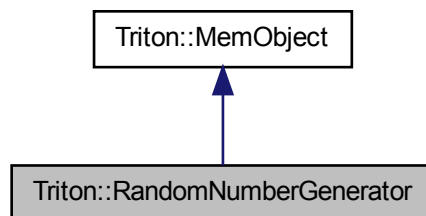
- [C:/triton/trunk/Public Headers/Ocean.h](#)

20.7 Triton::RandomNumberGenerator Class Reference

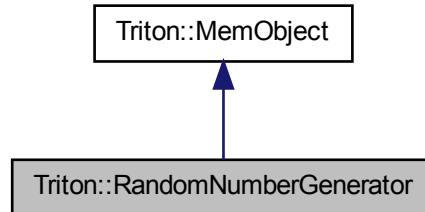
An interface for generating random numbers in Triton.

```
#include <RandomNumberGenerator.h>
```

Inheritance diagram for Triton::RandomNumberGenerator:



Collaboration diagram for Triton::RandomNumberGenerator:



Public Member Functions

- virtual double TRITONAPI [GetRandomDouble](#) (double start, double end) const =0
Return an evenly distributed random double-precision number within a given range.
- virtual int TRITONAPI [GetRandomInt](#) (int start, int end) const =0
Return an evenly distributed random integer within a given range.

20.7.1 Detailed Description

An interface for generating random numbers in Triton. Subclass this interface and pass an instance to [Environment::SetRandomNumberGenerator](#) in order to override Triton's default usage of rand(). This may be useful for enforcing deterministic behavior across several channels.

20.7.2 Member Function Documentation

20.7.2.1 virtual double TRITONAPI Triton::RandomNumberGenerator::GetRandomDouble (double start, double end) const [pure virtual]

Return an evenly distributed random double-precision number within a given range.

Parameters

<i>start</i>	The lowest value in the range
<i>end</i>	The highest value in the range

Returns

An evenly distributed random number within the range.

20.7.2.2 `virtual int TRITONAPI Triton::RandomNumberGenerator::GetRandomInt (int start, int end) const` [pure virtual]

Return an evenly distributed random integer within a given range.

Parameters

<i>start</i>	The lowest value in the range
<i>end</i>	The highest value in the range

Returns

An evenly distributed random number within the range.

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

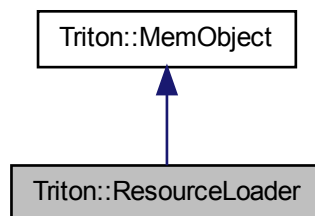
- [C:/triton/trunk/Public Headers/RandomNumberGenerator.h](#)

20.8 Triton::ResourceLoader Class Reference

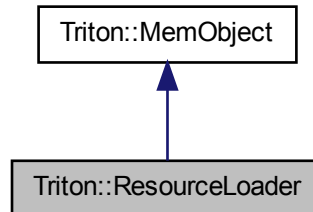
This class is used whenever Triton needs to load textures, data files, or shaders from mass storage; you may extend this class to override our default use of POSIX filesystem calls with your own resource management if you wish.

```
#include <ResourceLoader.h>
```

Inheritance diagram for Triton::ResourceLoader:



Collaboration diagram for Triton::ResourceLoader:



Public Member Functions

- [ResourceLoader](#) (const char *resourceDirPath)
Constructor; requires a path to the Triton resources folder.
- virtual [~ResourceLoader](#) ()
Virtual destructor; frees the memory of the resource path string.
- virtual void TRITONAPI [SetResourceDirPath](#) (const char *path)
Sets the path to the Triton resources folder, which will be pre-pended to all resource filenames passed into [LoadResource\(\)](#).
- const char *TRITONAPI [GetResourceDirPath](#) () const
Retrieves the path set by [SetResourceDirPath\(\)](#).
- virtual bool TRITONAPI [LoadResource](#) (const char *pathName, char *&data, unsigned int &dataLen, bool text)
Load a resource from mass storage; the default implementation uses the POSIX functions [fopen\(\)](#), [fread\(\)](#), and [fclose\(\)](#) to do this, but you may override this method to load resources however you wish.
- virtual void TRITONAPI [FreeResource](#) (char *data)
Frees the resource data memory that was returned from [LoadResource\(\)](#).

20.8.1 Detailed Description

This class is used whenever Triton needs to load textures, data files, or shaders from mass storage; you may extend this class to override our default use of POSIX filesystem calls with your own resource management if you wish. If you have your own system

of packed files, you can include Triton's resources directory into it and implement your own [ResourceLoader](#) to access our resources within your pack files.

20.8.2 Member Function Documentation

20.8.2.1 `virtual void TRITONAPI Triton::ResourceLoader::FreeResource (char * data)`
[virtual]

Frees the resource data memory that was returned from [LoadResource\(\)](#).

The data pointer will be invalid following this call.

20.8.2.2 `virtual bool TRITONAPI Triton::ResourceLoader::LoadResource (const char * pathName, char *& data, unsigned int & dataLen, bool text)` [virtual]

Load a resource from mass storage; the default implementation uses the POSIX functions `fopen()`, `fread()`, and `fclose()` to do this, but you may override this method to load resources however you wish.

The caller is responsible for calling [FreeResource\(\)](#) when it's done consuming the resource data in order to free its memory.

Parameters

<i>pathName</i>	The path to the desired resource, relative to the location of the resources folder previously specified in SetResourceDirPath() .
<i>data</i>	A reference to a <code>char *</code> that will return the resource's data upon a successful load.
<i>dataLen</i>	A reference to an unsigned int that will return the number of bytes loaded upon a successful load.
<i>text</i>	True if the resource is a text file, such as a shader. If true, a terminating null character will be appended to the resulting data and the file will be opened in text mode.

Returns

True if the resource was located and loaded successfully, false otherwise.

See also

[SetResourceDirPath](#)

20.8.2.3 `virtual void TRITONAPI Triton::ResourceLoader::SetResourceDirPath (const char * path)` [virtual]

Sets the path to the Triton resources folder, which will be pre-pended to all resource filenames passed into [LoadResource\(\)](#).

This method also calls the Win32 function `SetDllDirectory` in order to add the dll sub-directory to the application's DLL search path.

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

- C:/triton/trunk/Public Headers/[ResourceLoader.h](#)

20.9 Triton::Utils Class Reference

A collection of static utility methods used by Triton.

```
#include <TritonCommon.h>
```

Static Public Member Functions

- static void TRITONAPI [DebugMsg](#) (const char *debugMessage)
Prints out an error message to the debugger's output window only if the setting enable-debug-messages is set to 'yes' in resources/Triton.config.
- static void TRITONAPI [ClearGLErrors](#) ()
Clears any existing OpenGL error codes so we may test for new ones.
- static bool TRITONAPI [PrintGLErrors](#) (const char *file, int line)
Prints out (and clears) any existing GL errors using [Utils::DebugMsg](#).
- static TRITON_STRING TRITONAPI [GetDLLPath](#) ()
Returns the resources subdirectory where our FFT implementation DLL's live for this specific build's compiler version and platform.
- static TRITON_STRING TRITONAPI [GetDLLSuffix](#) ()
Retrieves the suffix on the FFT implementation DLL's for this specific build flavor's runtime library.
- static TRITON_STRING TRITONAPI [GetDLLExtension](#) ()
Retrieves the DLL extension.
- static struct _D3DXMACRO *TRITONAPI [GetDX9Macros](#) (IDirect3DDevice9 *device)
Returns HLSL preprocessor defines for DirectX9, indicating the shader models available to our shaders.

20.9.1 Detailed Description

A collection of static utility methods used by Triton.

20.9.2 Member Function Documentation

20.9.2.1 `static void TRITONAPI Triton::Utils::ClearGLErrors () [static]`

Clears any existing OpenGL error codes so we may test for new ones.

20.9.2.2 `static TRITON_STRING TRITONAPI Triton::Utils::GetDLLExtension () [inline, static]`

Retrieves the DLL extension.

20.9.2.3 `static TRITON_STRING TRITONAPI Triton::Utils::GetDLLPath () [inline, static]`

Returns the resources subdirectory where our FFT implementation DLL's live for this specific build's compiler version and platform.

20.9.2.4 `static TRITON_STRING TRITONAPI Triton::Utils::GetDLLSuffix () [static]`

Retrieves the suffix on the FFT implementation DLL's for this specific build flavor's runtime library.

20.9.2.5 `static struct _D3DXMACRO* TRITONAPI Triton::Utils::GetDX9Macros (IDirect3DDevice9 * device) [static, read]`

Returns HLSL preprocessor defines for DirectX9, indicating the shader models available to our shaders.

Returns

NULL if minimum system requirements for DX9 are not present, a null-terminated D3DXMACRO array otherwise.

20.9.2.6 `static bool TRITONAPI Triton::Utils::PrintGLErrors (const char * file, int line) [static]`

Prints out (and clears) any existing GL errors using [Utils::DebugMsg](#).

Returns

True if no error code was present.

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

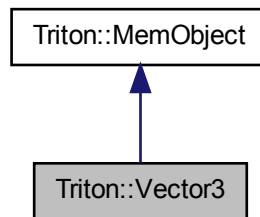
- [C:/triton/trunk/Public Headers/TritonCommon.h](#)

20.10 Triton::Vector3 Class Reference

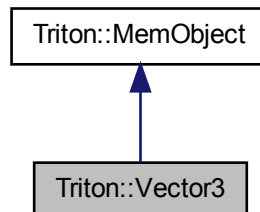
A 3D double-precision Vector class and its operations.

```
#include <Vector3.h>
```

Inheritance diagram for Triton::Vector3:



Collaboration diagram for Triton::Vector3:



Public Member Functions

- [Vector3 \(\)](#)
Default constructor, initializes to (0,0,0).
- [Vector3 \(double px, double py, double pz\)](#)
Constructs a [Vector3](#) with the given x,y,z coordinates.

- **Vector3** (const double *p)
Constructs a Vector 3 from a pointer to 3 doubles.
- **Vector3** (const **Vector3** &v)
Copy constructor.
- double TRITONAPI **Length** () const
Returns the length of the vector.
- double TRITONAPI **SquaredLength** () const
Returns the squared length of the vector, which is faster than computing the length.
- void TRITONAPI **Normalize** ()
Scales the vector to be of length 1.0.
- double TRITONAPI **Dot** (const **Vector3** &v) const
Determines the dot product between this vector and another, and returns the result.
- **Vector3** TRITONAPI **Cross** (const **Vector3** &v) const
Determines the cross product between this vector and another, and returns the result.
- **Vector3** TRITONAPI **operator*** (double n) const
Scales each x,y,z value of the vector by a constant n, and returns the result.
- **Vector3** TRITONAPI **operator*** (const **Vector3** &v) const
Multiplies the components of two vectors together, and returns the result.
- **Vector3** TRITONAPI **operator+** (double n) const
Adds a constant n to each component of the vector, and returns the result.
- **Vector3** TRITONAPI **operator-** (const **Vector3** &v) const
Subtracts the specified vector from this vector, and returns the result.
- **Vector3** TRITONAPI **operator+** (const **Vector3** &v) const
Adds this vector to the specified vector, and returns the result.
- bool TRITONAPI **operator==** (const **Vector3** &v) const
Tests if two vectors are exactly equal.
- bool TRITONAPI **operator!=** (const **Vector3** &v) const
Test if two vectors are not exactly equal.
- void TRITONAPI **Serialize** (std::ostream &s) const
Write this vector's data to a file.
- void TRITONAPI **Unserialize** (std::istream &s)
Restore this vector from a file.

Public Attributes

- double `x`

Data members `x,y,z` public for convenience.

20.10.1 Detailed Description

A 3D double-precision Vector class and its operations.

20.10.2 Constructor & Destructor Documentation

20.10.2.1 `Triton::Vector3::Vector3 ()` [inline]

Default constructor, initializes to (0,0,0).

20.10.2.2 `Triton::Vector3::Vector3 (double px, double py, double pz)` [inline]

Constructs a `Vector3` with the given `x,y,z` coordinates.

20.10.3 Member Function Documentation

20.10.3.1 `Vector3 TRITONAPI Triton::Vector3::Cross (const Vector3 & v) const` [inline]

Determines the cross product between this vector and another, and returns the result.

20.10.3.2 `double TRITONAPI Triton::Vector3::Dot (const Vector3 & v) const` [inline]

Determines the dot product between this vector and another, and returns the result.

20.10.3.3 `double TRITONAPI Triton::Vector3::Length () const` [inline]

Returns the length of the vector.

20.10.3.4 `void TRITONAPI Triton::Vector3::Normalize ()` [inline]

Scales the vector to be of length 1.0.

20.10.3.5 `bool TRITONAPI Triton::Vector3::operator!= (const Vector3 & v) const` [inline]

Test if two vectors are not exactly equal.

20.10.3.6 `Vector3 TRITONAPI Triton::Vector3::operator* (double n) const [inline]`

Scales each x,y,z value of the vector by a constant n, and returns the result.

20.10.3.7 `Vector3 TRITONAPI Triton::Vector3::operator* (const Vector3 & v) const [inline]`

Multiplies the components of two vectors together, and returns the result.

20.10.3.8 `Vector3 TRITONAPI Triton::Vector3::operator+ (double n) const [inline]`

Adds a constant n to each component of the vector, and returns the result.

20.10.3.9 `Vector3 TRITONAPI Triton::Vector3::operator+ (const Vector3 & v) const [inline]`

Adds this vector to the specified vector, and returns the result.

20.10.3.10 `Vector3 TRITONAPI Triton::Vector3::operator- (const Vector3 & v) const [inline]`

Subtracts the specified vector from this vector, and returns the result.

20.10.3.11 `bool TRITONAPI Triton::Vector3::operator== (const Vector3 & v) const [inline]`

Tests if two vectors are exactly equal.

20.10.3.12 `void TRITONAPI Triton::Vector3::Serialize (std::ostream & s) const [inline]`

Write this vector's data to a file.

20.10.3.13 `double TRITONAPI Triton::Vector3::SquaredLength () const [inline]`

Returns the squared length of the vector, which is faster than computing the length.

20.10.3.14 `void TRITONAPI Triton::Vector3::Unserialize (std::istream & s) [inline]`

Restore this vector from a file.

20.10.4 Member Data Documentation

20.10.4.1 double Triton::Vector3::x

Data members x,y,z public for convenience.

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

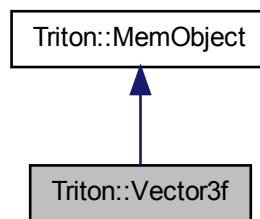
- [C:/triton/trunk/Public Headers/Vector3.h](#)

20.11 Triton::Vector3f Class Reference

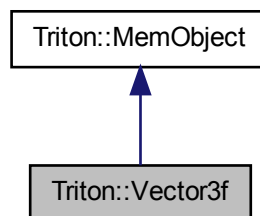
A 3D single-precision vector class, and its operations.

```
#include <Vector3.h>
```

Inheritance diagram for Triton::Vector3f:



Collaboration diagram for Triton::Vector3f:



Public Member Functions

- [Vector3f](#) ()
Default constructor; does not initialize the vector.
- [Vector3f](#) (const [Triton::Vector3](#) &*v*)
Construct a single precision vector from a double precision one.
- [Vector3f](#) (float *px*, float *py*, float *pz*)
*Constructs a [Vector3f](#) from the specified single-precision floating point *x*, *y*, and *z* values.*
- float TRITONAPI [Length](#) ()
Returns the length of this vector.
- void TRITONAPI [Normalize](#) ()
Scales the vector to be of length 1.0.
- double TRITONAPI [Dot](#) (const [Vector3](#) &*v*) const
Returns the dot product of this vector with the specified [Vector3](#).
- [Vector3f](#) TRITONAPI [operator-](#) (const [Vector3f](#) &*v*) const
Subtracts the specified vector from this vector, and returns the result.
- [Vector3f](#) TRITONAPI [operator+](#) (const [Vector3f](#) &*v*) const
Adds this vector to the specified vector, and returns the result.

Public Attributes

- float *x*
*Data members *x*, *y*, *z* are public for convenience.*

20.11.1 Detailed Description

A 3D single-precision vector class, and its operations.

20.11.2 Constructor & Destructor Documentation

20.11.2.1 [Triton::Vector3f::Vector3f](#) () [[inline](#)]

Default constructor; does not initialize the vector.

20.11.2.2 `Triton::Vector3f::Vector3f (const Triton::Vector3 & v)` [inline]

Construct a single precision vector from a double precision one.

20.11.2.3 `Triton::Vector3f::Vector3f (float px, float py, float pz)` [inline]

Constructs a `Vector3f` from the specified single-precision floating point x, y, and z values.

20.11.3 Member Function Documentation**20.11.3.1** `double TRITONAPI Triton::Vector3f::Dot (const Vector3 & v) const` [inline]

Returns the dot product of this vector with the specified `Vector3`.

20.11.3.2 `float TRITONAPI Triton::Vector3f::Length ()` [inline]

Returns the length of this vector.

20.11.3.3 `void TRITONAPI Triton::Vector3f::Normalize ()` [inline]

Scales the vector to be of length 1.0.

20.11.3.4 `Vector3f TRITONAPI Triton::Vector3f::operator+ (const Vector3f & v) const`
[inline]

Adds this vector to the specified vector, and returns the result.

20.11.3.5 `Vector3f TRITONAPI Triton::Vector3f::operator- (const Vector3f & v) const`
[inline]

Subtracts the specified vector from this vector, and returns the result.

20.11.4 Member Data Documentation**20.11.4.1** `float Triton::Vector3f::x`

Data members x, y, z are public for convenience.

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

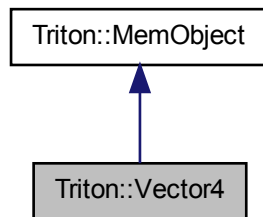
- `C:/triton/trunk/Public Headers/Vector3.h`

20.12 Triton::Vector4 Class Reference

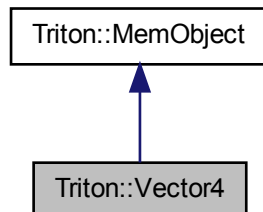
A simple double-precision 4D vector class with no operations defined.

```
#include <Vector4.h>
```

Inheritance diagram for Triton::Vector4:



Collaboration diagram for Triton::Vector4:



Public Member Functions

- [Vector4](#) (double px, double py, double pz, double pw)
Constructs a [Vector4](#) from the given x, y, z, and w values.
- [Vector4](#) (const [Vector3](#) &v3)
Constructs a [Vector4](#) from a [Vector3](#), setting w to 1.

- [Vector4](#) ()
Default constructor; initializes the [Vector4](#) to (0, 0, 0, 1)
- double TRITONAPI [Dot](#) (const [Vector4](#) &v) const
Determines the dot product between this vector and another, and returns the result.
- [Vector4](#) TRITONAPI [operator*](#) (double n) const
Scales each x,y,z value of the vector by a constant n, and returns the result.
- [Vector4](#) TRITONAPI [operator*](#) (const [Vector4](#) &v) const
Multiplies the components of two vectors together, and returns the result.
- [Vector4](#) TRITONAPI [operator+](#) (double n) const
Adds a constant n to each component of the vector, and returns the result.
- [Vector4](#) TRITONAPI [operator-](#) (const [Vector4](#) &v) const
Subtracts the specified vector from this vector, and returns the result.
- [Vector4](#) TRITONAPI [operator+](#) (const [Vector4](#) &v) const
Adds this vector to the specified vector, and returns the result.

Public Attributes

- double [x](#)
The x, y, z, and w data members are public for convenience.

20.12.1 Detailed Description

A simple double-precision 4D vector class with no operations defined. Essentially a struct with constructors.

20.12.2 Constructor & Destructor Documentation

20.12.2.1 [Triton::Vector4::Vector4](#) (double *px*, double *py*, double *pz*, double *pw*)
[inline]

Constructs a [Vector4](#) from the given x, y, z, and w values.

20.12.2.2 [Triton::Vector4::Vector4](#) (const [Vector3](#) & *v3*) [inline]

Constructs a [Vector4](#) from a [Vector3](#), setting w to 1.

20.12.3 Member Function Documentation

20.12.3.1 `double TRITONAPI Triton::Vector4::Dot (const Vector4 & v) const` `[inline]`

Determines the dot product between this vector and another, and returns the result.

20.12.3.2 `Vector4 TRITONAPI Triton::Vector4::operator* (double n) const` `[inline]`

Scales each x,y,z value of the vector by a constant n, and returns the result.

20.12.3.3 `Vector4 TRITONAPI Triton::Vector4::operator* (const Vector4 & v) const`
`[inline]`

Multiplies the components of two vectors together, and returns the result.

20.12.3.4 `Vector4 TRITONAPI Triton::Vector4::operator+ (double n) const` `[inline]`

Adds a constant n to each component of the vector, and returns the result.

20.12.3.5 `Vector4 TRITONAPI Triton::Vector4::operator+ (const Vector4 & v) const`
`[inline]`

Adds this vector to the specified vector, and returns the result.

20.12.3.6 `Vector4 TRITONAPI Triton::Vector4::operator- (const Vector4 & v) const`
`[inline]`

Subtracts the specified vector from this vector, and returns the result.

20.12.4 Member Data Documentation

20.12.4.1 `double Triton::Vector4::x`

The x, y, z, and w data members are public for convenience.

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

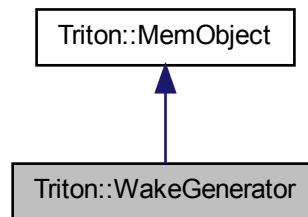
- [C:/triton/trunk/Public Headers/Vector4.h](#)

20.13 Triton::WakeGenerator Class Reference

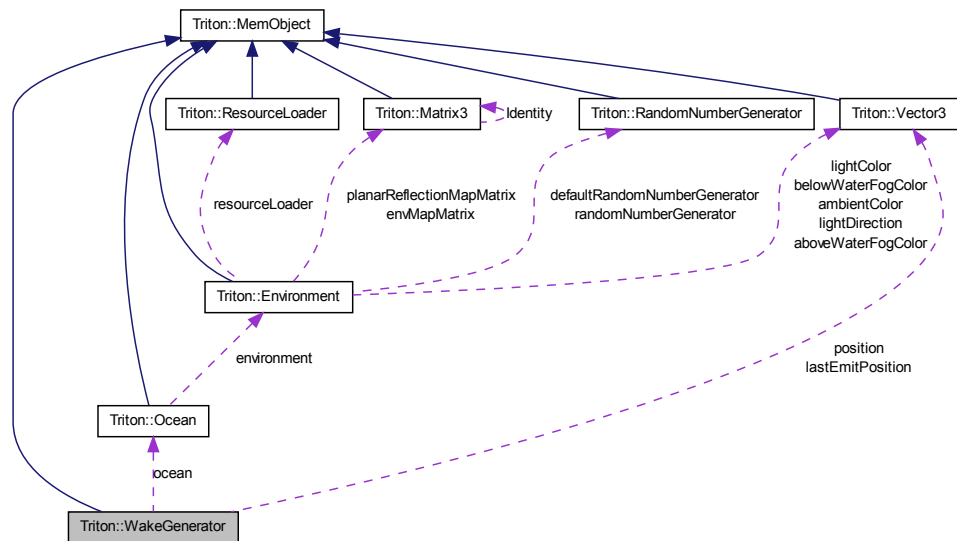
A [WakeGenerator](#) represents an object on the water that generates a wake as it moves, such as a ship.

```
#include <WakeGenerator.h>
```

Inheritance diagram for Triton::WakeGenerator:



Collaboration diagram for Triton::WakeGenerator:



Public Member Functions

- [WakeGenerator](#) ([Ocean](#) *pOcean, bool pSprayEffects=false, double pSprayOffset=0.0, double pLength=100.0, double pBeamWidth=20.0)

Construct a [WakeGenerator](#) with the same [Triton::Ocean](#) it will be associated with.

- void TRITONAPI [Update](#) (const [Vector3](#) &pPosition, const [Vector3](#) &pDirection, double pVelocity, double pTime)
For any active [WakeGenerator](#), this should be called every frame to update its position and velocity.
- [Vector3](#) TRITONAPI [GetPosition](#) () const
Retrieves the world position of the [WakeGenerator](#).
- double TRITONAPI [GetVelocity](#) () const
Retrieves the velocity of the [WakeGenerator](#).

20.13.1 Detailed Description

A [WakeGenerator](#) represents an object on the water that generates a wake as it moves, such as a ship. Simply call [Triton::WakeGenerator::Update\(\)](#) to move the object and generate a realistic wake behind it. Any [WakeGenerator](#) moving at constant velocity will generate a wake of 19.46 degrees behind it, but acceleration, deceleration, and curved paths are all handled properly as well.

20.13.2 Constructor & Destructor Documentation

20.13.2.1 [Triton::WakeGenerator::WakeGenerator](#) ([Ocean](#) * pOcean, bool pSprayEffects = false, double pSprayOffset = 0.0, double pLength = 100.0, double pBeamWidth = 20.0) [inline]

Construct a [WakeGenerator](#) with the same [Triton::Ocean](#) it will be associated with.

Parameters

<i>pOcean</i>	The Triton::Ocean object you will associate this WakeGenerator with.
<i>pSprayEffects</i>	Whether you wish this wake to emit spray particles originating from this wake generator.
<i>pSprayOffset</i>	Use this to have spray particles emitted from a point different from where the wakes are generated from. This distance will be added to the current wake generator's position along the direction of travel. For example, if you want wakes to originate from the stern of a ship but have spray emitted from the bow, you would pass in the length of the ship for pSprayOffset, and update the WakeGenerator with the position of the stern each frame. Unused if pSprayEffects is false.
<i>pLength</i>	The length of the object generating the wake.
<i>pBeamWidth</i>	The width of the object generating the wake.

20.13.3 Member Function Documentation

20.13.3.1 `Vector3 TRITONAPI Triton::WakeGenerator::GetPosition () const [inline]`

Retrieves the world position of the [WakeGenerator](#).

Returns

The world position of the [WakeGenerator](#), as last specified by [Triton::WakeGenerator::Update\(\)](#).

20.13.3.2 `double TRITONAPI Triton::WakeGenerator::GetVelocity () const [inline]`

Retrieves the velocity of the [WakeGenerator](#).

Returns

The velocity of the [WakeGenerator](#) in world units per second, as last specified by [Triton::WakeGenerator::Update\(\)](#).

20.13.3.3 `void TRITONAPI Triton::WakeGenerator::Update (const Vector3 & pPosition, const Vector3 & pDirection, double pVelocity, double pTime)`

For any active [WakeGenerator](#), this should be called every frame to update its position and velocity.

No wake will be generated until this is called.

Parameters

<i>pPosition</i>	The position of the object generating the wake, such as the stern of a ship, in world coordinates.
<i>pDirection</i>	A normalized direction vector indicating the direction this object is moving in.
<i>pVelocity</i>	The velocity of the object generating the wake, in world units per second.
<i>pTime</i>	The current simulated time sample, in seconds. This may be relative to any reference point in time, as long as that reference point is consistent among the multiple calls to Update() .

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

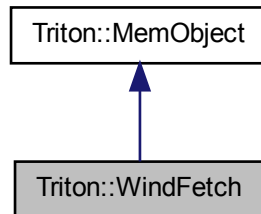
- [C:/triton/trunk/Public Headers/WakeGenerator.h](#)

20.14 Triton::WindFetch Class Reference

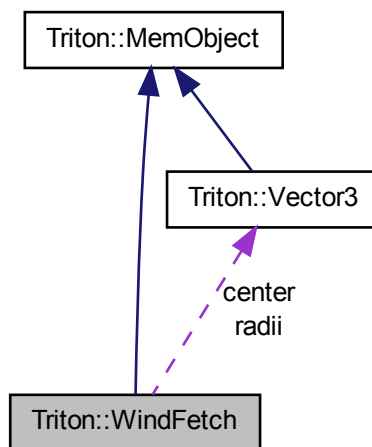
A localized or global area of wind of given speed and direction.

```
#include <WindFetch.h>
```

Inheritance diagram for Triton::WindFetch:



Collaboration diagram for Triton::WindFetch:



Public Member Functions

- [WindFetch \(\)](#)
Default constructor.
- void TRITONAPI [SetWind](#) (double speed, double direction)

Sets the wind speed and direction of this wind fetch.

- void TRITONAPI [SetLocalization](#) (const [Vector3](#) ¢er, const [Vector3](#) &radii)

Sets a localized area, in the form of an ellipsoid, in which the wind fetch is active.

- void TRITONAPI [ClearLocalization](#) ()

Clears any localization and makes this wind fetch globally applied.

- void TRITONAPI [GetWindAtLocation](#) (const [Vector3](#) &position, double &windSpeed, double &windDirection) const

Retrieves the wind direction and speed from this wind fetch at the given location.

20.14.1 Detailed Description

A localized or global area of wind of given speed and direction.

20.14.2 Constructor & Destructor Documentation

20.14.2.1 Triton::WindFetch::WindFetch ()

Default constructor.

20.14.3 Member Function Documentation

20.14.3.1 void TRITONAPI Triton::WindFetch::ClearLocalization ()

Clears any localization and makes this wind fetch globally applied.

See also

[SetLocalization\(\)](#)

20.14.3.2 void TRITONAPI Triton::WindFetch::GetWindAtLocation (const [Vector3](#) & position, double & windSpeed, double & windDirection) const

Retrieves the wind direction and speed from this wind fetch at the given location.

If the location specified is not included by the bounds of this wind fetch, or the wind fetch is not global, no wind will be returned.

Parameters

<i>position</i>	The location at which you want to retrieve wind information from this fetch.
-----------------	--

<i>windSpeed</i>	The wind speed, in units per second, resulting from this fetch at the given position.
<i>windDirection</i>	The direction of the wind, in radians, resulting from this fetch at the given position.

20.14.3.3 void TRITONAPI Triton::WindFetch::SetLocalization (const Vector3 & center, const Vector3 & radii)

Sets a localized area, in the form of an ellipsoid, in which the wind fetch is active.

If this method is not called, the wind fetch is assumed to be global.

See also

[ClearLocalization\(\)](#)

Parameters

<i>center</i>	The center of the ellipsoid that models the bounds of the wind fetch.
<i>radii</i>	The radii in X, Y, and Z of the ellipsoid, in world units.

20.14.3.4 void TRITONAPI Triton::WindFetch::SetWind (double speed, double direction)

Sets the wind speed and direction of this wind fetch.

Parameters

<i>speed</i>	The wind speed in world units per second.
<i>direction</i>	The wind direction, in radians.

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

- [C:/triton/trunk/Public Headers/WindFetch.h](#)

Chapter 21

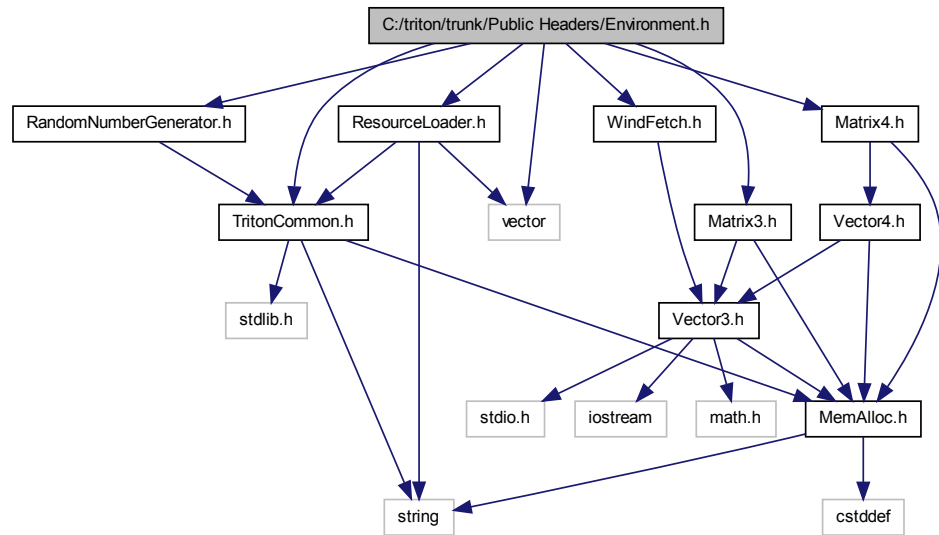
File Documentation

21.1 C:/triton/trunk/Public Headers/Environment.h File Reference

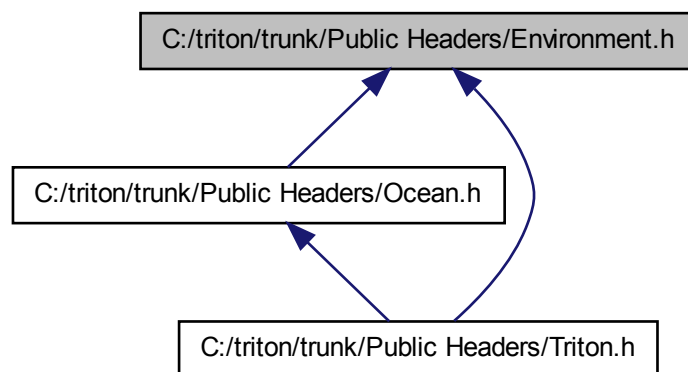
The public interface for setting Triton's environmental parameters.

```
#include "TritonCommon.h"  
#include "ResourceLoader.h"  
#include "RandomNumberGenerator.h"  
#include "WindFetch.h"  
#include "Matrix3.h"  
#include "Matrix4.h"  
#include <vector>
```

Include dependency graph for Environment.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Environment](#)

Triton's public interface for specifying the environmental conditions and camera properties.

Enumerations

- enum **CoordinateSystem** { ,

[Triton::WGS84_YUP](#), [Triton::SPHERICAL_ZUP](#), [Triton::SPHERICAL_YUP](#),
[Triton::FLAT_ZUP](#),

[Triton::FLAT_YUP](#) }

Supported coordinate systems for the Environment constructor.

- enum **Renderer** { ,

[Triton::OPENGL_3_2](#), [Triton::OPENGL_4_0](#), [Triton::OPENGL_4_1](#), [Triton::DIRECTX_9](#),

[Triton::DIRECT3D9_EX](#), [Triton::DIRECTX_11](#) }

Support renderers for the Environment constructor.

- enum **EnvironmentError** { , [Triton::NO_CONFIG_FOUND](#), [Triton::NULL_RESOURCE_LOADER](#), [Triton::NO_DEVICE](#) }

Error codes returned from Environment::Initialize().

21.1.1 Detailed Description

The public interface for setting Triton's environmental parameters.

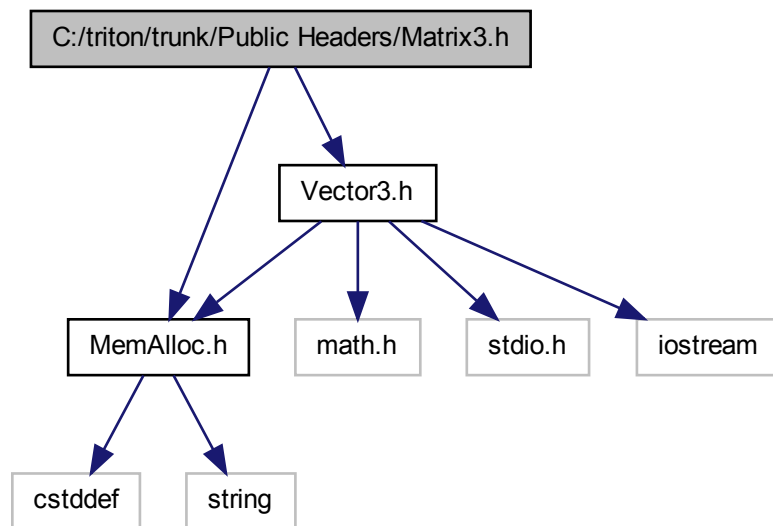
21.2 C:/triton/trunk/Public Headers/Matrix3.h File Reference

Implements a 3x3 matrix and its operations.

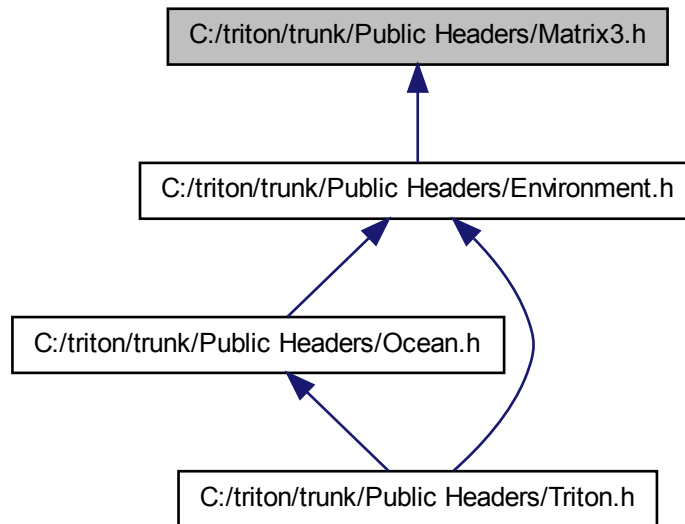
```
#include "MemAlloc.h"
```

```
#include "Vector3.h"
```

Include dependency graph for Matrix3.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Matrix3](#)

A simple 3x3 matrix class and its operations.

21.2.1 Detailed Description

Implements a 3x3 matrix and its operations.

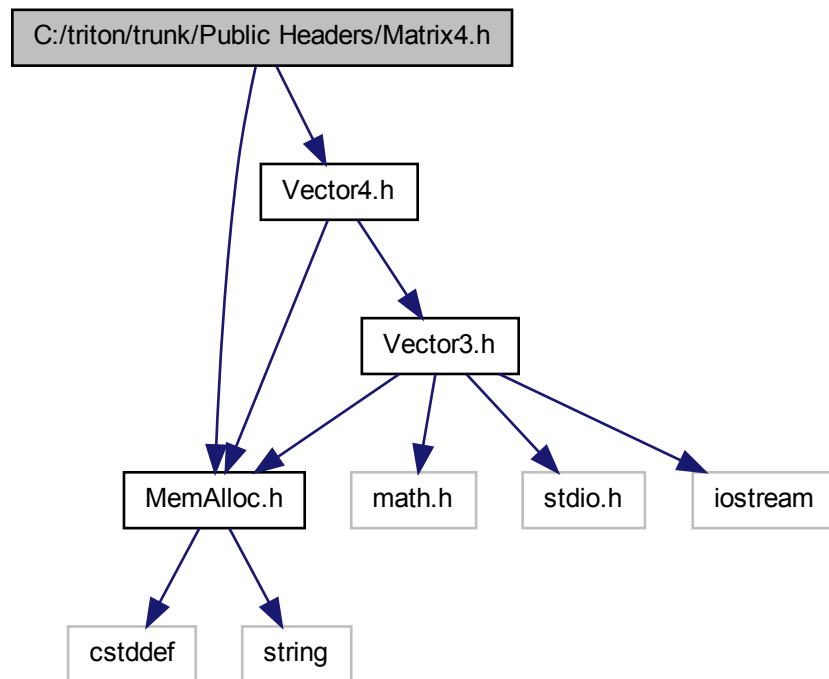
21.3 C:/triton/trunk/Public Headers/Matrix4.h File Reference

An implementation of a 4x4 matrix and some simple operations on it.

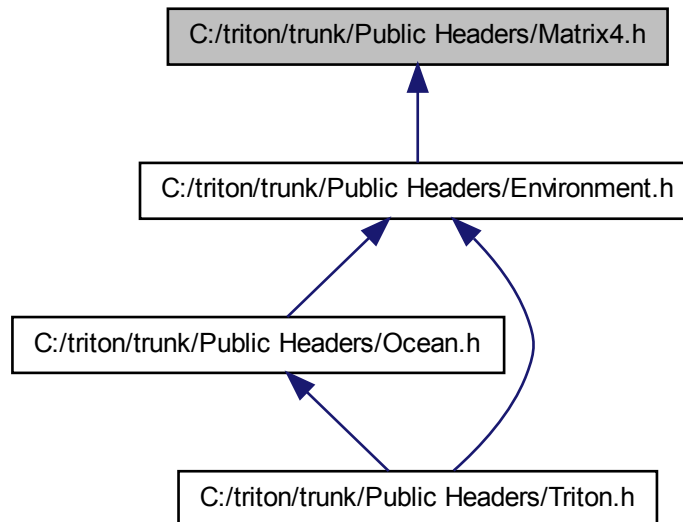
```
#include "MemAlloc.h"
```

```
#include "Vector4.h"
```

Include dependency graph for Matrix4.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Matrix4](#)

An implementation of a 4x4 matrix and some simple operations on it.

21.3.1 Detailed Description

An implementation of a 4x4 matrix and some simple operations on it.

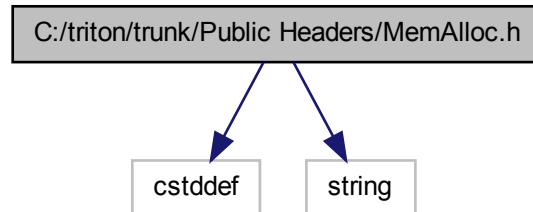
21.4 C:/triton/trunk/Public Headers/MemAlloc.h File Reference

Memory allocation interface for SilverLining.

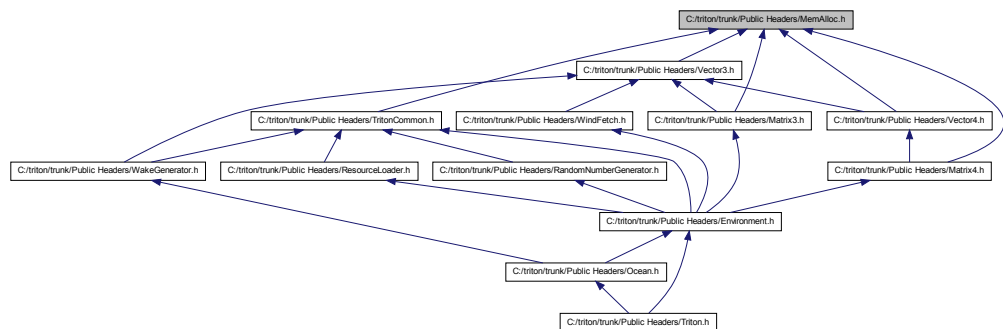
```
#include <cstddef>
```

```
#include <string>
```

Include dependency graph for MemAlloc.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Allocator](#)

You may extend the [Allocator](#) class to hook your own memory management scheme into Triton.

- class [Triton::MemObject](#)

This base class for all Triton objects intercepts the new and delete operators, routing them through [Triton::Allocator\(\)](#).

21.4.1 Detailed Description

Memory allocation interface for SilverLining.

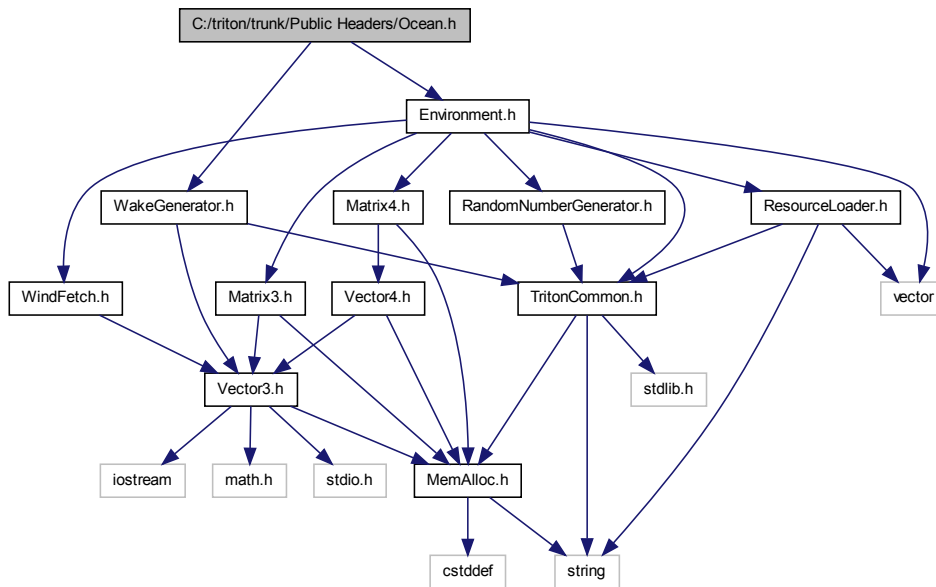
21.5 C:/triton/trunk/Public Headers/Ocean.h File Reference

Triton's Ocean model interface.

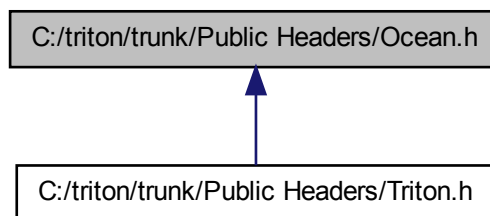
```
#include "Environment.h"
```

```
#include "WakeGenerator.h"
```

Include dependency graph for Ocean.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Ocean](#)

The [Ocean](#) class allows you to configure and draw Triton's water simulation.

Enumerations

- enum **WaterModelTypes** { , [Triton::TESSENDORF](#) }

Enumerates the different water models available for simulating ocean waves.

21.5.1 Detailed Description

Triton's Ocean model interface.

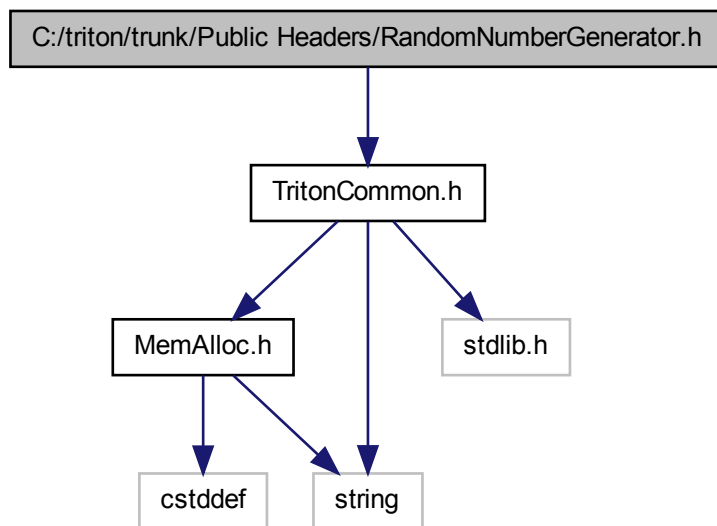
21.6 C:/triton/trunk/Public Headers/RandomNumberGenerator.h File Reference

An interface for overriding Triton's generation of random numbers.

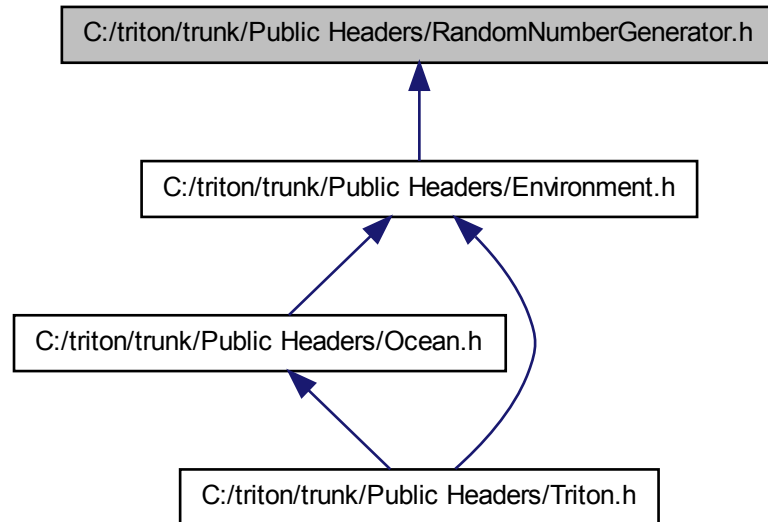
```
#include "TritonCommon.h"
```

21.6 C:/triton/trunk/Public Headers/RandomNumberGenerator.h File Reference

Include dependency graph for RandomNumberGenerator.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::RandomNumberGenerator](#)

An interface for generating random numbers in Triton.

21.6.1 Detailed Description

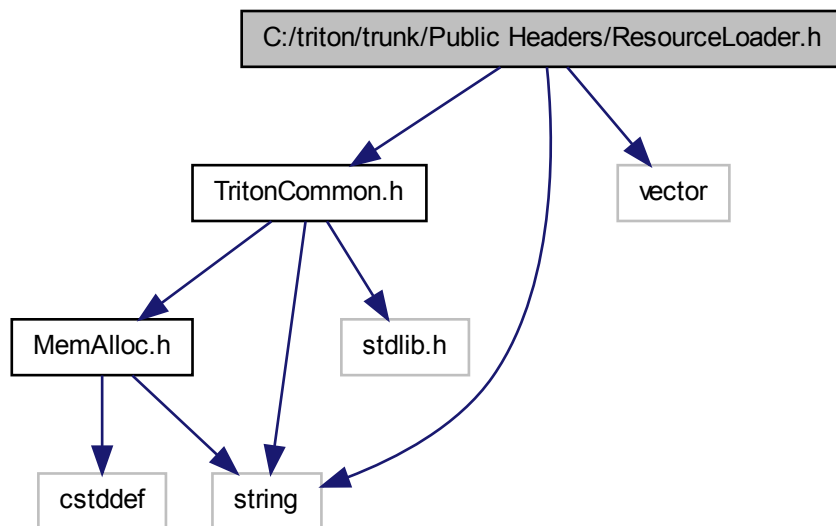
An interface for overriding Triton's generation of random numbers.

21.7 C:/triton/trunk/Public Headers/ResourceLoader.h File Reference

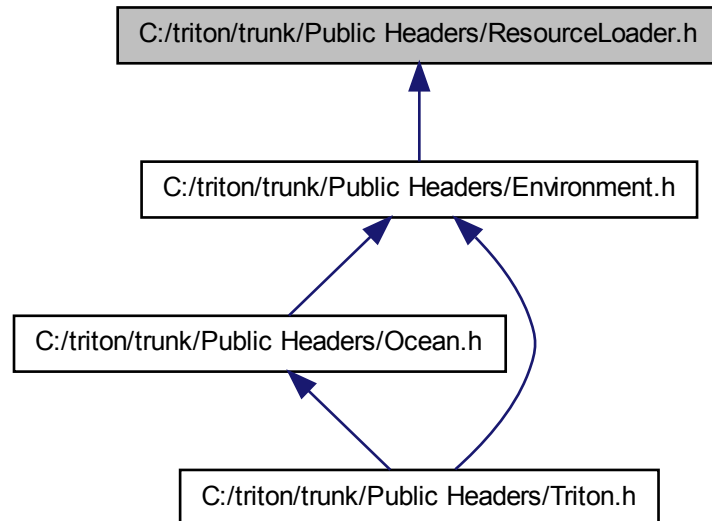
A class for loading Triton's resources from mass storage, which you may extend.

```
#include "TritonCommon.h"  
#include <vector>  
#include <string>
```

Include dependency graph for ResourceLoader.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::ResourceLoader](#)

This class is used whenever Triton needs to load textures, data files, or shaders from mass storage; you may extend this class to override our default use of POSIX filesystem calls with your own resource management if you wish.

21.7.1 Detailed Description

A class for loading Triton's resources from mass storage, which you may extend.

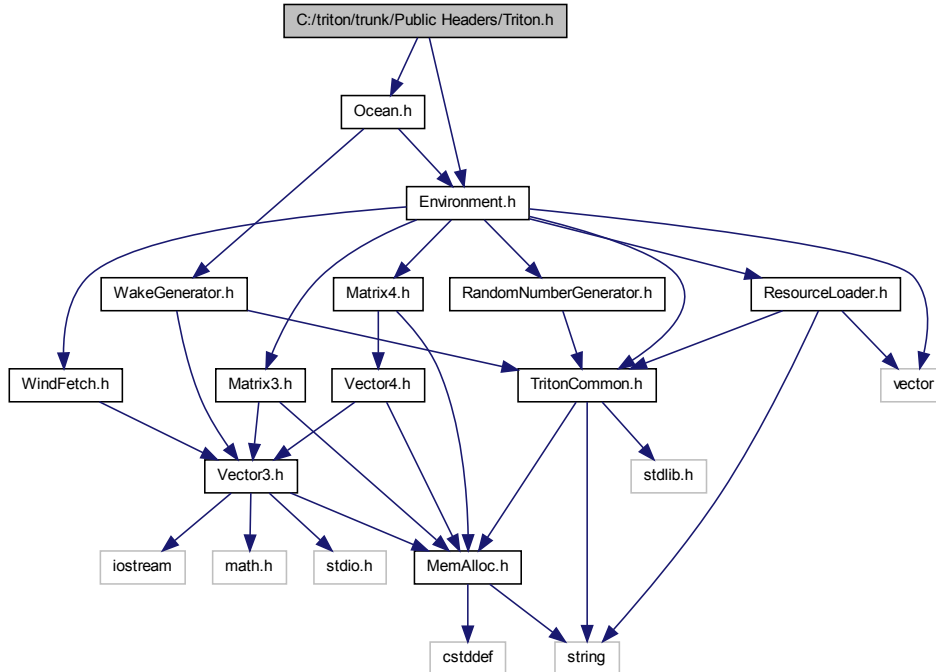
21.8 C:/triton/trunk/Public Headers/Triton.h File Reference

A convenience header that includes the main public headers for Triton.

```
#include "Environment.h"
```

```
#include "Ocean.h"
```

Include dependency graph for Triton.h:



21.8.1 Detailed Description

A convenience header that includes the main public headers for Triton.

21.9 C:/triton/trunk/Public Headers/TritonCommon.h File Reference

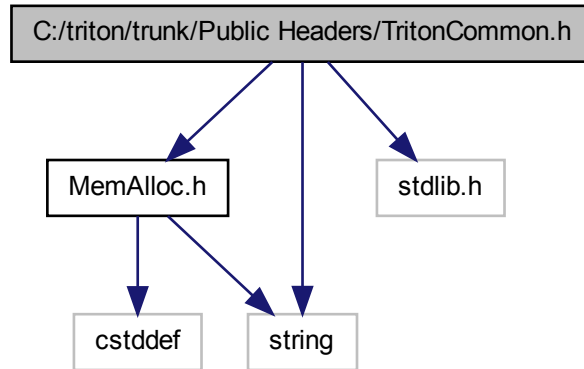
Common typedefs and defines used within Triton.

```

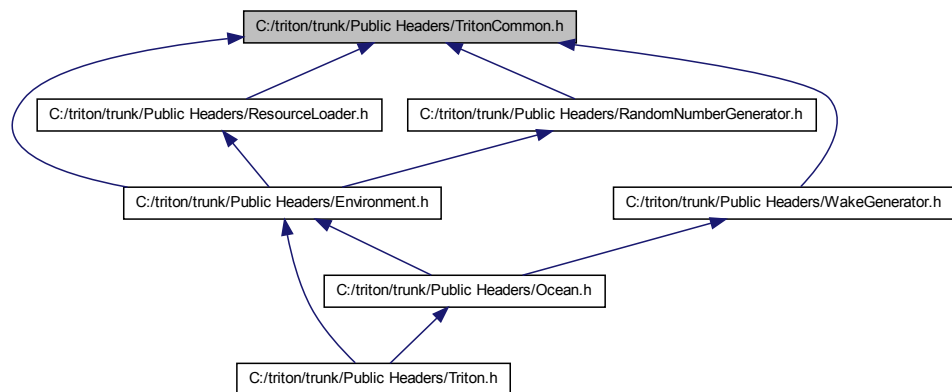
#include "MemAlloc.h"
#include <stdlib.h>
#include <string>

```

Include dependency graph for TritonCommon.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Utils](#)

A collection of static utility methods used by Triton.

Typedefs

- typedef int [Triton::ShaderHandle](#)
A renderer-agnostic handle for a shader.
- typedef int [Triton::TextureHandle](#)
A renderer-agnostic handle for a texture.

21.9.1 Detailed Description

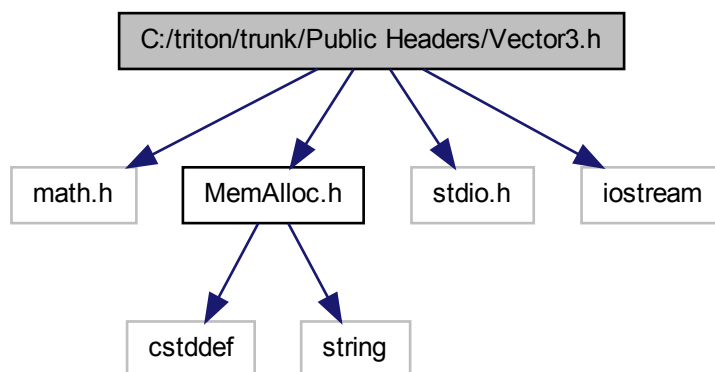
Common typedefs and defines used within Triton.

21.10 C:/triton/trunk/Public Headers/Vector3.h File Reference

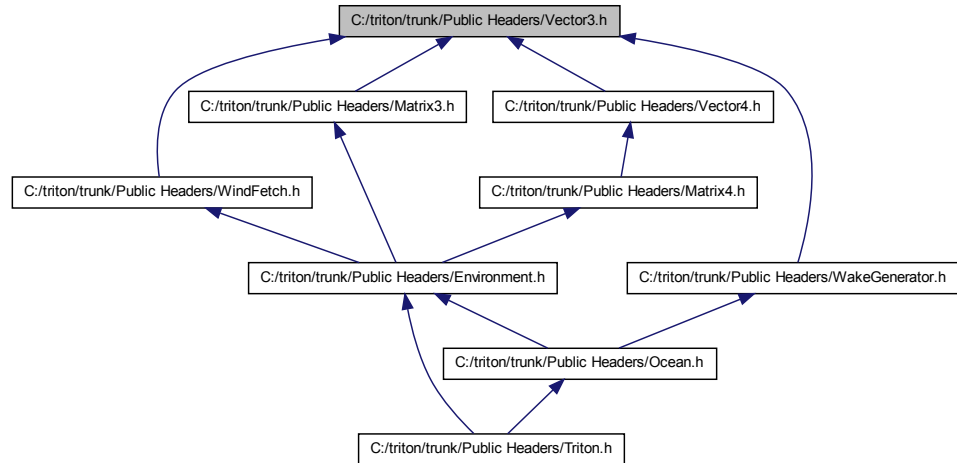
A 3D Vector class and its operations.

```
#include <math.h>
#include "MemAlloc.h"
#include <stdio.h>
#include <iostream>
```

Include dependency graph for Vector3.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Vector3](#)
A 3D double-precision Vector class and its operations.
- class [Triton::Vector3f](#)
A 3D single-precision vector class, and its operations.

21.10.1 Detailed Description

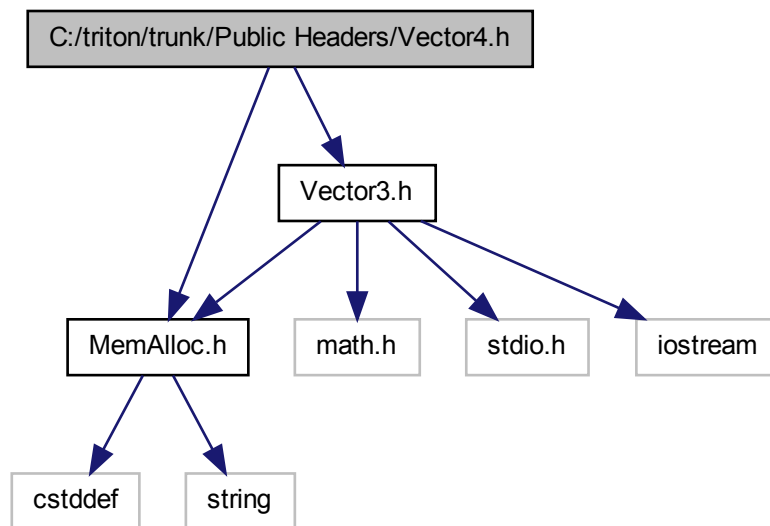
A 3D Vector class and its operations.

21.11 C:/triton/trunk/Public Headers/Vector4.h File Reference

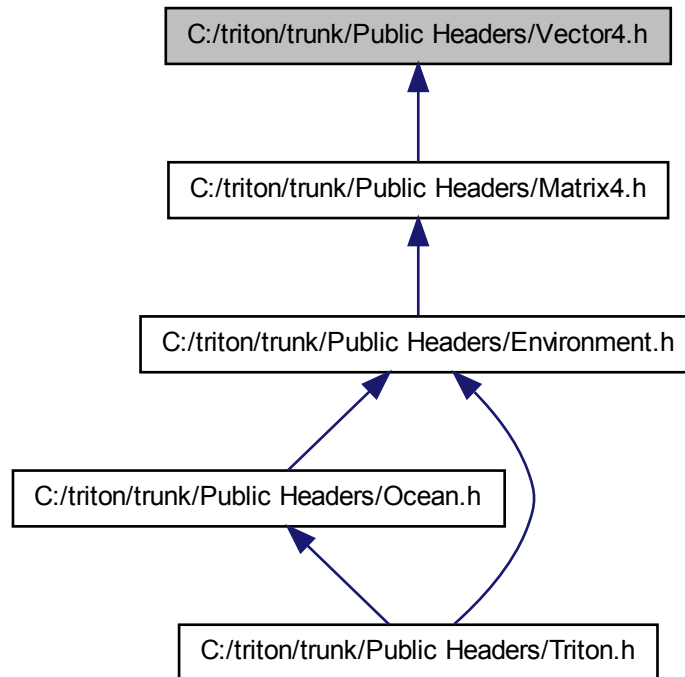
A simple 4D vector class.

```
#include "MemAlloc.h"
#include "Vector3.h"
```

Include dependency graph for Vector4.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Vector4](#)

A simple double-precision 4D vector class with no operations defined.

21.11.1 Detailed Description

A simple 4D vector class.

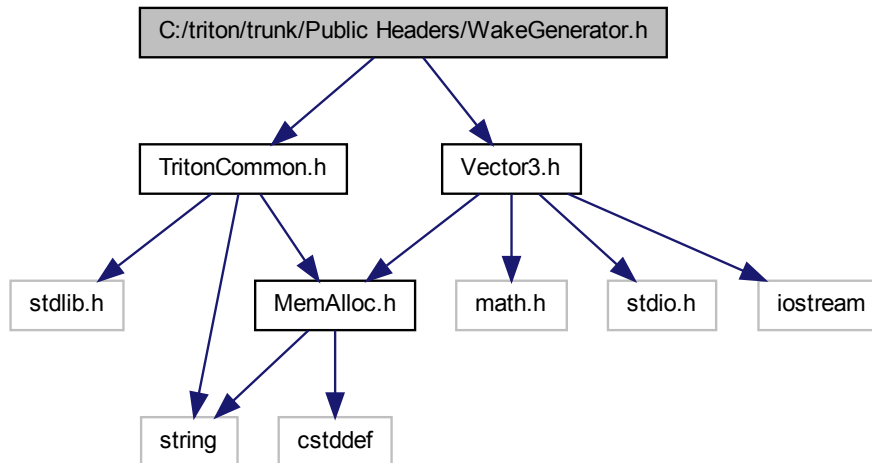
21.12 C:/triton/trunk/Public Headers/WakeGenerator.h File Reference

An object that generates a ship Kelvin wake as it moves.

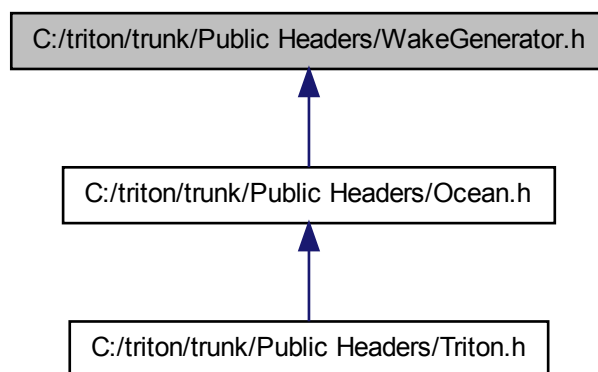
```
#include "TritonCommon.h"
```

```
#include "Vector3.h"
```

Include dependency graph for WakeGenerator.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::WakeGenerator](#)

A *WakeGenerator* represents an object on the water that generates a wake as it moves, such as a ship.

21.12.1 Detailed Description

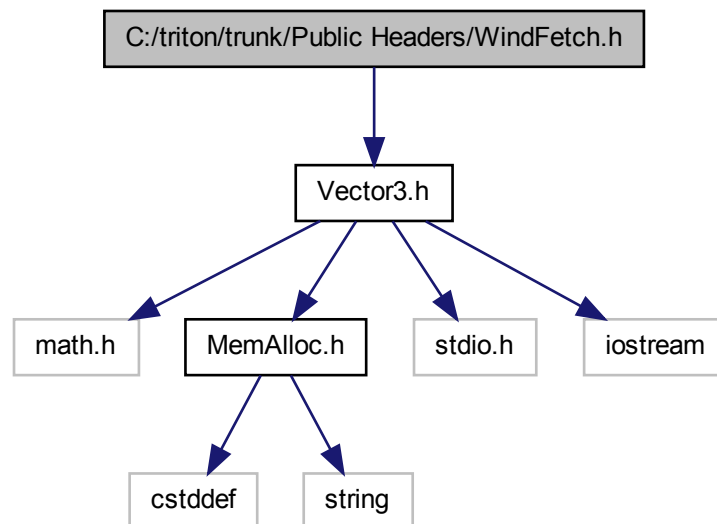
An object that generates a ship Kelvin wake as it moves.

21.13 C:/triton/trunk/Public Headers/WindFetch.h File Reference

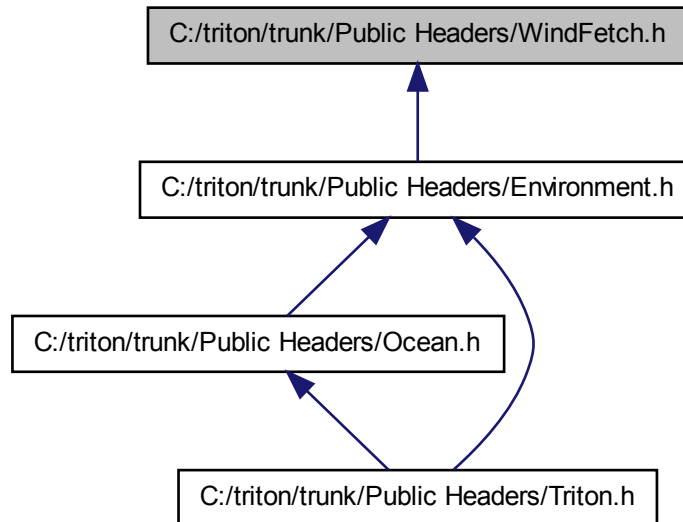
A localized or global area of wind of given speed and direction.

```
#include "Vector3.h"
```

Include dependency graph for WindFetch.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::WindFetch](#)

A localized or global area of wind of given speed and direction.

21.13.1 Detailed Description

A localized or global area of wind of given speed and direction.

Index

- ~Environment
 - Triton::Environment, 56
- ~Matrix3
 - Triton::Matrix3, 69
- ~Matrix4
 - Triton::Matrix4, 73
- ~Ocean
 - Triton::Ocean, 80
- AddWindFetch
 - Triton::Environment, 56
- C:/triton/trunk/Public Headers/Environment.h,
109
- C:/triton/trunk/Public Headers/Matrix3.h,
111
- C:/triton/trunk/Public Headers/Matrix4.h,
113
- C:/triton/trunk/Public Headers/MemAlloc.h,
115
- C:/triton/trunk/Public Headers/Ocean.h, 117
- C:/triton/trunk/Public Headers/RandomNum-
berGenerator.h, 118
- C:/triton/trunk/Public Headers/ResourceLoader.h,
120
- C:/triton/trunk/Public Headers/Triton.h, 122
- C:/triton/trunk/Public Headers/TritonCom-
mon.h, 123
- C:/triton/trunk/Public Headers/Vector3.h, 125
- C:/triton/trunk/Public Headers/Vector4.h, 126
- C:/triton/trunk/Public Headers/WakeGen-
erator.h, 128
- C:/triton/trunk/Public Headers/WindFetch.h,
130
- ClearGLErrors
 - Triton::Utils, 91
- ClearLocalization
 - Triton::WindFetch, 106
- ClearWindFetches
 - Triton::Environment, 56
- Create
 - Triton::Ocean, 80
- Cross
 - Triton::Vector3, 94
- CullSphere
 - Triton::Environment, 56
- D3D9DeviceLost
 - Triton::Ocean, 80
- D3D9DeviceReset
 - Triton::Ocean, 81
- Dot
 - Triton::Vector3, 94
 - Triton::Vector3f, 98
 - Triton::Vector4, 101
- Draw
 - Triton::Ocean, 81
- EnableSpray
 - Triton::Ocean, 81
- EnableWireframe
 - Triton::Ocean, 81
- Environment
 - Triton::Environment, 56
- FreeResource
 - Triton::ResourceLoader, 89
- FromRx
 - Triton::Matrix3, 70
- FromRy
 - Triton::Matrix3, 70
- FromRz
 - Triton::Matrix3, 70
- FromXYZ
 - Triton::Matrix3, 70
- GetAboveWaterVisibility
 - Triton::Environment, 57
- GetAmbientLightColor
 - Triton::Environment, 57
- GetBelowWaterVisibility
 - Triton::Environment, 57
- GetCameraMatrix

- Triton::Environment, 57
- GetCameraPosition
 - Triton::Environment, 58
- GetChoppiness
 - Triton::Ocean, 81
- GetConfigOption
 - Triton::Environment, 58
- GetCoordinateSystem
 - Triton::Environment, 58
- GetDepth
 - Triton::Ocean, 82
- GetDevice
 - Triton::Environment, 58
- GetDirectionalLightColor
 - Triton::Environment, 58
- GetDLLExtension
 - Triton::Utils, 91
- GetDLLPath
 - Triton::Utils, 91
- GetDLLSuffix
 - Triton::Utils, 91
- GetDX9Macros
 - Triton::Utils, 91
- GetEnvironmentMap
 - Triton::Environment, 58
- GetEnvironmentMapMatrix
 - Triton::Environment, 59
- GetFFTName
 - Triton::Ocean, 82
- GetHeight
 - Triton::Ocean, 82
- GetLightDirection
 - Triton::Environment, 59
- GetNumTriangles
 - Triton::Ocean, 83
- GetPlanarReflectionDisplacementScale
 - Triton::Environment, 59
- GetPlanarReflectionMap
 - Triton::Environment, 59
- GetPlanarReflectionMapMatrix
 - Triton::Environment, 59
- GetPosition
 - Triton::WakeGenerator, 104
- GetProjectionMatrix
 - Triton::Environment, 59
- GetRandomDouble
 - Triton::RandomNumberGenerator, 86
- GetRandomInt
 - Triton::RandomNumberGenerator, 87
- GetRandomNumberGenerator
 - Triton::Environment, 59
- GetRefractionColor
 - Triton::Ocean, 83
- GetRenderer
 - Triton::Environment, 60
- GetResourceLoader
 - Triton::Environment, 60
- GetRightVector
 - Triton::Environment, 60
- GetRow
 - Triton::Matrix4, 74
- GetSeaLevel
 - Triton::Environment, 60
- GetShaderObject
 - Triton::Ocean, 83
- GetUpVector
 - Triton::Environment, 60
- GetVelocity
 - Triton::WakeGenerator, 104
- GetWaveHeading
 - Triton::Ocean, 83
- GetWind
 - Triton::Environment, 60
- GetWindAtLocation
 - Triton::WindFetch, 106
- GetWorldUnits
 - Triton::Environment, 61
- Initialize
 - Triton::Environment, 61
- InverseCramers
 - Triton::Matrix4, 74
- IsDirectX
 - Triton::Environment, 61
- IsGeocentric
 - Triton::Environment, 62
- IsOpenGL
 - Triton::Environment, 62
- Length
 - Triton::Vector3, 94
 - Triton::Vector3f, 98
- LoadResource
 - Triton::ResourceLoader, 89
- Matrix3
 - Triton::Matrix3, 69
- Matrix4
 - Triton::Matrix4, 73
- Normalize

- Triton::Vector3, [94](#)
- Triton::Vector3f, [98](#)
- operator*
 - Triton::Matrix3, [70](#), [71](#)
 - Triton::Matrix4, [74](#)
 - Triton::Vector3, [94](#), [95](#)
 - Triton::Vector4, [101](#)
- operator+
 - Triton::Vector3, [95](#)
 - Triton::Vector3f, [98](#)
 - Triton::Vector4, [101](#)
- operator-
 - Triton::Vector3, [95](#)
 - Triton::Vector3f, [98](#)
 - Triton::Vector4, [101](#)
- operator==
 - Triton::Vector3, [95](#)
- PrintGLErrors
 - Triton::Utils, [91](#)
- Serialize
 - Triton::Vector3, [95](#)
- SetAboveWaterVisibility
 - Triton::Environment, [62](#)
- SetAllocator
 - Triton::Allocator, [50](#)
- SetAmbientLight
 - Triton::Environment, [62](#)
- SetBelowWaterVisibility
 - Triton::Environment, [62](#)
- SetCameraMatrix
 - Triton::Environment, [63](#)
- SetChoppiness
 - Triton::Ocean, [83](#)
- SetConfigOption
 - Triton::Environment, [63](#)
- SetDepth
 - Triton::Ocean, [84](#)
- SetDirectionalLight
 - Triton::Environment, [63](#)
- SetEnvironmentMap
 - Triton::Environment, [63](#)
- SetLicenseCode
 - Triton::Environment, [64](#)
- SetLocalization
 - Triton::WindFetch, [107](#)
- SetPlanarReflectionMap
 - Triton::Environment, [64](#)
- SetProjectionMatrix
 - Triton::Environment, [65](#)
- SetRandomNumberGenerator
 - Triton::Environment, [66](#)
- SetRefractionColor
 - Triton::Ocean, [84](#)
- SetResourceDirPath
 - Triton::ResourceLoader, [89](#)
- SetSeaLevel
 - Triton::Environment, [66](#)
- SetWind
 - Triton::WindFetch, [107](#)
- SetWorldUnits
 - Triton::Environment, [66](#)
- SimulateSeaState
 - Triton::Environment, [66](#)
- SprayEnabled
 - Triton::Ocean, [84](#)
- SquaredLength
 - Triton::Vector3, [95](#)
- ToFloatArray
 - Triton::Matrix3, [70](#)
 - Triton::Matrix4, [74](#)
- Transpose
 - Triton::Matrix3, [70](#)
 - Triton::Matrix4, [74](#)
- Triton::Allocator, [49](#)
 - SetAllocator, [50](#)
- Triton::Environment, [50](#)
 - ~Environment, [56](#)
 - AddWindFetch, [56](#)
 - ClearWindFetches, [56](#)
 - CullSphere, [56](#)
 - Environment, [56](#)
 - GetAboveWaterVisibility, [57](#)
 - GetAmbientLightColor, [57](#)
 - GetBelowWaterVisibility, [57](#)
 - GetCameraMatrix, [57](#)
 - GetCameraPosition, [58](#)
 - GetConfigOption, [58](#)
 - GetCoordinateSystem, [58](#)
 - GetDevice, [58](#)
 - GetDirectionalLightColor, [58](#)
 - GetEnvironmentMap, [58](#)
 - GetEnvironmentMapMatrix, [59](#)
 - GetLightDirection, [59](#)
 - GetPlanarReflectionDisplacementScale, [59](#)
 - GetPlanarReflectionMap, [59](#)

- GetPlanarReflectionMapMatrix, 59
- GetProjectionMatrix, 59
- GetRandomNumberGenerator, 59
- GetRenderer, 60
- GetResourceLoader, 60
- GetRightVector, 60
- GetSeaLevel, 60
- GetUpVector, 60
- GetWind, 60
- GetWorldUnits, 61
- Initialize, 61
- IsDirectX, 61
- IsGeocentric, 62
- IsOpenGL, 62
- SetAboveWaterVisibility, 62
- SetAmbientLight, 62
- SetBelowWaterVisibility, 62
- SetCameraMatrix, 63
- SetConfigOption, 63
- SetDirectionalLight, 63
- SetEnvironmentMap, 63
- SetLicenseCode, 64
- SetPlanarReflectionMap, 64
- SetProjectionMatrix, 65
- SetRandomNumberGenerator, 66
- SetSeaLevel, 66
- SetWorldUnits, 66
- SimulateSeaState, 66
- Triton::Matrix3, 67
 - ~Matrix3, 69
 - FromRx, 70
 - FromRy, 70
 - FromRz, 70
 - FromXYZ, 70
 - Matrix3, 69
 - operator*, 70, 71
 - ToFloatArray, 70
 - Transpose, 70
- Triton::Matrix4, 71
 - ~Matrix4, 73
 - GetRow, 74
 - InverseCramers, 74
 - Matrix4, 73
 - operator*, 74
 - ToFloatArray, 74
 - Transpose, 74
- Triton::MemObject, 75
- Triton::Ocean, 77
 - ~Ocean, 80
 - Create, 80
 - D3D9DeviceLost, 80
 - D3D9DeviceReset, 81
 - Draw, 81
 - EnableSpray, 81
 - EnableWireframe, 81
 - GetChoppiness, 81
 - GetDepth, 82
 - GetFFTName, 82
 - GetHeight, 82
 - GetNumTriangles, 83
 - GetRefractionColor, 83
 - GetShaderObject, 83
 - GetWaveHeading, 83
 - SetChoppiness, 83
 - SetDepth, 84
 - SetRefractionColor, 84
 - SprayEnabled, 84
- Triton::RandomNumberGenerator, 85
 - GetRandomDouble, 86
 - GetRandomInt, 87
- Triton::ResourceLoader, 87
 - FreeResource, 89
 - LoadResource, 89
 - SetResourceDirPath, 89
- Triton::Utils, 90
 - ClearGLErrors, 91
 - GetDLLExtension, 91
 - GetDLLPath, 91
 - GetDLLSuffix, 91
 - GetDX9Macros, 91
 - PrintGLErrors, 91
- Triton::Vector3, 92
 - Cross, 94
 - Dot, 94
 - Length, 94
 - Normalize, 94
 - operator*, 94, 95
 - operator+, 95
 - operator-, 95
 - operator==, 95
 - Serialize, 95
 - SquaredLength, 95
 - Unserialize, 95
 - Vector3, 94
 - x, 96
- Triton::Vector3f, 96
 - Dot, 98
 - Length, 98
 - Normalize, 98
 - operator+, 98

- operator-, 98
- Vector3f, 97, 98
- x, 98
- Triton::Vector4, 99
 - Dot, 101
 - operator*, 101
 - operator+, 101
 - operator-, 101
 - Vector4, 100
 - x, 101
- Triton::WakeGenerator, 101
 - GetPosition, 104
 - GetVelocity, 104
 - Update, 104
 - WakeGenerator, 103
- Triton::WindFetch, 104
 - ClearLocalization, 106
 - GetWindAtLocation, 106
 - SetLocalization, 107
 - SetWind, 107
 - WindFetch, 106
- Unserialize
 - Triton::Vector3, 95
- Update
 - Triton::WakeGenerator, 104
- Vector3
 - Triton::Vector3, 94
- Vector3f
 - Triton::Vector3f, 97, 98
- Vector4
 - Triton::Vector4, 100
- WakeGenerator
 - Triton::WakeGenerator, 103
- WindFetch
 - Triton::WindFetch, 106
- x
 - Triton::Vector3, 96
 - Triton::Vector3f, 98
 - Triton::Vector4, 101