

VRLab UMEÅ UNIVERSITY

2003-03-06

OsgVortex User Guide

OsgVortex User Guide

VRlab, UMEÅ UNIVERSITY

OsgVortex User Guide v0.40.1

© VRlab
Umeå University SE-901 87 Umeå
Phone 46 90 896 99 26 • Fax 46 90 786 61 26

Table of Contents



ABOUT THE	
DOCUMENTATION	1
WHAT IS OSGVORTEX	1
VORTEX	1
OPENSCENEGRAPH	2
DECLARATIVE SCRIPTS	2
USING OSGVORTEX	4
INSTALLING	4
DISTRIBUTION	4
SETTING UP THE	
ENVIRONMENT IN	
WINDOWS	4
DIRECTORY STRUCTURE	5
INSTALLATION	
VERIFICATION	6
SAMPLE SCRIPTS	7
PERFORMANCE	7
THE VIEWER	8
Key bindings	8
Navigating in the window	9
SYSTEM DESCRIPTION	10
OSGVORTEX SCRIPTS	11
Preprocessor	12
Mathematical Expressions	13
Constants	13
Tips how to format files	14
FILE FORMAT OF	
OSGVORTEX	14
World	15
Viewer	17
Material	18
Object	20
Joint	28
USING SOUND	34
WHAT IF IT GOES WRONG?	
	36
KNOWN BUGS	36



About the documentation

This document should be read in parallel to the Vortex simulation toolkit documentation, as it doesn't describe the basic functionality of all the parameters, joints, primitives etc. To gain more knowledge in that specific area, read the documentation that comes with Vortex.

It doesn't either try to teach how OpenSceneGraph works or is used, for this I refer to the documentation found in the distribution of OpenSceneGraph.

What is OsgVortex

This chapter will give an introduction to the tool OsgVortex and describe the overall system.

Designing a real-time simulation system is a complicated and large task. OsgVortex lets you describe a simulation system containing both visual and physical objects. There are few systems that encompass both visual as well as physical objects. VEGA Prime from MultiGen Inc. is one example where they have successfully done this. Unfortunately VEGA Prime is a huge and expensive system, not suitable for academic courses, simple prototyping etc.

OsgVortex is a small middle layer between two different packages; OpenSceneGraph and Vortex and is intended to be easy and intuitive to use. OsgVortex lets the user create objects in an ASCII script file. By running the sample viewer that comes with OsgVortex the script is executed and a simulation world is created. By allowing easy changing of values in the script, prototyping of a simulation world is possible.

Vortex

Vortex from Critical Mass Labs inc. Montreal Canada is a commercial toolkit (API) for handling rigid-body dynamics. It allows the programmer to create objects that has physical properties such as mass, velocity, geometry etc. By setting up a system with objects and joints connecting object, a

complicated fully physical system can be created. Vortex steps through the system for each time-step and solves the dynamic problems concerning forces generated from collisions, friction as well as constraints from joints. Vortex is said to be a real-time simulation toolkit as it can solve rather large systems where the simulation time is the same as the clock time. If the simulation time runs slower than the clock time (happens when the system becomes too large and Vortex is unable to solve the system fast enough) the simulation will look like it is running in slow-motion. If, on the other hand, the simulation time is running faster than the clock-time (if Vortex solves the system faster than the step-time) the system will run too fast, like looking at an old Charlie Chaplin movie. OsgVortex handles this by running the physical simulation in a separate thread. This will allow the physical simulation to run as fast or as slow as it has to ensure a real-time simulation.

As Vortex is not free it depends on a license, which is shipped together with OsgVortex.

OpenSceneGraph

OpenSceneGraph, OSG (www.openscenegraph.org) is an open source project lead by Robert Osfield, Scotland. It is an ambitious project with the goals of creating an efficient, portable, Object Oriented scenegraph for handling 3D graphics.

The status of OSG at the time of writing this manual is Beta version 0.9. It has loaders for many common formats, among one of the most common: 3DS, Open Flight, LWO, OBJ.

The benefit of using an open source project is the code availability as well as the short cycle of updates. There is a growing community around OSG which is rather efficient to fix bugs. Another benefit is that there are no licenses or fees involved.

A drawback is of course that newer versions often break the users code due to the quick changes.

Declarative scripts

To create a simulation with OsgVortex a script file has to be created. The script is parsed only during startup of the system. It is a *declarative lazy* parsing language. *Lazy*: only information relevant to OsgVortex will be used, i.e. it will discard any additional information in the script that OsgVortex is not interested of.

It is a *declarative* language, i.e. it can only be used to describe an initial state of the system, not how the system should behave in a functional manner. There are no loops to define iterative executions. Although there are methods to create several objects (more later on on this). At the writing time the following preprocessor directives are included: `#define` `#ifdef` `#else` `#endif` `#include` `#undef`

Mathematical expressions are allowed, examples of that are:

```
Size 2
Height Size*sin(PI)
```

SYSTEM OVERVIEW

The above example would declare two variables: `Size` and `Height`, where `Height` would have the value $2 \cdot \sin(3.1416) = 0.1096$

With this in mind let's install and run a sample application.

Using OsgVortex

This chapter will get you started with OsgVortex, setting up the environment and run sample simulations.

OsgVortex runs under several platforms, at the writing moment only Windows and Linux are tested. As it depends on OpenSceneGraph (runs under Linux, Solaris, Irix, Windows, HPUnix) and Vortex (runs under Irix, Windows and Linux) it will always be restricted by those dependencies.

Installing

The safest way of receiving the latest version is to either contact: info@vrlab.com or andersb@cs.umu.se.

Distribution

There are two different distributions, run-time and developer. Run-time contains a sample viewer executable, data and sample scripts. The developer distribution contains the above, but additionally also libraries and header files to generate new applications using the C++ API.

Environment Variables

The environment variables relevant to OsgVortex are:

VRLAB_PREFIX	Specify the path to the installation of OsgVortex.
OSGV_OUTPUT_LEVEL	Specifies the verbosity of OsgVortex. How much messages that will be printed to the console when running an OsgVortex application. 1 – Information messages and warnings will be printed 2 – Default level, Warnings will be printed. 3 – No information will be printed.

OSGV_FILE_PATH	A semicolon (in unix) and colon (in windows) separated list of directories in which OsgVortex will search for scripts, sounds and lightmaps.
OSG_FILE_PATH	A semicolon (in unix) and colon (in windows) separated list of directories in which OpenSceneGraph will search for textures and models.
PATH	Specifies the path to executable files (and dynamic libraries under windows).
LD_LIBRARY_PATH	(Only in unix). Specifies the search path for dynamic libraries.
MATHNGIN_LICENSE_FILE	Specifies the path to the file or the directory containing the license information for Vortex. Contact support@cm-labs.com for more information about Vortex.

Setting up the environment in Windows

Before running any other scripts or executables that comes with OsgVortex an environment variable named `VRLAB_PREFIX` must be set. It should point to the installation of OsgVortex. For example:

```
DOS> set VRLAB_PREFIX=c:\OsgVortex\
tcsh> setenv VRLAB_PREFIX /home/OsgVortex
```

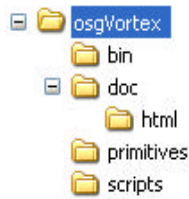
The distribution of OsgVortex comes with an initialization script that setup the environment for use with OsgVortex. To initialize the environment for OsgVortex do the following:

```
DOS> %VRLAB_PREFIX%\setup.bat
tcsh> source ${VRLAB_PREFIX}/setup.sh
```

Directory structure

The structure of OsgVortex is depending of the distribution type, Developer or Run-time.

The run-time distribution contains the following sub-directories:



bin contains executable files and dynamic library necessary to run OsgVortex.

doc contains the documentation of OsgVortex, both programmer and Users Guide.

primitives contains models and textures (images) that is necessary for OsgVortex.

scripts contains sample scripts for OsgVortex. Scripts that you write on your own does not have to reside in the directory scripts. They can be placed anywhere.

The root directory of OsgVortex also contains some important files:

setup.bat is a BAT file to setup the environment variables necessary to run OsgVortex. This file can be edited to reflect the path of your installation of OsgVortex.

test.bat is a BAT file that can be run to test if the installation was successful. (See Installation verification).

The developer distribution also contains include, lib and bin.

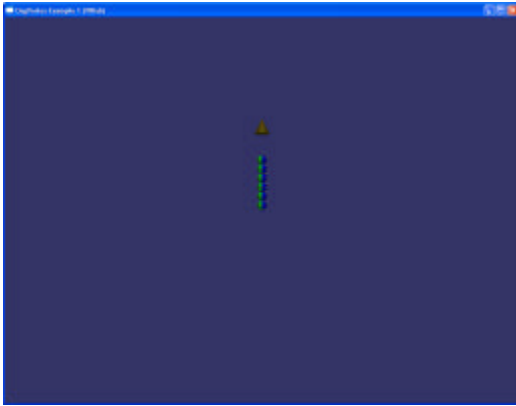
Installation verification

To verify that the installation of OsgVortex was successful there is scripts files that can be executed. Make sure that *setup.bat/.sh* is executed prior to any other scripts.

The *test.bat/.sh* script runs several sample scripts that exists in the scripts subdirectory. The file *test.bat/.sh* is located in *c:\OsgVortex* (or any other path where you previously installed OsgVortex).

```
DOS> test.bat
tcsh> ./test.sh
```

If the installation were successful a window with some 3D graphics will be displayed looking something like this:



If not, then the problem is most probably the settings of the environment variables.

There is also another script called rundemos.bat which runs another setup of sample scripts. You can run this by doing the following:

```
DOS> rundemos.bat  
tcsh> source rundemos.bat
```

Sample scripts

The best way to learn OsgVortex is to look at the sample scripts that come in the scripts sub-directory. There are scripts that demonstrate composite objects, visual objects, hinge joints, prismatic joints etc...

In the directory scripts\demos there are several more scripts created by students using OsgVortex. There are also examples of scripts generated with perl scripts. That is by using perl they output OsgVortex scripts that is read into the OsgVortex viewer. This makes it possible to use more elaborate algorithms to place all the objects and hinges in space.

Performance

The performance of your computer is very important to OsgVortex. If the performance is to low, the simulation will run in slow-motion, creating unrealistic environments.

By running the sample application test.bat the performance can be measured by pressing the key 'p' on the keyboard. In the top left corner a number will be displayed. This number indicates the frame-rate of the graphics. The frame-rate is 1/frequency of the updates of the graphical window. This can be strongly influenced by a number of things: performance of the

graphics card, number of objects in the scene, size of the window, complexity of objects in the scene and last but not least, if the graphics adapter is set to synchronization or not.

Under windows this will at least give a hint of the performance of the computer. On a fast machine it should be equal to or larger than the frequency of the display (usually 75-85Hz) the slower the machine is the lower the frame-rate will be. If the frame-rate is lower than 15 hz, the machine is close to be too slow for OsgVortex.

The viewer

The application OsgVortex is called a *viewer*, it will parse the script file you specify on the command line as an argument and when the parsing is successfully done, it will create a window and display the simulation environment.

Some important aspects are:

Scene – The graphical object that the simulation contains and can be seen in.

Viewpoint – The virtual camera that looks into the graphical scene. The viewpoint can be manipulated using the mouse.

Origo – The position of the scene that has the coordinates [0,0,0].

Key bindings

To control the viewer there are a lot of key bindings, i.e. actions connected to the keyboard. The most important are:

Key	Functionality
<i>space</i>	<i>Reset the position of the viewer (viewpoint)</i>
<i>esc</i>	<i>Quit the program</i>
<i>r</i>	<i>Reset the physical simulation</i>
<i>p</i>	<i>Print graphics statistics</i>
<i>k</i>	<i>Print physics statistics</i>
<i>T</i>	<i>Toggle texturing mode on/off</i>
<i>w</i>	<i>Toggle Wireframe/Solid/Point rendering mode</i>
<i>L</i>	<i>Toggle light on/off</i>

Navigating in the window

The viewpoint is manipulated using the mouse.

Left mouse is used for rotating the camera around origo in a mode called spaceball mode.

Right mouse is used for moving the camera forward/backward.

Middle mouse is used for pan up/down/left/right

Run the test.bat and try to navigate around in the scene.

Now when OsgVortex is installed and the installation is verified, we are ready to write our own scripts.

System description

This chapter will describe the *OsgVortex* system.

OsgVortex can be said to exist on two different levels: there is a Object oriented API (Application Programming Interface) written in C++ and a *Parser*. The parser is responsible for translating the *OsgVortex* scripts into calls in the *OsgVortex* API. So there is two ways of creating a simulation: either by writing a script or writing it directly using the C++ classes, or for that matter, both. A simulation can contain the following items:

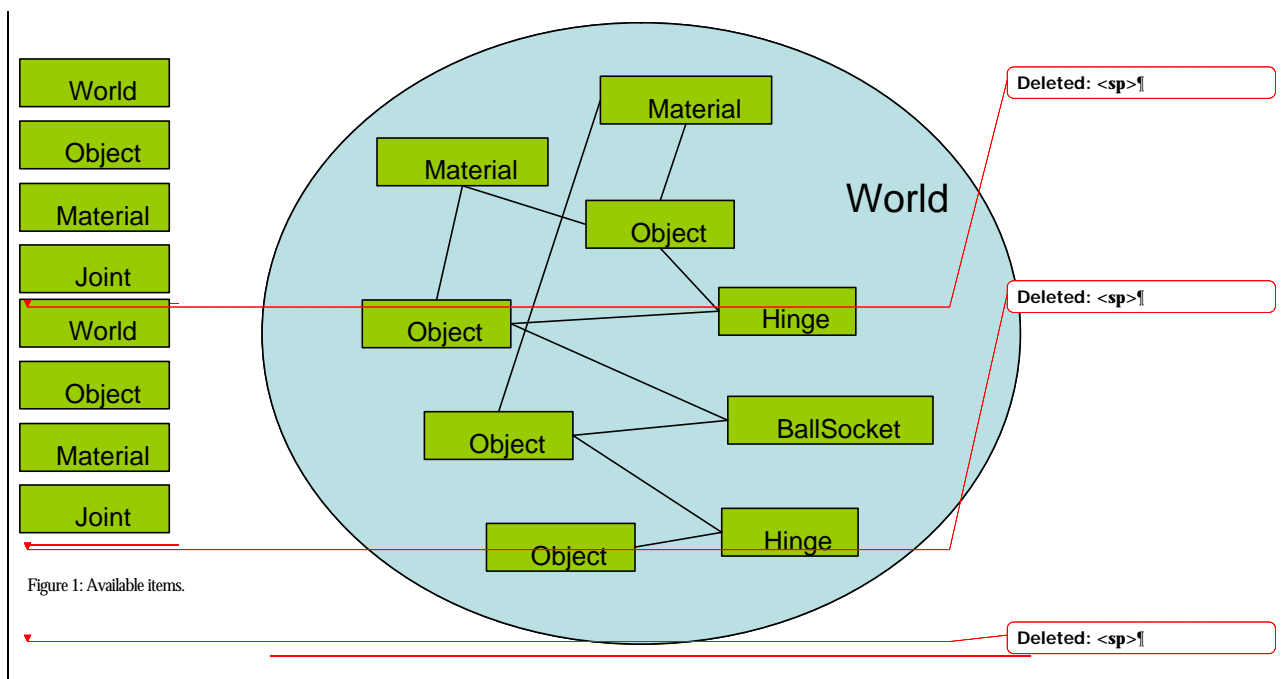


Figure 1: Available items.

Figure 2: A sample simulation world.

A simulation world is created by specifying one or more of these items in an *OsgVortex* script file. A *parser* will parse the script and use the appropriate C++ classes to build the world. When the world is created a *simulation engine* takes over and simulates the system.

OsgVortex scripts

This chapter will describe the file format used for creating OsgVortex simulations.

Scripts in OsgVortex are used for creating virtual environments containing visual as well as physical objects. The specifications are object oriented in its syntax. All entries in a script file are called *items*. An *item* is consists of two things: a *key* and *data*: `<Key>` `<Data>`

Valid keys must only contain alphanumeric characters and underscore '_':

Valid keys:

```
a_string
_another_key3
343
```

Invalid keys:

```
A string
:-)
a-value
```

The type of the key is derived from the type of the data.

Example of data	Derived type
1 2334	INT
1.0 1E10 0.234	FLOAT
"a string"	STRING
[2 34.3 4]	FLOAT VECTOR
{ member 45}	STRUCT (containing one INT member)

A *STRUCT* is a recursive datatype which can any of the other data types, include *STRUCTS*. The *key* is case-sensitive, so AnInt and Anint are treated as two separate variables.

An example of a defition is:

```
Object {
  Name "a namestring"
  VisualAttributes {
    Primitive "box"
  }
}
```

The above example is a struct containing two members, Name (which is a string) and VisualAttributes (wich in self is a struct containing one member, Primitive, which is a string).

Comments can be inserted into a script by using the // comment string. Everything after a // on a line is discarded by the script parser, unless it is inside a string. So string_val "//" is allowed.

Example of comments are:

```
Object { // An object will be created
  // The name of the object will be sven
  Name "sven"
} // End of struct
```

Preprocessor

To simplify development of simulations a few preprocessor directives has been added.

Name	Description	Example
#define <i>macro-name value</i>	Declares a macro, a text string (<i>macro-name</i>) that upon found in a script will be replaced with <i>value</i> .	#define HYP sqrt(a*a+b*b+c*c)
#undef <i>macro-name</i>	Undeclares a macro.	#undef HYP
#ifdef <i>macro-name</i> #else #endif	If the macro <i>macro-name</i> is defined, the code between this #ifdef and the following #else or #endif will be parsed. #ifdef can also be nested.	#ifdef HYP value 10 #ifdef URK value 34.54 #endif #else value 20 #endif

Table 1 Preprocessor directives.

Mathematical Expressions

Mathematical expressions are allowed in an OsgVortex script. Table 2 lists all predefined mathematical functions. The notation of expressions are standard using paranthesis:

```
avalue 34.4*(sin(3)/randInterval(0,1)*exp(3.4))
```

Name	Description	Example
sin, cos, tan, asin, acos, atan, sinh, cosh, tanh	Trigonometric functions	sin(3.4) sinh(0.3)
exp(x), log(x), log10(x), sqrt(x), floor(x), ceil(x), fabs(x), hypot		cos(3.4)
deg(x), rad(x)	Conversion between degrees and radians.	deg(0.3)
randInterval(low, high)	Random number generator. Generates a floating point random number in the interval <i>low</i> to <i>high</i> .	randInterval(0, 1)

Table 2 Predefined functions.

Constants

There are currently two constants defined in OsgVortex. These two constants can be used in expressions.

Name	Description
pi	The value of PI (3.14159265358979323846)
e	The natural base e (2.71828182845904523536).
INFINITY	The largest possible float representation according to Vortex.

Table 3 Predefined constants.

Tips how to format files

Indentation of struct members like in the above example are a clever way of making scripts more readable.

```
Astruct {
  astructmember 3
  substruct {
    astructmember
  }
}
```

A more readable example:

```
Astruct {
  astructmember 3
  substruct {
    astructmember
  }
}
```

File format of OsgVortex

Some keys are required and some have default values. In the detailed description of each key this will be pointed out.

There is a very important file namely **complete_syntax.osv**, this file contains the complete syntax of an OsgVortex script and is usually up to date. Make sure you read that file thoroughly. It can come in handy as a reference to the syntax.

The keys that OsgVortex currently recognizes are:

Key	Type	Required?	Description
Object	STRUCT	No	Creates an Object, it can be visual or physical or both.
Material	STRUCT	No	Creates a physical material. It defines the contact between two objects.
Viewer	STRUCT	No	Viewer specifications.
World	STRUCT	No	Specifies attributes that is needed to create a physical simulation world.
Joint	STRUCT	No	Creates a joint. Only physical right now.

Sound			Specifies a sound with its wave file and attributes.
SoundSystem	STRUCT	NO	Initializes the sound system used for spatial (3D) sound.
Shadow			Initializes the shadowing.
CollisionPair	STRUCT	No	Defines a pair of object that will/will not generate collision events.
KinematicLink	STRUCT	No	Defines a connection between a Kinematic object (an Object with PhysicalAttribute.Kinematic set to 1) and a Kinematic joint.

The tables below describing the different keys have the following structure: every key is defined as:

Name of key	Type	Default value	Range	Description
-------------	------	---------------	-------	-------------

If a key doesn't have a default value, i.e. it is required, this is indicated with *none*. If a key is a STRUCT then the key is printed in **bold** and will have its own table below.

The order in which the keys are specified is not important at all. Use common sense to arrange the file in the way that makes most sense to you.

World

World sets some global attributes for the whole physical simulation world. It can be to choose from different solvers, friction models etc...

Name	Type	Default	Range	Description
StepLength	FLOAT	0.01	0...	Defines the simulation step length.
Gravity	FLOAT VECTOR	[0 0 9.81]		Specifies the gravity in the simulation. [X, Y, Z]
AutoDisable	INT	0	0/1	Specifies whether objects should be taken out of the dynamic simulation when they are assumed to be standing still, and later on be

				activated during a collision event.
SafeTime	INT	1	0/1	
Gamma	FLOAT	8xStepLength	0...	If gamma = 0, no relaxation is done at all, and joints will drift apart as the simulation progresses. If gamma is large, a large correction is applied which attempts to zero the constraint violation. The product of gamma*h, where h is the time step, is preferably less than 0.5, and certainly less than 1.0. For most simulations a value of gamma in the range 0.1 through 0.8 will work well. The simulation may become unstable if gamma is chosen poorly. If the time step is varied frequently, the product of gamma*h should be held constant, by resetting gamma when the time step changes. Gamma is more sensitive to time step variation than epsilon.
Epsilon	FLOAT		0...	Epsilon has units of time squared over mass. You should think of 1/epsilon as roughly equivalent to a spring constant. If you know that your system has no singularity, you can let epsilon be as near 0 as you like. If you use a lot of really heavy masses, you should decrease epsilon from its default value. If you use a very small time step, you should consider using a much smaller epsilon.
IntegrationModel	STRING	"KeaStepper"	"StableStepper", "KeaStepper", "SimpleStepper"	
LCPSettings	STRUCT			Contains settings for the LCP solver

Members of World.LCPSettings

Name	Type	Default	Range	Description
MaxIterations	INT	40	0...	Maximum number of iterations used by LCP solver. This defaults to 10 and should only be changed if you have found problems with the defaults. If you set this too high, your applications could slow down in some cases when there are jamming situations with very many contacts with friction.

MaxBlockIterations	INT	20	0...	The LCP solver works using the faster block iterations by default. This can fail however so we force a ceiling on the number of lock iterations. By default, we use 10 i.e., the same as the max number of iterations. Do not change this unless you are certain you understand what it does
Tolerance	FLOAT	0.0001	0...	The LCP solver works using the faster block iterations by default. This can fail however so we force a ceiling on the number of block iterations. By default, we use 10 i.e., the same as the max number of iterations. Do not change this unless you are certain you understand what it does
DegeneracyFilter	FLOAT	0.0001	0...	This sets up a filter to ignore degenerate constraint. The number should be of the order of 1. The default is 3. A very large number will make all contacts very spongy. A very small number can produce inefficient results

Example:

```
World {
  StepLength 0.01
  Gravity [0 0 -9.81] // Gravity constant

  LCPSettings {
    MaxIterations 40
    MaxBlockIterations 20
    Tolerance 0.0001
    DegeneracyFilter 0.0001
  }
}
```

Viewer

The Viewer structure contains settings for the Window and Camera.

Name	Type	Default	Range	Description
Window	STRUCT			Specifies Window settings for the simulation.
Camera	STRUCTURE			Specifies initial camera position and orientation.
FOV	FLOAT			Specifies the <i>horizontal</i> and <i>vertical</i> field of view

	VECTOR[2]			
AspectRatio	FLOAT			Specifies the aspect ratio for the window.
SceneViewLightEnable	INT	1	0, 1	If equals to 0, the default existing scene view light (GL_LIGHT0) will be enabled. This can be shut off, to gain more control over the lighting of the scene.

Members of Viewer.Window

Name	Type	Default	Range	Description
Title	STRING			Specifies the title of the window.
Origin	INT VECTOR[2]	[0 0]		Specifies the lower left position of the window.
Size	INT VECTOR[2]	[1024 860]	0...	Specifies the size of window in pixels.
FullScreen	INT	0	0, 1	If equals to 1, the window will cover the whole display initially.

Example.

```
Viewer {
  Window {
    Title "OSG - Vortex demo"
    Origin [1 1]
    Size [1024 768]
    FullScreen 1
  }
  Camera {
    Eye [x y z]
    Up [x y z]
    Center [x y z]
  }
  FOV [80 80*sqrt(2)]
  SceneViewLightEnable
}
```

Material

Material is a specification of the contact parameters when two objects collide. It may sound strange to create one material for what happens when two objects collides, but a collision between one object is not very useful now is it? Sounds like the sound of one hand clap...

Name	Type	Default	Range	Description
ID	STRING	<i>required</i>		Specifies a unique identification of this material.
Tuple	FLOAT VECTOR	<i>required</i>		Actually one OsgVortex Material creates two Vortex materials. This TUPLE indicates the index of these two materials. When an object is created later on, the Material key in Object refers to this TUPLE.
Friction	FLOAT	<i>required</i>		Friction coefficient. 0-100
Restitution	FLOAT	<i>required</i>		Specifies how much energy is kept after a collision. 1 – all energy is kept, 0 none.
RestitutionThreshold	FLOAT			
Softness	FLOAT			
PrimarySlide	FLOAT			
SecondarySlide	FLOAT			
PrimarySlip	FLOAT			
SecondarySlip	FLOAT			
AdhesiveForce	FLOAT		0...	Adhesive force allows for objects that stick together, as if they were glued. It specifies the minimal force needed to separate the two objects.
FrictionModel	STRING	"Box"	"Box" "ScaledBox"	Specifies the friction model used when calculating friction between two objects. ScaledBox is a multipass method which could be more costly.

Example:

```
Material {
  ID "Wheel-floor"
  Tuple [1 0]
  Friction 10
```

```

    Restitution 0.1
    FrictionModel "ScaledBox"
}

```

Object

Object is the key to the whole simulation. Without objects there would be no simulation. An object can be *Visual* (i.e. have visual properties) and/or *Physical* (have physical properties). A Visual object can be seen in the simulation, a physical object can interact with other physical object in the simulation. This means that an object can be Physical only, i.e. it will appear in the simulation but not visually. This can sometimes come in handy to create object that connects to joints but are not visible.

Name	Type	Default	Range	Description
ID	STRING	<i>required</i>		Specifies an unique identification of this material.
Dynamic	INT	<i>required</i>	0/1	Specifies whether this is a static or a dynamic object.
Size	FLOAT VECTOR	<i>required</i>		Specifies the size of the object. This can be an array of different sizes depending on the primitive used later.
Orientation	FLOAT VECTOR	[0 0 0]		Rotation around X, Y, Z in Euler angles, right handed coordinate system.
Position	FLOAT VECTOR	[0 0 0]		Specifies the position in world coordinates
PhysicalAttributes	STRUCT			Specifies the physical attributes of an object
VisualAttributes	STRUCT			Specifies the visual attributes of an object.
CopyOp	STRUCT			Specifies how many copies of this object that will be created and how they will be placed.

Members of Object.PhysicalAttributes

Name	Type	Default	Range	Description
Enabled	INT	1	0/1	Specifies if this object should be enabled in the simulation or not?
Density	FLOAT	<i>required</i>	0...	Specifies the density of the object. The Mass is calculated from the primitive. This ONLY works for primitives. Density cannot be specified together with Mass.
Mass	FLOAT	<i>required</i>	0...	Specifies the mass of the object. Cannot be specified at the same time as Density.
AngularDamping	FLOAT		0.001	Specifies the rotational damping, sort of the internal resistance to rotational acceleration.
LinearDamping				Specifies the linear damping, the resistance linear acceleration.
Material	INT	<i>MdtDefaultMaterial</i> ()	0...	Specifies the material index for this object. This must be an integer specified in the Tuple key of a Material .
AngularVelocity	FLOAT VECTOR	[0 0 0]		Specifies the rotational
LinearVelocity		[0 0 0]		
MomentOfInertia				Specifies the Moment of Inertia Tensor.
Kinematic	INT	0		If set this object is a Kinematic object. This means that the Visual part of the Object, (the node in the scenegraph) will control the physical object. That is, the opposite of a normal Object. This also requires a KinematicLink to be created.
Geometry	STRUCT			Specifies the collision geometry of this Physical object. If no geometry is specified, no collisions with this object is

				possible.
Sensor	STRUCT			Specifies that this object is a Sensor object, i.e. it has only a collision geometry, no dynamic body. It will not take part of the dynamic simulation.
CompositeMembers	STRUCT			Specifies that this is a Composite object that is build up from other dynamic Objects.

Members of Object.PhysicalAttributes.Geometry

Name	Type	Default	Range	Description
Primitive	STRING		sphere ,box, cone, cylinder	Specifies the primitive that represents the object physically (collision). This also specifies the length of the size vector.
File	STRING			Not implemented yet.
ParseMethod	STRING			Not implemented yet.

Members of Object.PhysicalAttributes.Sensor

Name	Type	Default	Range	Description
FollowObject	STRING			Specifies the object to which this Sensor object will be attached. When the FollowObject moves, this object follows it.

Members of Object. VisualAttributes

Name	Type	Default	Range	Description
Geometry	STRUCT			Specifies the geometry of this Visual object.
Light	STRUCT			Specifies that this Visual Object also has a light source attached to it.
Lightmaps	STRUCT			If specified, pre-calculated lightmaps will be used for this model.
Shadower	INT	0	0/1	If true this object will generate shadows.
Shadowed	INT	0	0/1	If true, shadows will fall onto this object.

Members of Object. VisualAttributes.Geometry

Name	Type	Default	Range	Description
Primitive	STRING		sphere, box, cone, cylinder	Specifies the primitive that represents the object visually. This also specifies the length of the size vector. This key conflicts with File and cannot be specified at the same time.
File				Specifies the filename of a 3D object that should be loaded for this object. This key conflicts with Primitive and cannot be specified at the same time.
Scale	FLOAT VECTOR	[1 1 1]		Specifies how much the loaded File should be scaled. (Only valid when File is specified).
Orientation	FLOAT VECTOR	[x y z] or [x y z w]		Specifies a rotation that will be applied to the loaded visual model. If the size of the array is 3m then a euler rotation is applied, otherwise it is treated as a quaternion.
Optimize	INT	0	0/1	Specifies if Optimization of the Visual part of the object should be performed. For larger object this can improve performance.

Members of Object. VisualAttributes.Light

Name	Type	Default	Range	Description
Ambient	FLOAT VECTOR			Specifies a 4 dimensional array of Red., Green, Blue, Alpha intensity for ambient light.
Diffuse	FLOAT VECTOR			Specifies a 4 dimensional array of Red., Green, Blue, Alpha intensity for diffuse light.
Specular	FLOAT VECTOR			Specifies a 4 dimensional array of Red., Green, Blue, Alpha intensity for specular light.
SpotCutOff	FLOAT			
SpotExponent	FLOAT			
QuadraticAttenuation	FLOAT			
LinearAttenuation	FLOAT			
ConstantAttenuation	FLOAT			

Members of Object. VisualAttributes.Lightmaps

Name	Type	Default	Range	Description
Path	STRING			Specifies the path to the directory containing pre-calculated lightmaps.
MinRadius	FLOAT	0.5	0..	Objects with a smaller bounding sphere radius than this value will not be processed.
MaxSplit	INTEGER	10		Maximum number of splits for a geometry when the system tries to make planar surfaces. The more curved surface, the higher this value has to be included in the process.

LeaveNonPlanar	INTEGER	1		
SplitPrimitiveSets	INTEGER	1		

Members of Object.CopyOp

The id of the first object created will have ID as name. The successors will be called <id>_index. For example ID “ball” and CopyOp { Copies 10 } would generate ball, ball_1, ball_2, ... ball_9.

Name	Type	Default	Range	Description
Copies	FLOAT VECTOR			Specifies the number of copies that will be created totally of this object.
PositionIncrement	FLOAT VECTOR			Specifies the increment in position that will be added to each new object.

Members of Object.CompositeMembers

Name	Type	Default	Range	Description
Object	STRING			Specifies the ID of a member of this CompositeObject. There can be several Object in a CompositeObject.

Example 1: Visual object

```
//A minimum visual Object:
Object {
  ID "unique obj id"
  Dynamic 0 // non-moving
  Size [0.4] // size
  VisualAttributes { // Can be seen in the simulation
    Geometry {
      Primitive "sphere" // Is a sphere (with radius 0.4)
    }
  }
}
```

Example 1 will create a visual non-moving sphere with a radius of 0.4 at position [0 0 0] with no rotation applied.

Example 2: Physical object

```

Object {
  ID "physical invisible"
  Size [1 1 1]
  Position [0 -1 0.5]
  Dynamic 0
  PhysicalAttributes {
    Geometry {
      Primitive "box"
    }
  }
}

```

Example 2 will create a physical-only non-moving cube with sides 1, 1, 1 at position [0 -1 0.5] with no rotation applied. This object will not show up in the simulation visually, but it will exist as an physical object that other object can collide with.

Example 3: Visible and Physical object

```

Object {
  ID "fully dynamic object"
  Size [0.5 1]
  Position [2 -3 4]
  Orientation [45 0 0]
  Dynamic 1
  PhysicalAttributes {
    Mass 1
    Geometry {
      Primitive "cylinder"
    }
  }
  VisualAttributes {
    Geometry {
      Primitive "cylinder"
    }
  }
}

```

Example 3 will create a visible cylinder that is also represented in the physical simulation. It will start at position [2 -3 4] and will be rotated 45 degrees around X. It has a mass of 1.

Example 4: Initial state

```

Object {
  ID "moving object"
  Size [0.1 1]
  Position [2 -3 4]
  Orientation [0 0 0]
  Dynamic 1
  PhysicalAttributes {
    Mass 1
    Geometry {
      Primitive "cylinder"
    }
  }
  AngularVelocity [0 1.2 3]
}

```

```

    LinearVelocity [2 -3 4]
    LinearDamping 0.1
    AngularDamping 0.1
    MomentOfInertia [0.3 0 0 0 0.3 0 0 0 0.3]
  }
  VisualAttributes {
    Geometry {
      Primitive "cylinder"
    }
  }
}

```

Example 4 creates an object that has an initial velocity, both rotational and linear. It also sets the Moment of Inertia tensor.

Example 5: Copies

```

Object {
  ID "moving object"
  CopyOp {
    Copies 6
    PositionIncrement [0 2 0]
  }
  Size [0.1 1]
  Position [2 -3 4]
  Orientation [0 0 0]
  Dynamic 1
  PhysicalAttributes {
    Mass 1
    Geometry {
      Primitive "cylinder"
    }
    AngularVelocity [0 1.2 3]
    LinearVelocity [2 -3 4]
    LinearDamping 0.1
    AngularDamping 0.1
    MomentOfInertia [0.3 0 0 0 0.3 0 0 0 0.3]
  }
  VisualAttributes {
    Geometry {
      Primitive "cylinder"
    }
  }
}

```

Example 5 creates 6 cylinders placed at position [2 -3 4], [2 -1 4], [2 1 4] ... Each object will have the same attributes otherwise.

Joint

A joint is a constraint that describes a relationship between two objects. There are several joints and they all have different features.

Name	Type	Default	Range	Description
ID	STRING			Specifies a unique identification.
Object1	STRUCT	<i>required</i>		
Object2	STRUCT	<i>Not-required</i>		
Enabled	INT	1		Determines if the joint should be enabled or not.
Position	FLOAT VECTOR	[0 0 0]		Specifies the position of the joint.
Axis	FLOAT VECTOR			Specifies the rotation axis of the hinge.
UpperLimit	FLOAT			Specifies the upper stop limit for a joint.
LowerLimit	FLOAT			Specifies the lower stop limit for a joint.
UpperStiffness	FLOAT			Stiffness of the UpperLimit.
LowerStiffness	FLOAT			Stiffness of the LowerLimit
UpperDamping	FLOAT			Specifies the damping of the UpperLimit
LowerDamping	FLOAT			Specifies the damping of the LowerLimit
DesiredVelocity	FLOAT			Specifies the desired velocity of this joint.
MaxForce	FLOAT			Specifies the maximum force the motor can operate with on this joint to reach the DesiredVelocity.
BallSocket	STRUCT			Creates a BallSocket joint
Hinge	STRUCT			Creates a Hinge joint
Linear1	STRUCT			Creates a Linear1 joint
CarWheel	STRUCT			Creates a CarWheel joint.

Prismatic	STRUCT			Creates a Prismatic joint.
Angular3	STRUCT			Creates a Angular3 joint.
FixedPath	STRUCT			Creates a FixedPath joint.

Members of Joint.Object1 and Joint.Object2

Name	Type	Default	Range	Description
ID				ID of the object that we want to attach to this joint
Position				Attachment position of this object to the joint
AttachmentQuaternion	FLOAT VECTOR			Attachment quaternion for Object 1. Only valid for Kinematic and RPRO joints.
LinearVelocity	FLOAT VECTOR			Specifies the initial velocity for this attached object. This is to get faster stabilization. Only valid for Kinematic joints.

Members of Joint.BallSocket

Name	Type	Default	Range	Description

Members of Joint.Hinge

Name	Type	Default	Range	Description

Members of Joint.Spring

Name	Type	Default	Range	Description
NaturalLength	FLOAT		0..	Specifies the length at which this spring is at rest.
Stiffness	FLOAT		0..	Specifies the stiffness of the spring.
Damping	FLOAT		0..	Specifies the damping of the spring.

Members of Joint.Prismatic

Name	Type	Default	Range	Description

Members of Joint.Linear1

Name	Type	Default	Range	Description

Members of Joint.FixedPath

Name	Type	Default	Range	Description

Members of Joint.Angular3

Name	Type	Default	Range	Description

Members of Joint.RPRO

Name	Type	Default	Range	Description
RelativeQuaternion	FLOAT VECTOR	[0001]		
RelativeAngularVelocity	FLOAT VECTOR	[000]		Specifies the relative rotational velocity between object 1 and 2.
AngularStrength	FLOAT			Specifies the strength for the joint in rotations.
LinearStrength	FLOAT			Specifies the linear strength of the joint.

Members of Joint.Kinematic

Name	Type	Default	Range	Description
RelativeQuaternion	FLOAT VECTOR[4]	[0001]		
RelativeAngularVelocity	FLOAT VECTOR[3]	[000]		Specifies the relative rotational velocity between object 1 and 2.
AngularStrength	FLOAT			Specifies the maximum force this

				joint can apply.
LinearVelocity	FLOAT VECTOR[3]	[000]		Specifies the linear velocity of this joint
LinearStrength	FLOAT VECTOR[3]	[INFINITY, ...]	0.INFINITY	Specifies the linear strength of this joint. (Translations) This is the maximum force this joint can pass onto any attached object.
AngularStrength	FLOAT VECTOR	[INFINITY, ...]	0.INFINITY	Specifies the angular strength of this joint. (rotation) This is the maximum force this joint can pass onto any attached object.

Example 1: Hinge

```

Joint {
  ID "hinge2"
  Hinge {} // this is a hinge
  Axis [0 0 1]
  Position [1.1 0 1]
  Object1 {
    ID "door2" // 1st object that is attached
  }
  Object2 {
    ID "wall2" // 2nd object.
  }
  Enabled 1 // It is enabled by default
  UpperDamping 300
  LowerDamping 0
  MaxForce 10 // Maximum force to achieve DesiredVelocity
  DesiredVelocity -5
  UpperLimit 10
  LowerLimit 0
}

```

Example 2: Spring

```

Joint {
  ID "spring1"
  Spring {
    NaturalLength 0.1
    Stiffness 10
    Damping 1
  }

  Object1 {
    ID "sphere1"
    Position [sphere_pos.x sphere_pos.y sphere_pos.z]
  }

  Object2 {
    ID "box1"
    Position [box.pos.x box.pos.y box.pos.z]
  }
  Enabled 1
}

```

Members of SoundSystem

Name	Type	Default	Range	Description
MaxSoundSources	INT	32	0..	Specifies the maximum number of hardware soundsources that can be used in the simulation
DopplerFactor	FLOAT	1	0.1	Specifies the amount of Doppler effect
Gain	FLOAT	1	0..2	Specifies the total gain of the sound system. Can be used to set the total volume of all sounds.
MaxVelocity	FLOAT	2	0..	Specifies the max velocity that will be used for the doppler calculation.

Members of Sound

Name	Type	Default	Range	Description
ID	STRING			Specifies a unique name identifier for this sound.
File	STRING			Specifies the file path to the sample that will be played for this sound.
Play	INT	0	0,1	If true, this sound will be played immediately during startup of the

				simulation. This requires that AllocateSource also I set to 1.
AllocateSource	INT	0	0, 1	If true, a hardware sound source will be allocated for this sound. Otherwise it can only be used as a event driven sound.
Position	FLOAT VECTOR[3]	[000]		Initial position of the sound.
Ambient	INT	0	0, 1	IF the sound is to be perceived as ambient, that is always following the listener, no attenuation calculations will be computed.
Gain	FLOAT	1	0, 2	Specifies the relative volume for this sound.
Direction	FLOAT VECTOR[0]	[001]		Specifies the direction vector of this sound.
SoundCone	STRUCT			Specifies the sound emitting cone for this sound.
Relative	INT	0	0, 1	Specifies that the sound is always relative to the listener..
ReferenceDistance	FLOAT	1	0..	The distance under which the volume for the source would normally drop by half
MaxDistance	FLOAT			Used with the Inverse Clamped Distance Model to set the distance where there will no longer be any attenuation of the source.
RolloffFactor	FLOAT	1	0..	The rate by which the sound decreases in intensity with distance.
Pitch	FLOAT	1	0..2	The playback speed for the sound.

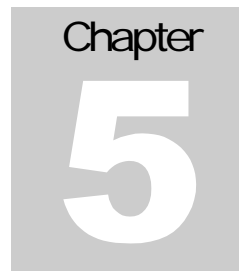
Using sound

Objects can be associated with a sound.

For examples, see sound.osv and sound_test.osv

sound.osv is an example where sound samples is attached to objects which can be moved in space. The associated sounds follows the objects as they move around.

sound_test.osv is another example of using sound. This example demonstrates a material sound, that is, if a sound sample is attached to a Material {} structure it can be made playing during a contact.



What if it goes wrong?

This chapter will describe the warning and error messages and what to do if the simulation doesn't work the way you want.

This could easily be the largest chapter in the whole manual. Unfortunately it is impossible to predict everything that can go wrong. There are many sources for mistakes. OsgVortex will try to inform the user of its current status. If something goes wrong the user should be informed.

There are three different levels of messages: Errors, Warning and Info.

Errors and warnings that OsgVortex will print out is important. Some of them is critical (Error) some are not critical but should not be ignored (Warning). There are also information messages (Info)

The environment variable `OSGV_OUTPUT_LEVEL` can be used to specify what levels of output you want from an OsgVortex application. See (Environment Variables).

Known bugs

- Expressions such as:

```
Astruct {
  sub1 {
    value 2
  }
  sub2 {
    x sub1.value // sub1.value is not known here.
  }
}
```

Where a substructure is trying to access a siblings member variable is not currently allowed (will be soon though).