# Gaffer Node Reference

**image engine**
**version 0.18.00, 2015**

# Gaffer

## Animation

Generates keyframed animation to be applied to plugs on other nodes.

**Plugs:**

**curves**

`Gaffer::Plug`

Stores animation curves. Rather than access these directly, prefer to use the Animation::acquire() method.

## Backdrop

A utility node which allows the positioning of other nodes on a coloured backdrop with optional text. Selecting a backdrop in the ui selects all the nodes positioned on it, and moving it moves them with it.

**Plugs:**

**description**

`Gaffer::StringPlug`

Text describing the contents of the backdrop - this will be displayed below the title.

**scale**

`Gaffer::FloatPlug`

Controls the size of the backdrop text.

**title**

`Gaffer::StringPlug`

The title for the backdrop - this will be displayed at the top of the backdrop.

## Box

A container for "subgraphs" - node networks which exist inside the Box and can be exposed by promoting selected internal plugs onto the outside of the Box.

Boxes can be used as an organisational tool for simplifying large graphs by collapsing them into sections which perform distinct tasks. They are also used for authoring files to be used with the Reference node.

## ContextVariablesComputeNode

Adds variables which can be referenced by upstream expressions.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**variables**

`Gaffer::CompoundDataPlug`

The variables to be added - arbitrary numbers of variables can be added here.

## Dot

A utility node which can be used for organising large graphs.

**Plugs:**

**label**

`Gaffer::StringPlug`

The label displayed when the type is set to custom.

**labelType**

`Gaffer::IntPlug`

The method used to apply an optional label to the dot. Using a node name is recommended, because it encourages the use of descriptive node names, and updates automatically when nodes are renamed or upstream connections change. The custom label does however provide more flexibility, since node names are restricted in the characters they can use.

## ExecutableOpHolder

Hosts Cortex Ops, allowing them to take perform tasks in graphs executed by Gaffer’s dispatchers. graph. This is most appropriate for hosting Ops which generate files on disk - for Ops which output objects directly, an OpHolder is a more appropriate host.

**Plugs:**

**dispatcher**

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

**batchSize**

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

**local**

`Gaffer::Plug`

Settings used by the local dispatcher.

**requirement**

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

**requirements**

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

## Expression

Utility node for computing values via scripted expressions.

**Plugs:**

## LocalDispatcher

Schedules execution of task graphs on the local machine. Tasks may be dispatched in the background to keep the UI responsive.

**Plugs:**

**executeInBackground**

`Gaffer::BoolPlug`

Executes the dispatched tasks in separate processes via a background thread. Foreground execution may still be forced for specific nodes using the dispatcher.local.executeInForeground plug on the node itself.

**frameRange**

`Gaffer::StringPlug`

The frame range to be used when framedMode is "CustomRange".

**framesMode**

`Gaffer::IntPlug`

Determines the active frame range to be dispatched as follows : - CurrentFrame uses the current timeline frame only. - FullRange uses the outer handles of the timeline (i.e. the full range of the script). - CustomRange uses a user defined range, as specified by the frameRange plug.

**ignoreScriptLoadErrors**

`Gaffer::BoolPlug`

Ignores errors loading the script when executing in the background. This is not recommended - fix the problem instead.

**jobName**

`Gaffer::StringPlug`

A descriptive name for the job.

**jobsDirectory**

`Gaffer::StringPlug`

A directory to store temporary files used by the dispatcher.

## LoopComputeNode

Applies a node network to an input iteratively.

## ObjectReader

Loads objects from disk using the readers provided by the Cortex project. In most cases it is preferable to use a dedicated SceneReader or ImageReader instead of this node.

**Plugs:**

**fileName**

`Gaffer::StringPlug`

The file to load.

**out**

`Gaffer::ObjectPlug`

The loaded object. Note that the ObjectToScene node may be used to convert this for use with the GafferScene module.

## ObjectWriter

Saves objects to disk using the writers provided by the Cortex project.

**Plugs:**

**dispatcher**

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

**batchSize**

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

**local**

`Gaffer::Plug`

Settings used by the local dispatcher.

**fileName**

`Gaffer::StringPlug`

The name of the file to write.

**in**

`Gaffer::ObjectPlug`

The object to be written to disk.

**parameters**

`Gaffer::CompoundPlug`

Additional parameters specific to the format of the file being written. These are created automatically based on the extension when the fileName is specified.

**requirement**

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

**requirements**

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

## OpHolder

Hosts Cortex Ops, allowing them to take part in computations performed in Gaffer’s dependency graph. For hosting Ops which create files, the ExecutableOpHolder is probably more appropriate.

**Plugs:**

## ParameterisedHolderDependencyNode

Hosts Cortex Parameterised classes

**Plugs:**

## ParameterisedHolderNode

Hosts Cortex Parameterised classes

**Plugs:**

## Preferences

A container for application preferences.

## ProceduralHolder

Hosts Cortex Procedurals, allowing them to be used in scenes generated by Gaffer.

**Plugs:**

**output**

`Gaffer::ObjectPlug`

The procedural itself. Connect this into an ObjectToScene node to allow it to be used by the GafferScene module.

## Random

Generates repeatable random values from a seed. This can be very useful for the procedural generation of variation. Numeric or colour values may be generated.

The random values are generated from a seed and a context variable - to get useful variation either the seed or the value of the context variable must be varied too.

**Plugs:**

**baseColor**

`Gaffer::Color3fPlug`

Used as the basis for the random colours generated for the outColor plug. All colours start with this value and then have a random HSV variation applied, using the ranges specified below.

**contextEntry**

`Gaffer::StringPlug`

The most important plug for achieving interesting variation. Should be set to the name of a context variable which will be different for each evaluation of the node. Good examples are "scene:path" to generate a different value per scene location, or "frame" to generate a different value per frame.

**floatRange**

`Gaffer::V2fPlug`

The minimum and maximum values that will be generated for the outFloat plug.

**hue**

`Gaffer::FloatPlug`

The +- range over which the hue of the base colour is varied.

**outColor**

`Gaffer::Color3fPlug`

Random colour output derived from seed, context variable, base colour, hue, saturation and value plugs.

**outFloat**

`Gaffer::FloatPlug`

Random floating point output derived from seed, context variable and float range plugs.

**saturation**

`Gaffer::FloatPlug`

The +- range over which the saturation of the base colour is varied.

**seed**

`Gaffer::IntPlug`

Seed for the random number generator. Different seeds produce different random numbers. When controlling two different properties using the same context variable, different seeds may be used to ensure that the generated values are different.

**value**

`Gaffer::FloatPlug`

The +- range over which the value of the base colour is varied.

## Reference

References a node network stored in another file. This can be used to share resources among scripts, build powerful non-linear workflows, and as the basis for custom asset management.

To generate a file to be referenced, build a network inside a Box node and then export it for referencing.

## ScriptNode

Defines a "script" - a Gaffer node network which can be saved to disk as a ".gfr" file and reloaded.

**Plugs:**

**fileName**

`Gaffer::StringPlug`

Where the script is stored.

**frameRange**

`Gaffer::ValuePlug`

Defines the start and end frames for the script. These don't enforce anything, but are typically used by dispatchers to control default frame ranges, and by the UI to define the range of the time slider.

**unsavedChanges**

`Gaffer::BoolPlug`

Indicates whether or not the script has been modified since it was last saved.

**variables**

`Gaffer::CompoundDataPlug`

Container for user-defined variables which can be used in expressions anywhere in the script.

## SwitchComputeNode

Chooses between multiple input connections, passing through the chosen input to the output.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**index**

`Gaffer::IntPlug`

The index of the input which is passed through. A value of 0 chooses the first input, 1 the second and so on. Values larger than the number of available inputs wrap back around to the beginning.

## SystemCommand

Runs system commands via a shell.

**Plugs:**

**command**

**dispatcher**

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

**batchSize**

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

**local**

`Gaffer::Plug`

Settings used by the local dispatcher.

**environmentVariables**

`Gaffer::CompoundDataPlug`

An arbitrary set of name/value pairs which will be set as environment variables when running the command.

**requirement**

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

**requirements**

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

**substitutions**

## TaskContextVariables

Adds variables which can be referenced by upstream expressions.

**Plugs:**

#### dispatcher

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

##### batchSize

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

##### local

`Gaffer::Plug`

Settings used by the local dispatcher.

#### requirement

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

#### requirements

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

#### variables

`Gaffer::CompoundDataPlug`

The variables to be added - arbitrary numbers of variables can be added here.

## TaskList

Used to collect executable tasks for dispatching all at once.

**Plugs:**

#### dispatcher

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

##### batchSize

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

##### local

`Gaffer::Plug`

Settings used by the local dispatcher.

#### requirement

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

#### requirements

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

## TimeWarpComputeNode

Changes the time at which upstream nodes are evaluated using the following formula :

```
upstreamFrame = frame * speed + offset
```

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**offset**

`Gaffer::FloatPlug`

Adds to the current frame value (after multiplication with speed).

**speed**

`Gaffer::FloatPlug`

Multiplies the current frame value.

## Wedge

Causes upstream nodes to be dispatched multiple times in a range of contexts, each time with a different value for a specified variable. This variable should be referenced in upstream expressions to apply variation to the tasks being performed. For instance, it could be used to drive a shader parameter to perform a series of "wedges" to demonstrate the results of a range of possible parameter values.

**Plugs:**

**colorSteps**

`Gaffer::IntPlug`

The number of steps in the wedge range defined when in "Colour Range" mode. The steps are distributed evenly from the start to the end of the ramp. Has no effect in other modes.

**dispatcher**

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

**batchSize**

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

**local**

`Gaffer::Plug`

Settings used by the local dispatcher.

**floatMax**

`Gaffer::FloatPlug`

The largest allowable value of the wedge range when the mode is set to "Float Range". Has no effect in other modes.

**floatMin**

`Gaffer::FloatPlug`

The smallest value of the wedge range when the mode is set to "Float Range". Has no effect in other modes.

**floatSteps**

`Gaffer::IntPlug`

The number of steps in the value range defined when in "Float Range" mode. The steps are distributed evenly between the min and max values. Has no effect in other modes.

**floats**

`Gaffer::FloatVectorDataPlug`

The list of values used when in "Float List" mode. Has no effect in other modes.

**indexVariable**

`Gaffer::StringPlug`

The name of an index context variable defined by the wedge. This is assigned values starting at 0 and incrementing for each new value - for instance a wedged float range might assign variable values of `0.25, 0,5, 0.75` or `0.1, 0,2, 0.3` but the corresponding index variable would take on values of `0, 1, 2` in both cases. The index variable is particularly useful for generating unique filenames when using a float range to perform wedged renders.

**intMax**

`Gaffer::IntPlug`

The largest allowable value of the wedge range when the mode is set to "Int Range". Has no effect in other modes.

**intMin**

`Gaffer::IntPlug`

The smallest value of the wedge range when the mode is set to "Int Range". Has no effect in other modes.

**intStep**

`Gaffer::IntPlug`

The step between successive values when the mode is set to "Int Range". Values are generated by adding this step to the minimum value until the maximum value is exceeded. Note that if (max - min) is not exactly divisible by the step then the maximum value may not be used at all. Has no effect in other modes.

**ints**

`Gaffer::IntVectorDataPlug`

The list of values used when in "Int List" mode. Has no effect in other modes.

**mode**

`Gaffer::IntPlug`

The method used to define the range of values used by the wedge. It is possible to define numeric or color ranges, and also to specify explicit lists of numbers or strings.

**ramp**

`Gaffer::SplinefColor3fPlug`

The range of colours used when the mode is set to "Colour Range". Has no effect in other modes.

**basis**

`Gaffer::ValuePlug`

!!!*EMPTY*!!!

**endPointMultiplicity**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

#### p0

`Gaffer::ValuePlug`

!!!*EMPTY*!!!

#### p1

`Gaffer::ValuePlug`

!!!*EMPTY*!!!

### requirement

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

### requirements

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

### strings

`Gaffer::StringVectorDataPlug`

The list of values used when in "String List" mode. Has no effect in other modes.

### variable

`Gaffer::StringPlug`

The name of the context variable defined by the wedge. This should be used in upstream expressions to apply the wedged value to specific nodes.

# GafferAppleseed

## AppleseedAttributes

Applies appleseed attributes to objects in the scene.

**Plugs:**

**attributes**

`Gaffer::CompoundDataPlug`

The attributes to be applied - arbitrary numbers of user defined attributes may be added as children of this plug via the user interface, or using the CompoundDataPlug API via python.

**cameraVisibility.value**

`Gaffer::BoolPlug`

Whether or not the object is visible to camera rays. To hide an object completely, use the visibility settings on the StandardAttributes node instead.

**lightVisibility.value**

`Gaffer::BoolPlug`

Whether or not the object is visible to light rays (whether or not it is visible to photons).

**shadowVisibility.value**

`Gaffer::BoolPlug`

Whether or not the object is visible to shadow rays (whether or not it casts shadows).

**diffuseVisibility.value**

`Gaffer::BoolPlug`

Whether or not the object is visible to diffuse rays - whether it casts bounce light or not.

**specularVisibility.value**

`Gaffer::BoolPlug`

Whether or not the object is visible in tight mirror reflections and refractions.

**glossyVisibility.value**

`Gaffer::BoolPlug`

Whether or not the object is visible in soft specular reflections and refractions.

**shadingSamples.value**

`Gaffer::IntPlug`

Number of samples to use when computing shading for the object.

**alphaMap.value**

`Gaffer::StringPlug`

Specifies a grayscale texture than can be used to efficiently discard unwanted parts of the surface of the object while computing ray intersections.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**global**

`Gaffer::BoolPlug`

Causes the attributes to be applied to the scene globals instead of the individual locations defined by the filter.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

## AppleseedLight

Loads appleseed lights.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

**name**

`Gaffer::StringPlug`

The name of the object in the output scene.

**out**

`GafferScene::ScenePlug`

The output scene.

**parameters**

`Gaffer::Plug`

The parameters of the light shader - these will vary based on the light type.

**sets**

`Gaffer::StringPlug`

A list of sets to include the object in. The names should be separated by spaces.

**transform**

`Gaffer::TransformPlug`

The transform applied to the object.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## AppleseedOptions

Sets global scene options applicable to the appleseed renderer. Use the StandardOptions node to set global options applicable to all renderers.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**options**

`Gaffer::CompoundDataPlug`

The options to be applied - arbitrary numbers of user defined options may be added as children of this plug via the user interface, or using the CompoundDataPlug API via python.

**renderPasses.value**

`Gaffer::IntPlug`

Number of render passes. When using photon mapping this is the number of progressive refinement passes used.

**sampler.value**

`Gaffer::StringPlug`

Sampler to use when generating samples

**aaSamples.value**

`Gaffer::IntPlug`

Number of anti-aliasing samples

**lightingEngine.value**

`Gaffer::StringPlug`

Lighting engine used when rendering

**meshFileFormat.value**

`Gaffer::StringPlug`

Mesh file format to use when exporting scenes to appleseed. It is recommended to set it to BinaryMesh as it is more efficient than Obj.

**shadingOverride.value**

`Gaffer::StringPlug`

Replaces all shaders in the scene by special diagnostics shaders that can visualize uvs, normals, … Useful for debugging scenes.

**environmentEDF.value**

`Gaffer::StringPlug`

Light to use as the environment.

**environmentEDFBackground.value**

`Gaffer::BoolPlug`

Whether or not the environment is visible in the background.

**drtIBL.value**

`Gaffer::BoolPlug`

Enable image-based lighting

**drtMaxBounces.value**

`Gaffer::IntPlug`

Maximum ray trace depth.If set to zero, use an unlimited number of bounces

**drtLightingSamples.value**

`Gaffer::FloatPlug`

Number of samples used to estimate direct lighting

**drtIBLSamples.value**

`Gaffer::FloatPlug`

Number of samples used to estimate environment lighting

**ptDirectLighting.value**

`Gaffer::BoolPlug`

Enable direct lighting

**ptIBL.value**

`Gaffer::BoolPlug`

Enable image-based lighting

**ptCaustics.value**

`Gaffer::BoolPlug`

Enable caustics

**ptMaxBounces.value**

`Gaffer::IntPlug`

Maximum number of path bounces.If set to zero, use an unlimited number of bounces

**ptLightingSamples.value**

`Gaffer::FloatPlug`

Number of samples used to estimate direct lighting

**ptIBLSamples.value**

`Gaffer::FloatPlug`

Number of samples used to estimate environment lighting

**ptMaxRayIntensity.value**

`Gaffer::FloatPlug`

Clamp intensity of rays (after the first bounce) to this value to reduce fireflies.Set to zero to disable

**photonType.value**

`Gaffer::StringPlug`

TODO: Not documented yet

**sppmDirectLighting.value**

`Gaffer::StringPlug`

Method used to estimate direct lighting

**sppmIBL.value**

`Gaffer::BoolPlug`

Enable image-based lighting

**sppmCaustics.value**

`Gaffer::BoolPlug`

Enable caustics

**sppmPhotonMaxBounces.value**

`Gaffer::IntPlug`

Maximum number of photon bounces.If set to zero, use an unlimited number of bounces

**sppmPathMaxBounces.value**

`Gaffer::IntPlug`

Maximum number of path bounces.If set to zero, use an unlimited number of bounces

**sppmLightPhotons.value**

`Gaffer::IntPlug`

Number of photons per render pass

**sppmEnvPhotons.value**

`Gaffer::IntPlug`

Number of environment photons per render pass

**sppmInitialRadius.value**

`Gaffer::FloatPlug`

Initial photon gathering radius in percent of the scene diameter.

**sppmMaxPhotons.value**

`Gaffer::IntPlug`

Maximum number of photons used to estimate radiance

**sppmAlpha.value**

`Gaffer::FloatPlug`

Evolution rate of photon gathering radius

**searchPath.value**

`Gaffer::StringPlug`

The filesystem paths where shaders and textures are searched for.

**numThreads.value**

`Gaffer::IntPlug`

Number of threads to use for rendering.Set to zero to use all CPU cores

**interactiveRenderFps.value**

`Gaffer::FloatPlug`

Maximum progressive rendering update rate in frames per second

**textureMem.value**

`Gaffer::IntPlug`

Texture cache size in bytes

**tileOrdering.value**

`Gaffer::StringPlug`

Tile rendering order

**out**

`GafferScene::ScenePlug`

The processed output scene.

## AppleseedRender

Performs offline batch rendering using the appleseed renderer. This is done in two phases - first the scene geometry is exported to mesh files and an appleseed project is generated, and then appleseed is invoked to render it.

**Plugs:**

**dispatcher**

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

**batchSize**

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

**local**

`Gaffer::Plug`

Settings used by the local dispatcher.

**fileName**

`Gaffer::StringPlug`

The name of the appleseed project file to be generated.

**in**

`GafferScene::ScenePlug`

The scene to be rendered.

**mode**

`Gaffer::StringPlug`

When in the standard "Render" mode, an appleseed project is generated and then renderered in appleseed. Alternatively, just the appleseed project can be generated and then another method can be used to post-process it or launch the render - a SystemCommand node may be useful for this.

**out**

`GafferScene::ScenePlug`

A pass-through of the input scene.

**bound**

`Gaffer::AtomicBox3fPlug`

!!!*EMPTY*!!!

**transform**

`Gaffer::M44fPlug`

!!!*EMPTY*!!!

**attributes**

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

**object**

`Gaffer::ObjectPlug`

!!!*EMPTY*!!!

**childNames**

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

**globals**

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

**setNames**

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

**set**

`GafferScene::PathMatcherDataPlug`

!!!*EMPTY*!!!

**requirement**

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

**requirements**

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

**verbosity**

`Gaffer::StringPlug`

Controls the verbosity of the appleseed renderer output.

## InteractiveAppleseedRender

Performs interactive renders using the appleseed renderer. The following edits are currently supported :

- Adding/removing lights
- Adjusting light parameters
- Moving the camera
- Changing shader assignments
- Editing shader networks
- Editing shader parameters

**Plugs:**

**in**

`GafferScene::ScenePlug`

The scene to be rendered.

**out**

`GafferScene::ScenePlug`

A direct pass-through of the input scene.

**bound**

`Gaffer::AtomicBox3fPlug`

!!!*EMPTY*!!!

**transform**

`Gaffer::M44fPlug`

!!!*EMPTY*!!!

**attributes**

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

**object**

`Gaffer::ObjectPlug`

!!!*EMPTY*!!!

**childNames**

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

**globals**

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

**setNames**

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

**set**

`GafferScene::PathMatcherDataPlug`

!!!*EMPTY*!!!

**state**

`Gaffer::IntPlug`

The interactive state.

**updateAttributes**

`Gaffer::BoolPlug`

When on, changes to attribute (and shaders) are reflected in the interactive render. When working with complex scenes, it may be worth turning this off to gain increased performance when only editing lights.

**updateCameras**

`Gaffer::BoolPlug`

When on, changes to the camera are reflected in the interactive render.

**updateCoordinateSystems**

`Gaffer::BoolPlug`

When on, changes to coordinate systems are reflected in the interactive render.

**updateLights**

`Gaffer::BoolPlug`

When on, changes to lights are reflected in the interactive render.

# GafferArnold

## ArnoldAttributes

Applies Arnold attributes to objects in the scene.

**Plugs:**

**attributes**

`Gaffer::CompoundDataPlug`

The attributes to be applied - arbitrary numbers of user defined attributes may be added as children of this plug via the user interface, or using the CompoundDataPlug API via python.

**cameraVisibility.value**

`Gaffer::BoolPlug`

Whether or not the object is visible to camera rays. To hide an object completely, use the visibility settings on the StandardAttributes node instead.

**shadowVisibility.value**

`Gaffer::BoolPlug`

Whether or not the object is visible to shadow rays (whether or not it casts shadows).

**reflectedVisibility.value**

`Gaffer::BoolPlug`

Whether or not the object is visible in tight mirror reflections.

**refractedVisibility.value**

`Gaffer::BoolPlug`

Whether or not the object is visible in refractions.

**diffuseVisibility.value**

`Gaffer::BoolPlug`

Whether or not the object is visible to diffuse rays - whether it casts bounce light or not.

**glossyVisibility.value**

`Gaffer::BoolPlug`

Whether or not the object is visible in soft specular reflections.

**subdivIterations.value**

`Gaffer::IntPlug`

The maximum number of subdivision steps to apply when rendering subdivision surface. To set an exact number of subdivisions, set the pixel error to 0 so that the maximum becomes the controlling factor. Use the MeshType node to ensure that a mesh is treated as a subdivision surface in the first place.

**subdivPixelError.value**

`Gaffer::FloatPlug`

The maximum allowable deviation from the true surface and the subdivided approximation. How the error is measured is determined by the metric below. Note also that the iterations value above provides a hard limit on the maximum number of subdivision steps, so if changing the pixel error setting appears to have no effect, you may need to raise the maximum.

#### subdivAdaptiveMetric.value

`Gaffer::StringPlug`

The metric used when performing adaptive subdivision as specified by the pixel error. The flatness metric ensures that the subdivided surface doesn't deviate from the true surface by more than the pixel error, and will tend to increase detail in areas of high curvature. The edge length metric ensures that the edge length of a polygon is never longer than the pixel metric, so will tend to subdivide evenly regardless of curvature - this can be useful when applying a displacement shader. The auto metric automatically uses the flatness metric when no displacement shader is applied, and the edge length metric when a displacement shader is applied.

### enabled

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

### filter

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

### global

`Gaffer::BoolPlug`

Causes the attributes to be applied to the scene globals instead of the individual locations defined by the filter.

### in

`GafferScene::ScenePlug`

The input scene

### out

`GafferScene::ScenePlug`

The processed output scene.

## ArnoldLight

Loads an Arnold light shader and uses it to output a scene with a single light.

**Plugs:**

### enabled

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

### name

`Gaffer::StringPlug`

The name of the object in the output scene.

### out

`GafferScene::ScenePlug`

The output scene.

### parameters

`Gaffer::Plug`

The parameters of the light shader - these will vary based on the light type.

**sets**

`Gaffer::StringPlug`

A list of sets to include the object in. The names should be separated by spaces.

**transform**

`Gaffer::TransformPlug`

The transform applied to the object.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## ArnoldOptions

Sets global scene options applicable to the Arnold renderer. Use the StandardOptions node to set global options applicable to all renderers.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**options**

`Gaffer::CompoundDataPlug`

The options to be applied - arbitrary numbers of user defined options may be added as children of this plug via the user interface, or using the CompoundDataPlug API via python.

**aaSamples.value**

`Gaffer::IntPlug`

Controls the number of rays per pixel traced from the camera. The more samples, the better the quality of antialiasing, motion blur and depth of field. The actual number of rays per pixel is the square of the AA samples value - so a value of 3 means 9 rays are traced, 4 means 16 rays are traced and so on.

**giDiffuseSamples.value**

`Gaffer::IntPlug`

Controls the number of rays traced when computing indirect illumination ("bounce light"). The number of actual diffuse rays traced is the square of this number.

### giGlossySamples.value

`Gaffer::IntPlug`

Controls the number of rays traced when computing glossy specular reflections. The number of actual specular rays traced is the square of this number.

### giRefractionSamples.value

`Gaffer::IntPlug`

Controls the number of rays traced when computing refractions. The number of actual specular rays traced is the square of this number.

### ignoreTextures.value

`Gaffer::BoolPlug`

Ignores all file textures, rendering as if they were all white.

### ignoreShaders.value

`Gaffer::BoolPlug`

Ignores all shaders, rendering as a simple facing ratio shader instead.

### ignoreAtmosphere.value

`Gaffer::BoolPlug`

Ignores all atmosphere shaders.

### ignoreLights.value

`Gaffer::BoolPlug`

Ignores all lights.

### ignoreShadows.value

`Gaffer::BoolPlug`

Skips all shadow calculations.

### ignoreSubdivision.value

`Gaffer::BoolPlug`

Treats all subdivision surfaces as simple polygon meshes instead.

### ignoreDisplacement.value

`Gaffer::BoolPlug`

Ignores all displacement shaders.

### ignoreBump.value

`Gaffer::BoolPlug`

Ignores all bump mapping.

### ignoreMotionBlur.value

`Gaffer::BoolPlug`

Ignores motion blur. Note that the turn off motion blur completely, it is more efficient to use the motion blur controls in the StandardOptions node.

### ignoreSSS.value

`Gaffer::BoolPlug`

Disables all subsurface scattering.

**textureSearchPath.value**

`Gaffer::StringPlug`

The locations used to search for texture files.

**proceduralSearchPath.value**

`Gaffer::StringPlug`

The locations used to search for procedural DSOs.

**shaderSearchPath.value**

`Gaffer::StringPlug`

The locations used to search for shader plugins.

**errorColorBadTexture.value**

`Gaffer::Color3fPlug`

The colour to display if an attempt is made to use a bad or non-existent texture.

**errorColorBadMesh.value**

`Gaffer::Color3fPlug`

The colour to display if bad geometry is encountered.

**errorColorBadPixel.value**

`Gaffer::Color3fPlug`

The colour to display for a pixel where a NaN is encountered.

**errorColorBadShader.value**

`Gaffer::Color3fPlug`

The colour to display if a problem occurs in a shader.

**out**

`GafferScene::ScenePlug`

The processed output scene.

## ArnoldRender

Performs offline batch rendering using the Arnold renderer. This is done in two phases - first a .ass file is generated and then Arnold is invoked to render it. Note though that the .ass file is lightweight, and contains little more than a procedural which will use Gaffer to generate the scene at render time.

**Plugs:**

**dispatcher**

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

**batchSize**

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

**local**

`Gaffer::Plug`

Settings used by the local dispatcher.

### fileName

`Gaffer::StringPlug`

The name of the .ass file to be generated.

### in

`GafferScene::ScenePlug`

The scene to be rendered.

### mode

`Gaffer::StringPlug`

When in the standard "Render" mode, an .ass file is generated and then renderered in Arnold. Alternatively, just the .ass file can be generated and then another method can be used to post-process it or launch the render - a SystemCommand node may be useful for this. Finally, an expanded .ass file may be generated - this will contain the entire expanded scene rather than just a procedural, and can be useful for debugging.

### out

`GafferScene::ScenePlug`

A pass-through of the input scene.

#### bound

`Gaffer::AtomicBox3fPlug`

!!!*EMPTY*!!!

#### transform

`Gaffer::M44fPlug`

!!!*EMPTY*!!!

#### attributes

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

#### object

`Gaffer::ObjectPlug`

!!!*EMPTY*!!!

#### childNames

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

#### globals

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

#### setNames

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

#### set

`GafferScene::PathMatcherDataPlug`

!!!*EMPTY*!!!

### requirement

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

**requirements**

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

**verbosity**

`Gaffer::IntPlug`

Controls the verbosity of the Arnold renderer output.

## ArnoldShader

Loads shaders for use in Arnold renderers. Use the ShaderAssignment node to assign shaders to objects in the scene.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**name**

`Gaffer::StringPlug`

The name of the shader being represented. This should be considered read-only. Use the Shader.loadShader() method to load a shader.

**parameters**

`Gaffer::CompoundPlug`

Where the parameters for the shader are represented.

**type**

`Gaffer::StringPlug`

The type of the shader being represented. This should be considered read-only. Use the Shader.loadShader() method to load a shader.

# GafferCortex

## ExecutableOpHolder

Hosts Cortex Ops, allowing them to take perform tasks in graphs executed by Gaffer’s dispatchers. graph. This is most appropriate for hosting Ops which generate files on disk - for Ops which output objects directly, an OpHolder is a more appropriate host.

**Plugs:**

**dispatcher**

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

**batchSize**

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

**local**

`Gaffer::Plug`

Settings used by the local dispatcher.

**requirement**

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

**requirements**

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

## ObjectReader

Loads objects from disk using the readers provided by the Cortex project. In most cases it is preferable to use a dedicated SceneReader or ImageReader instead of this node.

**Plugs:**

**fileName**

`Gaffer::StringPlug`

The file to load.

**out**

`Gaffer::ObjectPlug`

The loaded object. Note that the ObjectToScene node may be used to convert this for use with the GafferScene module.

## ObjectWriter

Saves objects to disk using the writers provided by the Cortex project.

**Plugs:**

**dispatcher**

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

**batchSize**

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

**local**

`Gaffer::Plug`

Settings used by the local dispatcher.

**fileName**

`Gaffer::StringPlug`

The name of the file to write.

**in**

`Gaffer::ObjectPlug`

The object to be written to disk.

**parameters**

`Gaffer::CompoundPlug`

Additional parameters specific to the format of the file being written. These are created automatically based on the extension when the fileName is specified.

**requirement**

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

**requirements**

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

## OpHolder

Hosts Cortex Ops, allowing them to take part in computations performed in Gaffer's dependency graph. For hosting Ops which create files, the ExecutableOpHolder is probably more appropriate.

**Plugs:**

## ParameterisedHolderDependencyNode

Hosts Cortex Parameterised classes

**Plugs:**

## ParameterisedHolderNode

Hosts Cortex Parameterised classes

**Plugs:**

## ProceduralHolder

Hosts Cortex Procedurals, allowing them to be used in scenes generated by Gaffer.

**Plugs:**

**output**

`Gaffer::ObjectPlug`

The procedural itself. Connect this into an ObjectToScene node to allow it to be used by the GafferScene module.

# GafferImage

## CDL

Applies color transformations provided by OpenColorIO via an OCIO CDLTransform.

**Plugs:**

**direction**

`Gaffer::IntPlug`

The direction to perform the color transformation.

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**offset**

`Gaffer::Color3fPlug`

Offset for the ASC CDL color correction formula.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**power**

`Gaffer::Color3fPlug`

Power for the ASC CDL color correction formula.

**saturation**

`Gaffer::FloatPlug`

Saturation from the v1.2 release of the ASC CDL color correction formula.

**slope**

`Gaffer::Color3fPlug`

Slope for the ASC CDL color correction formula.

## Clamp

Clamps channel values so that they fit within a specified range. Clamping is performed for each channel individually, and out-of-range colours may be highlighted by setting them to a value different to the clamp threshold itself.

**Plugs:**

**channels**

`GafferImage::ChannelMaskPlug`

The subset of channels to operate on.

#### enabled

`Gaffer::BoolPlug`

Turns the node on and off.

#### in

`GafferImage::ImagePlug`

The input image

#### max

`Gaffer::Color4fPlug`

The maximum value - values above this will be clamped if maxEnabled is on.

#### maxClampTo

`Gaffer::Color4fPlug`

By default, values above the maximum value are clamped to the maximum value itself. If maxClampToEnabled is on, they are instead set to this value. This can be useful for highlighting out-of-range values.

#### maxClampToEnabled

`Gaffer::BoolPlug`

Turns on the effect of maxClampTo, allowing out of range values to be highlighted.

#### maxEnabled

`Gaffer::BoolPlug`

Turns on clamping for values above the max value.

#### min

`Gaffer::Color4fPlug`

The minimum value - values below this will be clamped if minEnabled is on.

#### minClampTo

`Gaffer::Color4fPlug`

By default, values below the minimum value are clamped to the minimum value itself. If minClampToEnabled is on, they are instead set to this value. This can be useful for highlighting out-of-range values.

#### minClampToEnabled

`Gaffer::BoolPlug`

Turns on the effect of minClampTo, allowing out of range values to be highlighted.

#### minEnabled

`Gaffer::BoolPlug`

Turns on clamping for values below the min value.

#### out

`GafferImage::ImagePlug`

The output image generated by this node.

## ColorSpace

Applies colour transformations provided by OpenColorIO. Configs are loaded from the configuration specified by the OCIO environment variable.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**inputSpace**

`Gaffer::StringPlug`

The colour space of the input image.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**outputSpace**

`Gaffer::StringPlug`

The colour space of the output image.

## Constant

Outputs an image of a constant flat colour.

**Plugs:**

**color**

`Gaffer::Color4fPlug`

The colour of the image.

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**format**

`GafferImage::FormatPlug`

The resolution and aspect ratio of the image.

**displayWindow**

`Gaffer::Box2iPlug`

!!!*EMPTY*!!!

**min**

`Gaffer::V2iPlug`

!!!*EMPTY*!!!

**x**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**max**

`Gaffer::V2iPlug`

!!!*EMPTY*!!!

**x**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**pixelAspect**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## CopyImageMetadata

Copies metadata entries from the second image to the first image based on name. If those entries already exist in the incoming image metadata, their values will be overwritten.

**Plugs:**

**copyFrom**

`GafferImage::ImagePlug`

The image to copy the metadata entries from.

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**invertNames**

`Gaffer::BoolPlug`

When on, matching names are ignored, and non-matching names are copied instead.

**names**

`Gaffer::StringPlug`

The names of metadata entries to be copied. This is a space separated list of entry names, which accepts Gaffer's standard string wildcards.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## Crop

Modifies the Data and/or Display Window, in a way that is either user-defined, or can be driven by the existing Data or Display Window.

**Plugs:**

**affectDataWindow**

`Gaffer::BoolPlug`

Whether to intersect the defined area with the input Data Window. It will never pad black onto the Data Window, it will only ever reduce the existing Data Window.

**affectDisplayWindow**

`Gaffer::BoolPlug`

Whether to assign a new Display Window based on the defined area.

**area**

`Gaffer::Box2iPlug`

The custom area to set the Data/Display Window to. This plug is only used if *Area Source* is set to Custom.

**min**

`Gaffer::V2iPlug`

!!!*EMPTY*!!!

**x**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**max**

`Gaffer::V2iPlug`

!!!*EMPTY*!!!

**x**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**areaSource**

`Gaffer::IntPlug`

Where to source the actual area to use. If this is set to DataWindow, it will use the input's Data Window, if it is set to DisplayWindow, it will use the input's Display Window, and if it is set to Custom, it will use the Area plug.

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**resetOrigin**

`Gaffer::BoolPlug`

Shifts the cropped image area back to the origin, so that the bottom left of the display window is at ( 0, 0 ).

## DeleteChannels

Deletes channels from an image.

**Plugs:**

**channels**

`GafferImage::ChannelMaskPlug`

The names of the channels to be deleted (or kept if the mode is set to Keep).

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**mode**

`Gaffer::IntPlug`

Defines how the channels listed in the channels plug are treated. Delete mode deletes the listed channels. Keep mode keeps the listed channels, deleting all others.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## DeleteImageMetadata

Deletes metadata entries from an image based on name.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**invertNames**

`Gaffer::BoolPlug`

When on, matching names are kept, and non-matching names are removed.

**names**

`Gaffer::StringPlug`

The names of metadata entries to be removed. This is a space separated list of entry names, which accepts Gaffer’s standard string wildcards.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## Display

Interactively displays images as they are rendered.

This node runs a server on a background thread, allowing it to receive images from both local and remote render processes. To set up a render to output to the Display node, use an Outputs node with an Interactive output configured to render to the same port as is specified on the Display node.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**port**

`Gaffer::IntPlug`

The port number on which to run the display server. Outputs which specify this port number will appear in this node - use multiple nodes with different port numbers to receive multiple images at once.

## DisplayTransform

Applies color transformations provided by OpenColorIO via a DisplayTransform file and OCIO FileTransform.

**Plugs:**

**display**

`Gaffer::StringPlug`

The name of the display to use.

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**inputColorSpace**

`Gaffer::StringPlug`

The colour space of the input image.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**view**

`Gaffer::StringPlug`

The name of the view to use.

## Grade

Performs a simple per-channel colour grading operation as follows :

A = multiply * (gain - lift) / (whitePoint - blackPoint) B = offset + lift - A * blackPoint result = pow( A * input + B, 1/ gamma )

See the descriptions for individual plug for a slightly more practical explanation of the formula.

**Plugs:**

**blackClamp**

`Gaffer::BoolPlug`

Clamps input values so they don’t go below 0.

**blackPoint**

`Gaffer::Color3fPlug`

The input colour which is considered to be "black". This colour is remapped to the lift value in the output image.

**channels**

`GafferImage::ChannelMaskPlug`

The subset of channels to operate on.

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**gain**

`Gaffer::Color3fPlug`

The colour that input pixels at the whitePoint become in the output image. This can be thought of as defining the lighter values of the image.

**gamma**

`Gaffer::Color3fPlug`

A gamma correction applied after all the remapping defined above.

**in**

`GafferImage::ImagePlug`

The input image

**lift**

`Gaffer::Color3fPlug`

The colour that input pixels at the blackPoint become in the output image. This can be thought of as lifting the darker values of the image.

**multiply**

`Gaffer::Color3fPlug`

An additional multiplier on the output values.

**offset**

`Gaffer::Color3fPlug`

An additional offset added to the output values.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**whiteClamp**

`Gaffer::BoolPlug`

Clamps output values so they don’t go above 1.

**whitePoint**

`Gaffer::Color3fPlug`

The input colour which is considered to be "white". This colour is remapped to the gain value in the output image.

## ImageContextVariables

Defines variables which can be referenced by upstream expressions.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**variables**

`Gaffer::CompoundDataPlug`

The variables to be defined - arbitrary numbers of variables can be added here.

## ImageLoop

Applies a user defined processing loop to an image. The content of the loop is defined by the node network placed between the previous and next plugs. The input image is sent around this loop for a set number of iterations and then emerges as the output image.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**indexVariable**

`Gaffer::StringPlug`

The name of a context variable used to specify the index of the current iteration. This can be referenced from expressions within the loop network to modify the operations performed during each iteration of the loop.

**iterations**

`Gaffer::IntPlug`

The number of times the loop is applied to form the output image.

**next**

`GafferImage::ImagePlug`

The image to be used as the start of the next iteration of the loop.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**previous**

`GafferImage::ImagePlug`

The result from the previous iteration of the loop, or the input image if no iterations have been performed yet. The content of the loop is defined by feeding this previous result through the image processing nodes of choice and back around into the next plug.

## ImageMetadata

Adds arbitrary metadata entires to an image. If those entries already exist in the incoming image metadata, their values will be overwritten.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**metadata**

`Gaffer::CompoundDataPlug`

The metadata to be applied - arbitrary numbers of user defined metadata may be added as children of this plug via the user interface, or using the CompoundDataPlug python API

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## ImageReader

Reads image files from disk using OpenImageIO. All file types supported by OpenImageIO are supported by the ImageReader.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**fileName**

`Gaffer::StringPlug`

The name of the file to be read. File sequences with arbitrary padding may be specified using the # character as a placeholder for the frame numbers.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**refreshCount**

`Gaffer::IntPlug`

May be incremented to force a reload if the file has changed on disk - otherwise old contents may still be loaded via Gaffer's cache.

## ImageSampler

Samples image colour at a specified pixel location.

**Plugs:**

**color**

`Gaffer::Color4fPlug`

The sampled colour.

**filter**

`GafferImage::FilterPlug`

The filter used to generate interpolated pixel values.

**image**

`GafferImage::ImagePlug`

The image to be sampled.

**pixel**

`Gaffer::V2fPlug`

The coordinates of the pixel to sample. These can have fractional values - the filter will be used to generate appropriate interpolate values.

## ImageStats

Calculates minimum, maximum and average colours for a region of an image. These outputs can then be used to drive other plugs within the node graph.

**Plugs:**

**average**

`Gaffer::Color4fPlug`

The per-channel mean values computed from the input image region.

**channels**

`GafferImage::ChannelMaskPlug`

The names of the channels to be analysed.

**in**

`GafferImage::ImagePlug`

The input image to be analysed.

**max**

`Gaffer::Color4fPlug`

The per-channel maximum values computed from the input image region.

**min**

`Gaffer::Color4fPlug`

The per-channel minimum values computed from the input image region.

**regionOfInterest**

`Gaffer::Box2iPlug`

The region of the image to be analysed.

**min**

`Gaffer::V2iPlug`

!!!*EMPTY*!!!

**max**

`Gaffer::V2iPlug`

!!!*EMPTY*!!!

## ImageSwitch

Chooses between multiple input images, passing through the chosen input to the output.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`Gaffer::ArrayPlug`

The input images

**index**

`Gaffer::IntPlug`

The index of the input which is passed through. A value of 0 chooses the first input, 1 the second and so on. Values larger than the number of available inputs wrap back around to the beginning.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## ImageTimeWarp

Changes the time at which upstream nodes are evaluated using the following formula :

```
upstreamFrame = frame * speed + offset
```

Note that this node does not perform frame blending of any sort - it is the responsibility of nodes upstream to do this.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**offset**

`Gaffer::FloatPlug`

Adds to the current frame value (after multiplication with speed).

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**speed**

`Gaffer::FloatPlug`

Multiplies the current frame value.

## ImageTransform

Scales, rotates and translates an image within its display window. Note that although the format is not changed, the data window is expanded to include the portions of the image which have been transformed outside of the display window, and these out-of-frame pixels can still be used by downstream nodes.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**filter**

`GafferImage::FilterPlug`

The pixel filter used when transforming the image. Each filter provides different tradeoffs between sharpness and the danger of aliasing or ringing.

**in**

`GafferImage::ImagePlug`

The input image

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**transform**

`Gaffer::Transform2DPlug`

The transformation to be applied to the image. The translate and pivot values are specified in pixels, and the rotate value is specified in degrees.

**translate**

`Gaffer::V2fPlug`

!!!*EMPTY*!!!

**x**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V2fPlug`

!!!*EMPTY*!!!

**x**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V2fPlug`

!!!*EMPTY*!!!

**x**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

## ImageWriter

Writes image files to disk using OpenImageIO. All file types supported by OpenImageIO are supported by the ImageWriter.

**Plugs:**

**channels**

`GafferImage::ChannelMaskPlug`

The channels to be written to the file.

**dispatcher**

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

**batchSize**

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

**local**

`Gaffer::Plug`

Settings used by the local dispatcher.

**dpx**

`Gaffer::ValuePlug`

Format options specific to DPX files.

**dataType**

`Gaffer::StringPlug`

The data type to be written to the DPX file.

**field3d**

`Gaffer::ValuePlug`

Format options specific to Field3D files.

**mode**

`Gaffer::IntPlug`

The write mode for the Field3D file - scanline or tiled data.

**dataType**

`Gaffer::StringPlug`

The data type to be written to the Field3D file.

**fileName**

`Gaffer::StringPlug`

The name of the file to be written. File sequences with arbitrary padding may be specified using the # character as a placeholder for the frame numbers.

**fits**

`Gaffer::ValuePlug`

Format options specific to FITS files.

**dataType**

`Gaffer::StringPlug`

The data type to be written to the FITS file.

**iff**

`Gaffer::ValuePlug`

Format options specific to IFF files.

**mode**

`Gaffer::IntPlug`

The write mode for the IFF file - scanline or tiled data.

**in**

`GafferImage::ImagePlug`

The image to be written to disk.

**jpeg**

`Gaffer::ValuePlug`

Format options specific to Jpeg files.

**compressionQuality**

`Gaffer::IntPlug`

The compression quality for the Jpeg file to be written. A value between 0 (low quality, high compression) and 100 (high quality, low compression).

**jpeg2000**

`Gaffer::ValuePlug`

Format options specific to Jpeg2000 files.

**dataType**

`Gaffer::StringPlug`

The data type to be written to the Jpeg2000 file.

**openexr**

`Gaffer::ValuePlug`

Format options specific to OpenEXR files.

**mode**

`Gaffer::IntPlug`

The write mode for the OpenEXR file - scanline or tiled data.

**compression**

`Gaffer::StringPlug`

The compression method to use when writing the OpenEXR file.

**dataType**

`Gaffer::StringPlug`

The data type to be written to the OpenEXR file.

**out**

`GafferImage::ImagePlug`

A pass-through of the input image.

**format**

`GafferImage::AtomicFormatPlug`

!!!*EMPTY*!!!

**dataWindow**

`Gaffer::AtomicBox2iPlug`

!!!*EMPTY*!!!

**metadata**

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

**channelNames**

`Gaffer::StringVectorDataPlug`

!!!*EMPTY*!!!

**channelData**

`Gaffer::FloatVectorDataPlug`

!!!*EMPTY*!!!

**png**

`Gaffer::ValuePlug`

Format options specific to PNG files.

**compression**

`Gaffer::StringPlug`

The compression method to use when writing the PNG file.

**compressionLevel**

`Gaffer::IntPlug`

The compression level of the PNG file. This is a value between 0 (no compression) and 9 (most compression).

**requirement**

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

**requirements**

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

**rla**

`Gaffer::ValuePlug`

Format options specific to RLA files.

**dataType**

`Gaffer::StringPlug`

The data type to be written to the RLA file.

### sgi

`Gaffer::ValuePlug`

Format options specific to SGI files.

#### dataType

`Gaffer::StringPlug`

The data type to be written to the SGI file.

### targa

`Gaffer::ValuePlug`

Format options specific to Targa files.

#### compression

`Gaffer::StringPlug`

The compression method to use when writing the Targa file.

### tiff

`Gaffer::ValuePlug`

Format options specific to TIFF files.

#### mode

`Gaffer::IntPlug`

The write mode for the TIFF file - scanline or tiled data.

#### compression

`Gaffer::StringPlug`

The compression method to use when writing the TIFF file.

#### dataType

`Gaffer::StringPlug`

The data type to be written to the TIFF file.

### webp

`Gaffer::ValuePlug`

Format options specific to WebP files.

#### compressionQuality

`Gaffer::IntPlug`

The compression quality for the WebP file to be written. A value between 0 (low quality, high compression) and 100 (high quality, low compression).

## LUT

Applies color transformations provided by OpenColorIO via a LUT file and OCIO FileTransform.

**Plugs:**

### direction

`Gaffer::IntPlug`

The direction to perform the color transformation.

### enabled

`Gaffer::BoolPlug`

Turns the node on and off.

**fileName**

`Gaffer::StringPlug`

The name of the LUT file to be read. Only OpenColorIO supported files will function as expected.

**in**

`GafferImage::ImagePlug`

The input image

**interpolation**

`Gaffer::IntPlug`

The interpolation mode for the color transformation.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## Merge

Composites two or more images together. The following operations are available :

- Add : A + B
- Atop : Ab + B(1-a)
- Divide : A / B
- In : Ab
- Out : A(1-b)
- Mask : Ba
- Matte : Aa + B(1.-a)
- Multiply : AB
- Over : A + B(1-a)
- Subtract : A - B
- Under : A(1-b) + B

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`Gaffer::ArrayPlug`

The input images

**operation**

`Gaffer::IntPlug`

The compositing operation used to merge the image together. See node documentation for more details.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## ObjectToImage

Converts Cortex ImagePrimitives into Gaffer's standard form, so they can processed using Gaffer's image processing nodes. This is a somewhat esoteric node, and is unlikely to have a great number of uses.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**object**

`Gaffer::ObjectPlug`

The Cortex ImagePrimitive to be converted. The most likely source of one of these would be an ObjectReader from the GafferCortex module.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## Offset

Offsets (translates) the image in integer increments. Because the increments may only be whole numbers, no filtering is necessary, and this node has improved performance compared to the equivalent ImageTransform.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**offset**

`Gaffer::V2iPlug`

The amount to offset the image by.

**x**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## OpenColorIO

Applies colour transformations provided by OpenColorIO. Configs are loaded from the configuration specified by the OCIO environment variable.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**inputSpace**

`Gaffer::StringPlug`

The colour space of the input image.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**outputSpace**

`Gaffer::StringPlug`

The colour space of the output image.

## Premultiply

Multiplies selected channels by a specified alpha channel.

**Plugs:**

**alphaChannel**

`Gaffer::StringPlug`

The channel to use as the alpha channel. The selected channel does not have to be *A*, but whichever channel is chosen will act as the alpha for the sake of this node. This channel will never be multiplied by itself - it will remain the same as the input.

**channels**

`GafferImage::ChannelMaskPlug`

The subset of channels to operate on.

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## Reformat

Reformats the image to a new resolution, scaling it to fit the new display window.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**filter**

`GafferImage::FilterPlug`

The pixel filter used when transforming the image. Each filter provides different tradeoffs between sharpness and the danger of aliasing or ringing.

**format**

`GafferImage::AtomicFormatPlug`

The format (resolution and pixel aspect ratio) of the output image.

**in**

`GafferImage::ImagePlug`

The input image

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## RemoveChannels

Deletes channels from an image.

**Plugs:**

**channels**

`GafferImage::ChannelMaskPlug`

The names of the channels to be deleted (or kept if the mode is set to Keep).

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**mode**

`Gaffer::IntPlug`

Defines how the channels listed in the channels plug are treated. Delete mode deletes the listed channels. Keep mode keeps the listed channels, deleting all others.

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## Resample

Utility node used internally within GafferImage, but not intended to be used directly by end users.

**Plugs:**

**boundingMode**

`Gaffer::IntPlug`

The method used when a filter references pixels outside the input data window.

**dataWindow**

`Gaffer::AtomicBox2fPlug`

The output data window. The contents of the input data window will be scaled and offset as necessary to fill this.

**debug**

`Gaffer::IntPlug`

Enables debug output. The HorizontalPass setting outputs an intermediate image filtered just in the horizontal direction - this is an internal optimisation used when filtering with a separable filter. The SinglePass setting forces all filtering to be done in a single pass (as if the filter was non-separable) and can be used for validating the results of the the two-pass (default) approach.

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**filter**

`Gaffer::StringPlug`

The filter used to perform the resampling. The name of any OIIO filter may be specified. The default automatically picks an appropriate high-quality filter based on whether or not the image is being enlarged or reduced.

**filterWidth**

`Gaffer::V2fPlug`

An override for the width of the filter used. This is specified as a number of pixels in the output image. The default value of 0 causes a good default width to be picked based on the filter type.

**x**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**in**

`GafferImage::ImagePlug`

The input image

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## Resize

Resizes the image to a new resolution, scaling the contents to fit the new size.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**filter**

`Gaffer::StringPlug`

The pixel used when transforming the image. Each filter provides different tradeoffs between sharpness and the danger of aliasing or ringing.

**fitMode**

`Gaffer::IntPlug`

Determines how the image is scaled to fit the new resolution. If the aspect ratios of the input and the output images are the same, then this has no effect, otherwise it dictates what method is used to preserve the aspect ratio of the data. Horizontal : The image is scaled so that it fills the full width of the output resolution and aspect ratio is preserved. Vertical : The image is scaled so that it fills the full height of the output resolution and aspect ratio is preserved. Fit : Automatically picks Horizontal or Vertical such that all of the input image is contained within the output image. Padding is applied top and bottom or left and right as necessary. Fill : Automatically picks Horizontal or Vertical such that the full output resolution is covered. The image contents will extend outside the top and bottom or left and right of the display window as necessary. Distort : Distorts the image so that the input display window is fitted exactly to the output display window.

**format**

`GafferImage::FormatPlug`

The new format (resolution and pixel aspect ratio) of the output image.

**displayWindow**

`Gaffer::Box2iPlug`

!!!*EMPTY*!!!

**min**

`Gaffer::V2iPlug`

!!!*EMPTY*!!!

**x**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**max**

`Gaffer::V2iPlug`

!!!*EMPTY*!!!

**x**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**pixelAspect**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**in**

`GafferImage::ImagePlug`

The input image

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## Shuffle

Shuffles data between image channels, for instance by copying R into G or a constant white into A.

**Plugs:**

**channels**

`Gaffer::ValuePlug`

The definition of the shuffling to be performed - an arbitrary number of channel edits can be made by adding Shuffle.ChannelPlugs as children of this plug.

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

## Unpremultiply

Divides selected channels by a specified alpha channel. If the alpha channel on a pixel is 0, then that pixel will remain the same as the input.

**Plugs:**

**alphaChannel**

`Gaffer::StringPlug`

The channel to use as the alpha channel. The selected channel does not have to be *A*, but whichever channel is chosen will act as the alpha for the sake of this node. This channel will never be divided by itself - it will remain the same as the input.

**channels**

`GafferImage::ChannelMaskPlug`

The subset of channels to operate on.

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

# GafferOSL

## OSLImage

Executes OSL shaders to perform image processing. Use the shaders from the OSL/ImageProcessing menu to read values from the input image and then write values back to it.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`GafferImage::ImagePlug`

The input image

**out**

`GafferImage::ImagePlug`

The output image generated by this node.

**shader**

`Gaffer::Plug`

The shader to be executed - connect the output from an OSL network here. A typical shader network to process RGB would look like this : InLayer→ProcessingNodes→OutLayer→OutImage

## OSLObject

Executes OSL shaders to perform object processing. Use the shaders from the OSL/ObjectProcessing menu to read primitive variables from the input object and then write primitive variables back to it.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

**shader**

`Gaffer::Plug`

The shader to be executed - connect the output from an OSL network here. A minimal shader network to process P would look like this : InPoint→ProcessingNodes→OutPoint→OutObject

## OSLShader

Loads OSL shaders for use in supported renderers. Use the ShaderAssignment node to assign shaders to objects in the scene.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**name**

`Gaffer::StringPlug`

The name of the shader being represented. This should be considered read-only. Use the Shader.loadShader() method to load a shader.

**parameters**

`Gaffer::CompoundPlug`

Where the parameters for the shader are represented.

**type**

`Gaffer::StringPlug`

The type of the shader being represented. This should be considered read-only. Use the Shader.loadShader() method to load a shader.

# GafferRenderMan

## InteractiveRenderManRender

Performs interactive renders using a RenderMan renderer which supports interactive edits. In 3delight the following edits are currently supported :

- Adding/removing lights
- Adjusting light parameters
- Moving the camera
- Moving coordinate systems
- Changing shader assignments
- Editing shader networks
- Editing shader parameters

**Plugs:**

**in**

`GafferScene::ScenePlug`

The scene to be rendered.

**out**

`GafferScene::ScenePlug`

A direct pass-through of the input scene.

**bound**

`Gaffer::AtomicBox3fPlug`

!!!*EMPTY*!!!

**transform**

`Gaffer::M44fPlug`

!!!*EMPTY*!!!

**attributes**

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

**object**

`Gaffer::ObjectPlug`

!!!*EMPTY*!!!

**childNames**

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

**globals**

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

**setNames**

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

#### set

`GafferScene::PathMatcherDataPlug`

!!!*EMPTY*!!!

### state

`Gaffer::IntPlug`

The interactive state.

### updateAttributes

`Gaffer::BoolPlug`

When on, changes to attribute (and shaders) are reflected in the interactive render. When working with complex scenes, it may be worth turning this off to gain increased performance when only editing lights.

### updateCameras

`Gaffer::BoolPlug`

When on, changes to the camera are reflected in the interactive render.

### updateCoordinateSystems

`Gaffer::BoolPlug`

When on, changes to coordinate systems are reflected in the interactive render.

### updateLights

`Gaffer::BoolPlug`

When on, changes to lights are reflected in the interactive render.

## RenderManAttributes

Applies RenderMan specific attributes to the scene.

**Plugs:**

### attributes

`Gaffer::CompoundDataPlug`

The attributes to be applied - arbitrary numbers of user defined attributes may be added as children of this plug via the user interface, or using the CompoundDataPlug API via python.

#### cameraVisibility.value

`Gaffer::BoolPlug`

Whether or not objects are visible to the camera. An object can be invisible to the camera but still appear in reflections and shadows etc. To make an object totally invisible, prefer the visibility setting provided by the StandardAttributes node.

#### cameraHitMode.value

`Gaffer::StringPlug`

Specifies if shading is performed when the object is hit by a camera ray, or if the primitive colour is used as an approximation instead.

#### transmissionVisibility.value

`Gaffer::BoolPlug`

Whether or not objects are visible to transmission rays. Objects that are visible to transmission rays will cast shadows, and those that aren’t won’t.

**transmissionHitMode.value**

`Gaffer::StringPlug`

Specifies if shading is performed when the object is hit by a tranmission ray, or if the primitive opacity is used as an approximation instead.

**diffuseVisibility.value**

`Gaffer::BoolPlug`

Whether or not objects are visible to diffuse rays - typically this means whether or not they cast occlusion and bounce light.

**diffuseHitMode.value**

`Gaffer::StringPlug`

Specifies if shading is performed when the object is hit by a diffuse ray, or if the primitive colour is used as an approximation instead.

**specularVisibility.value**

`Gaffer::BoolPlug`

Whether or not objects are visible to specular rays - typically this means whether or not they are visible in reflections and refractions.

**specularHitMode.value**

`Gaffer::StringPlug`

Specifies if shading is performed when the object is hit by a specular ray, or if the primitive colour is used as an approximation instead.

**photonVisibility.value**

`Gaffer::BoolPlug`

Whether or not objects are visible to photons.

**photonHitMode.value**

`Gaffer::StringPlug`

Specifies if shading is performed when the object is hit by a photon, or if the primitive colour is used as an approximation instead.

**shadingRate.value**

`Gaffer::FloatPlug`

Specifies how finely objects are diced before they are shaded. Smaller values give higher quality but slower rendering.

**relativeShadingRate.value**

`Gaffer::FloatPlug`

Specifies a multiplier on the shading rate. Note that if shading rate is specified at multiple locations above an object in the hierarchy, they are multiplied together to arrive at the final shading rate.

**matte.value**

`Gaffer::BoolPlug`

Matte objects don’t appear in the render, but cut holes in the alpha of all objects behind them.

**displacementBound.value**

`Gaffer::FloatPlug`

Specifies the maximum amount the displacement shader will displace by.

**maxDiffuseDepth.value**

`Gaffer::IntPlug`

The maximum depth for diffuse ray bounces.

**maxSpecularDepth.value**

`Gaffer::IntPlug`

The maximum depth for specular (reflection) ray bounces.

**traceDisplacements.value**

`Gaffer::BoolPlug`

Whether or not displacements are taken into account when raytracing. This can be fairly expensive.

**traceBias.value**

`Gaffer::FloatPlug`

This bias value affects rays. It is an offset applied to the ray origin, moving it slightly away from the surface launch point in the ray direction. This offset can prevent blotchy artifacts resulting from the ray immediately finding an intersection with the surface it just left. Usually, 0.01 is the default scene value.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**global**

`Gaffer::BoolPlug`

Causes the attributes to be applied to the scene globals instead of the individual locations defined by the filter.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

## RenderManLight

Loads a RenderMan light shader and uses it to output a scene with a single light.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

**name**

`Gaffer::StringPlug`

The name of the object in the output scene.

**out**

`GafferScene::ScenePlug`

The output scene.

**parameters**

`Gaffer::Plug`

The parameters of the light shader - these will vary based on the light type.

**sets**

`Gaffer::StringPlug`

A list of sets to include the object in. The names should be separated by spaces.

**transform**

`Gaffer::TransformPlug`

The transform applied to the object.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## RenderManOptions

Sets global scene options applicable to RenderMan renderers. Use the StandardOptions node to set global options applicable to all renderers.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**options**

`Gaffer::CompoundDataPlug`

The options to be applied - arbitrary numbers of user defined options may be added as children of this plug via the user interface, or using the CompoundDataPlug API via python.

**pixelSamples.value**

`Gaffer::V2iPlug`

The number of primary samples to divide each pixel into in the X and Y directions. For example, 3x3 gives a total of 9 samples per pixel. This is the primary quality control for geometric antialiasing and motion blur.

**hider.value**

`Gaffer::StringPlug`

The "Hidden" hider means the classic REYES algorithm is used, and the "Raytrace" hider means a more modern raytraced algorithm is used.

**hiderDepthFilter.value**

`Gaffer::StringPlug`

The filter used to compute a single depth value per pixel from the depths in each pixel sample.

**hiderJitter.value**

`Gaffer::BoolPlug`

Whether or not each pixel sample is jittered about the centre of its subpixel position, or if they're aligned in a regular grid. If in doubt, leave this on.

**hiderSampleMotion.value**

`Gaffer::BoolPlug`

May be turned off to disable the sampling of motion blur, but keep motion vectors available for use in shaders. This is useful for rendering a motion vector pass to allow 2D motion blur to be applied as a post process. If you simply wish to turn off motion blur entirely, then use the motion blur settings in the StandardOptions node.

**hiderExtremeMotionDOF.value**

`Gaffer::BoolPlug`

An alternative sampling algorithm which is more expensive, but gives higher quality results when objects are both moving quickly and are out of focus.

**hiderProgressive.value**

`Gaffer::BoolPlug`

Renders at progressively increasing levels of quality, to give quick low quality feedback at the start of an interactive render. Only applies when the raytrace hider is used.

**statisticsLevel.value**

`Gaffer::IntPlug`

Determines the verbosity of statistics output.

**statisticsFileName.value**

`Gaffer::StringPlug`

The name of a file where the statistics will be written.

**statisticsProgress.value**

`Gaffer::BoolPlug`

Turning this on causes a render progress percentage to be printed out continuously during rendering.

**shaderSearchPath.value**

`Gaffer::StringPlug`

The filesystem paths where shaders are searched for. Paths should be separated by *:*.

**textureSearchPath.value**

`Gaffer::StringPlug`

The filesystem paths where shaders are located. Paths should be separated by *:*.

#### displaySearchPath.value

`Gaffer::StringPlug`

The filesystem paths where display driver plugins are located. These will be used when searching for drivers specified using the Outputs node. Paths should be separated by :.

#### archiveSearchPath.value

`Gaffer::StringPlug`

The filesystem paths where RIB archives are located. These will be used when searching for archives specified using the ExternalProcedural node. Paths should be separated by :.

#### proceduralSearchPath.value

`Gaffer::StringPlug`

The filesystem paths where DSO procedurals are located. These will be used when searching for procedurals specified using the ExternalProcedural node. Paths should be separated by :.

### out

`GafferScene::ScenePlug`

The processed output scene.

## RenderManRender

Performs offline batch rendering using a RenderMan renderer. This is done in two phases - first a RIB file is generated and then the renderer is invoked to render it in a separate process. Note though that the RIB file is lightweight, and contains a single procedural which will invoke Gaffer to generate the scene on demand at runtime. The RIB therefore requires very little disk space.

**Plugs:**

### command

`Gaffer::StringPlug`

The system command used to invoke the renderer - this can be edited to add any custom flags that are necessary, or to use a different renderer. The rib filename is automatically appended to the command before it is invoked.

### dispatcher

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

#### batchSize

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

#### local

`Gaffer::Plug`

Settings used by the local dispatcher.

### in

`GafferScene::ScenePlug`

The scene to be rendered.

### mode

`Gaffer::StringPlug`

When in "Render" mode, a RIB file is generated and then renderered by running the renderer on it. In "Generate RIB only" mode, only the RIB is generated, and a subsequent node could be used to post-process or launch the render in another way - a SystemCommand node may be useful for this.

**out**

`GafferScene::ScenePlug`

A pass-through of the input scene.

**bound**

`Gaffer::AtomicBox3fPlug`

!!!*EMPTY*!!!

**transform**

`Gaffer::M44fPlug`

!!!*EMPTY*!!!

**attributes**

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

**object**

`Gaffer::ObjectPlug`

!!!*EMPTY*!!!

**childNames**

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

**globals**

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

**setNames**

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

**set**

`GafferScene::PathMatcherDataPlug`

!!!*EMPTY*!!!

**requirement**

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

**requirements**

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

**ribFileName**

`Gaffer::StringPlug`

The name of the RIB file to be generated.

## RenderManShader

Loads shaders for use in RenderMan renderers. Use the ShaderAssignment node to assign shaders to objects in the scene.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**name**

`Gaffer::StringPlug`

The name of the shader being represented. This should be considered read-only. Use the Shader.loadShader() method to load a shader.

**out**

`Gaffer::Plug`

The output from the shader.

**parameters**

`Gaffer::CompoundPlug`

Where the parameters for the shader are represented.

**type**

`Gaffer::StringPlug`

The type of the shader being represented. This should be considered read-only. Use the Shader.loadShader() method to load a shader.

# GafferScene

## AimConstraint

Transforms objects so that they are aimed at a specified target.

**Plugs:**

**aim**

`Gaffer::V3fPlug`

The aim vector, specified in object space. The object will be transformed so that this vector points at the target.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

**target**

`Gaffer::StringPlug`

The scene location to which the objects are constrained. The world space transform of this location forms the basis of the constraint target, but is modified by the targetMode and targetOffset values before the constraint is applied.

**targetMode**

`Gaffer::IntPlug`

The precise location of the target transform - this can be derived from the origin or bounding box of the target location.

**targetOffset**

`Gaffer::V3fPlug`

An offset applied to the target transform before the constraint is applied. The offset is measured in the object space of the target location.

**up**

`Gaffer::V3fPlug`

The up vector, specified in object space. The object will be transformed so that this vector points up in world space, as far as is possible.

## AlembicSource

Loads Alembic caches. Please note that Gaffer requires a bounding box to be computable for every location in the scene. Alembic files can store such bounding boxes, but in practice they often don't. In this case Gaffer must perform a full scene traversal to compute the appropriate bounding box. It is recommended that if performance is a priority, bounding boxes should be stored explicitly in the Alembic cache, or the Cortex SceneCache (.scc) format should be used instead, since it always stores accurate bounds.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

**fileName**

`Gaffer::StringPlug`

The path to the .abc file to load. Both older HDF5 and newer Ogawa caches are supported.

**out**

`GafferScene::ScenePlug`

The output scene.

**refreshCount**

`Gaffer::IntPlug`

Can be incremented to invalidate Gaffer's memory cache and force a reload if the .abc file is changed on disk.

## AttributeVisualiser

Visualises attribute values by applying a constant shader to display them as a colour.

**Plugs:**

**attributeName**

`Gaffer::StringPlug`

The name of the attribute to be visualised. The value of the attribute will be converted to a colour using the chosen mode and then assigned using a constant shader.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**max**

`Gaffer::FloatPlug`

Used in the Color and False Color modes to define the value which is mapped to white or the right end of the spline respectively.

**min**

`Gaffer::FloatPlug`

Used in the Color and False Color modes to define the value which is mapped to black or the left end of the spline respectively.

**mode**

`Gaffer::IntPlug`

The method used to turn the attribute value into a colour for visualisation. - Color : This only works for attributes which already contain a colour or numeric value. The value is converted directly to a colour, using the min and max values to perform a remapping. - FalseColor : This only works for numeric attributes. Values between min and max are used to look up a colour in the ramp below. - Random : This works for any attribute type - a random colour is chosen for each unique attribute value. - Shader Node Color : This only works when visualising a shader attribute. It uses the node colour for the shader node which is assigned.

**out**

`GafferScene::ScenePlug`

The processed output scene.

**ramp**

`Gaffer::SplinefColor3fPlug`

Provides the colour mapping for the False Color mode. Values between min and max are remapped using the colours from the ramp (left to right).

**basis**

`Gaffer::ValuePlug`

!!!*EMPTY*!!!

**matrix**

`Gaffer::M44fPlug`

!!!*EMPTY*!!!

**step**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**endPointMultiplicity**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**p0**

`Gaffer::ValuePlug`

!!!*EMPTY*!!!

**x**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::Color3fPlug`

!!!*EMPTY*!!!

**p1**

`Gaffer::ValuePlug`

!!!*EMPTY*!!!

**x**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::Color3fPlug`

!!!*EMPTY*!!!

**shaderName**

`Gaffer::StringPlug`

The name of the shader used to perform the visualisation. The default value is for an OpenGL shader which will be used in the viewport. It's possible to perform a visualisation for other renderers by entering a different shader name here.

**shaderParameter**

`Gaffer::StringPlug`

The name of the shader parameter used to perform the visualisation. The default value is for an OpenGL shader which will be used in the viewport.

**shaderType**

`Gaffer::StringPlug`

The type of shader used to perform the visualisation. The default value is for an OpenGL shader which will be used in the viewport. It's possible to perform a visualisation for other renderers by entering a different shader type here.

## Camera

Produces scenes containing a camera. To choose which camera is used for rendering, use a StandardOptions node.

**Plugs:**

**clippingPlanes**

`Gaffer::V2fPlug`

The near and far clipping planes.

**x**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

**fieldOfView**

`Gaffer::FloatPlug`

The field of view, specified in degrees, and interpreted as defined in the RenderMan specification. This is only relevant for perspective cameras.

**name**

`Gaffer::StringPlug`

The name of the object in the output scene.

**out**

`GafferScene::ScenePlug`

The output scene.

**projection**

`Gaffer::StringPlug`

The basic camera type.

**sets**

`Gaffer::StringPlug`

A list of sets to include the object in. The names should be separated by spaces.

**transform**

`Gaffer::TransformPlug`

The transform applied to the object.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## ClippingPlane

Creates an arbitrary clipping plane. This is like the near and far clipping planes provided by the Camera node, but can be positioned arbitrarily in space. All geometry on the positive Z side of the plane is clipped away.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

**name**

`Gaffer::StringPlug`

The name of the clipping plane to be created.

**out**

`GafferScene::ScenePlug`

The output scene.

**sets**

`Gaffer::StringPlug`

A list of sets to include the object in. The names should be separated by spaces.

**transform**

`Gaffer::TransformPlug`

The transform applied to the object.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## CoordinateSystem

Produces scenes containing a coordinate system. Coordinate systems have two main uses :

- To visualise the transform at a particular location. In this respect they're similar to locators or nulls in other packages.
- To define a named coordinate system to be used in shaders at render time. This is useful for defining projections or procedural solid textures. The full path to the location of the coordinate system should be used to refer to it within shaders.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

**name**

`Gaffer::StringPlug`

The name of the object in the output scene.

**out**

`GafferScene::ScenePlug`

The output scene.

**sets**

`Gaffer::StringPlug`

A list of sets to include the object in. The names should be separated by spaces.

**transform**

`Gaffer::TransformPlug`

The transform applied to the object.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## Cube

Produces scenes containing a cube.

**Plugs:**

**dimensions**

`Gaffer::V3fPlug`

The size of the cube.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

**name**

`Gaffer::StringPlug`

The name of the object in the output scene.

**out**

`GafferScene::ScenePlug`

The output scene.

**sets**

`Gaffer::StringPlug`

A list of sets to include the object in. The names should be separated by spaces.

**transform**

`Gaffer::TransformPlug`

The transform applied to the object.

#### translate

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

#### rotate

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

#### scale

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

#### pivot

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## CustomAttributes

Applies arbitrary user-defined attributes to locations in the scene. Note that for most common cases the StandardAttributes, OpenGLAttributes, RenderManAttributes, and ArnoldAttributes nodes should be used in preference - they provide predefined sets of attributes with customised user interfaces. The CustomAttributes node is of most use when needing to set a custom attribute not supported by the specialised nodes.

**Plugs:**

### attributes

`Gaffer::CompoundDataPlug`

The attributes to be applied - arbitrary numbers of user defined attributes may be added as children of this plug via the user interface, or using the CompoundDataPlug API via python.

### enabled

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

### filter

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

### global

`Gaffer::BoolPlug`

Causes the attributes to be applied to the scene globals instead of the individual locations defined by the filter.

### in

`GafferScene::ScenePlug`

The input scene

### out

`GafferScene::ScenePlug`

The processed output scene.

## CustomOptions

Applies arbitrary user-defined options to the root of the scene. Note that for most common cases the StandardOptions, OpenGLOptions, RenderManOptions, and ArnoldOptions nodes should be used in preference - they provide predefined sets of options with customised user interfaces. The CustomOptions node is of most use when needing to set a custom option not supported by the specialised nodes.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**options**

`Gaffer::CompoundDataPlug`

The options to be applied - arbitrary numbers of user defined options may be added as children of this plug via the user interface, or using the CompoundDataPlug API via python.

**out**

`GafferScene::ScenePlug`

The processed output scene.

## DeleteAttributes

Deletes attributes from locations within the scene. Those locations will then inherit the attribute values from ancestor locations instead, or will fall back to using the default attribute value.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**invertNames**

`Gaffer::BoolPlug`

When on, matching names are kept, and non-matching names are removed.

**names**

`Gaffer::StringPlug`

The names of attributes to be removed. Names should be separated by spaces and can use Gaffer’s standard wildcards.

**out**

`GafferScene::ScenePlug`

The processed output scene.

## DeleteOptions

A node which removes options from the globals.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**invertNames**

`Gaffer::BoolPlug`

When on, matching names are kept, and non-matching names are removed.

**names**

`Gaffer::StringPlug`

The names of options to be removed. Names should be separated by spaces and can use Gaffer's standard wildcards.

**out**

`GafferScene::ScenePlug`

The processed output scene.

## DeleteOutputs

A node which removes outputs from the globals.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**invertNames**

`Gaffer::BoolPlug`

When on, matching names are kept, and non-matching names are removed.

**names**

`Gaffer::StringPlug`

The names of outputs to be removed. Names should be separated by spaces and can use Gaffer's standard wildcards.

**out**

`GafferScene::ScenePlug`

The processed output scene.

## DeletePrimitiveVariables

Deletes primitive variables from objects. The primitive variables to be deleted are chosen based on name.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**invertNames**

`Gaffer::BoolPlug`

When on, the primitive variables matched by names are kept, and the non-matching primitive variables are deleted.

**names**

`Gaffer::StringPlug`

The names of the primitive variables to be deleted. Names should be specified by spaces, and Gaffer's standard wildcard characters may be used.

**out**

`GafferScene::ScenePlug`

The processed output scene.

## DeleteSets

A node which removes object sets.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**invertNames**

`Gaffer::BoolPlug`

When on, matching names are kept, and non-matching names are removed.

**names**

`Gaffer::StringPlug`

Space separated list of set names to be removed.

**out**

`GafferScene::ScenePlug`

The processed output scene.

## Displays

Defines the image outputs to be created by the renderer. Arbitrary outputs can be defined within the UI and also via the `Outputs::addOutput()` API. Commonly used outputs may also be predefined at startup via a config file - see $GAFFER_ROOT/startup/gui/outputs.py for an example.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

**outputs**

`Gaffer::ValuePlug`

The outputs defined by this node.

## Duplicate

Duplicates a part of the scene. The duplicates are parented alongside the original, and have a transform applied to them.

**Plugs:**

**copies**

`Gaffer::IntPlug`

The number of copies to be made.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**name**

`Gaffer::StringPlug`

The name given to the copies. If this is left empty, the name from the target will be used instead. The names will have a numeric suffix applied to distinguish between the different copies, unless only a single copy is being made. Even in the case of a single copy, a suffix will be applied if necessary to keep the names unique.

**out**

`GafferScene::ScenePlug`

The processed output scene.

**parent**

`Gaffer::StringPlug`

For internal use only.

**target**

`Gaffer::StringPlug`

The part of the scene to be duplicated.

**transform**

`Gaffer::TransformPlug`

The transform to be applied to the copies. The transform is applied iteratively, so the second copy is transformed twice, the third copy is transformed three times and so on.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## ExternalProcedural

References external geometry procedurals and archives.

**Plugs:**

**bound**

`Gaffer::Box3fPlug`

The bounding box of the external procedural or archive.

**min**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**max**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

**fileName**

`Gaffer::StringPlug`

The path to the external procedural or archive.

**name**

`Gaffer::StringPlug`

The name of the object in the output scene.

**out**

`GafferScene::ScenePlug`

The output scene.

**parameters**

`Gaffer::CompoundDataPlug`

An arbitrary set of parameters to be passed to the external procedural.

**sets**

`Gaffer::StringPlug`

A list of sets to include the object in. The names should be separated by spaces.

**transform**

`Gaffer::TransformPlug`

The transform applied to the object.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## FilterSwitch

Chooses between multiple input filters, passing the chosen filter through to the output.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the filter. When it is off, the filter does not match any locations.

**in**

`Gaffer::ArrayPlug`

The input filters

**index**

`Gaffer::IntPlug`

The index of the input which is passed through. A value of 0 chooses the first input, 1 the second and so on. Values larger than the number of available inputs wrap back around to the beginning.

**out**

`Gaffer::IntPlug`

The result of the filter. This should be connected into the "filter" plug of a FilteredSceneProcessor.

## FreezeTransform

Resets the transforms at the specified scene locations, baking the old transforms into the vertices of any child objects so that they remain the same in world space. Essentially this turns transforms in the hierarchy into rigid deformations of the objects.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

## Grid

" A grid. This is used to draw the grid in the viewer, but is also included as a node in case it might be useful, perhaps for placing a grid in renders done using the OpenGLRender node.

**Plugs:**

**borderColor**

`Gaffer::Color3fPlug`

The colour of the lines forming the border of the grid.

**borderPixelWidth**

`Gaffer::FloatPlug`

The width of the lines forming the border of the grid. This width applies only to the OpenGL representation of the grid.

### centerColor

`Gaffer::Color3fPlug`

The colour of the two lines forming the central cross of the grid.

### centerPixelWidth

`Gaffer::FloatPlug`

The width of the two lines forming the central cross of the grid. This width applies only to the OpenGL representation of the grid.

### dimensions

`Gaffer::V2fPlug`

The size of the grid in the x and y axes. Use the transform to rotate the grid into a different plane.

#### x

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

#### y

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

### enabled

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

### gridColor

`Gaffer::Color3fPlug`

The colour of the lines forming the main part of the grid.

### gridPixelWidth

`Gaffer::FloatPlug`

The width of the lines forming the main part of the grid. This width applies only to the OpenGL representation of the grid.

### name

`Gaffer::StringPlug`

The name of the grid.

### out

`GafferScene::ScenePlug`

The output scene.

### spacing

`Gaffer::FloatPlug`

The size of the space between adjacent lines in the grid.

### transform

`Gaffer::TransformPlug`

The transform applied to the grid.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## Group

Groups together several input scenes under a new parent. If the input scenes contain locations with identical names, they are automatically renamed to make them unique in the output scene.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`Gaffer::ArrayPlug`

The input scenes

**name**

`Gaffer::StringPlug`

The name of the group to be created. All the input scenes will be parented under this group.

**out**

`GafferScene::ScenePlug`

The processed output scene.

**transform**

`Gaffer::TransformPlug`

The transform for the group itself. This will be inherited by the objects parented under it.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## Instancer

Instances an input scene onto the vertices of a target object, making one copy per vertex. Note that in Gaffer, the instances are not limited to being a single object, and each instance need not be identical. Instances can instead include entire hierarchies and can be varied from point to point, and individual instances may be modified downstream without affecting the others. Gaffer ensure's that where instances happen to be identical, they share memory, and performs automatic instancing at the object level when exporting to the renderer (this occurs for all nodes, not just the Instancer).

Per-instance variation can be achieved using the A common use case is to use this to randomise the index on a SceneSwitch node, to choose randomly between several instances, but it can be used to drive *any* property of the upstream graph.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**instance**

**name**

`Gaffer::StringPlug`

The name of the location the instances will be generated below. This will be parented directly under the parent location.

**out**

`GafferScene::ScenePlug`

The processed output scene.

**parent**

`Gaffer::StringPlug`

The object on which to make the instances. This must have a "P" primitive variable specifying the location of each instance.

## Isolate

Isolates objects by removing paths not matching a filter from the scene.

**Plugs:**

**adjustBounds**

`Gaffer::BoolPlug`

By default, the bounding boxes of ancestor locations are automatically updated when children are removed. This can be turned off if necessary to get improved performance - in this case the bounding boxes will still wholly contain the contents at each location, but may be bigger than necessary.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**from**

`Gaffer::StringPlug`

The ancestor to isolate the objects from. Only locations below this will be removed.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

## MapOffset

Adds an offset to object texture coordinates. Provides a convenient way of looking at specific texture UDIMs.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**offset**

`Gaffer::V2fPlug`

An offset added to the texture coordinates. Note that moving the texture coordinates in the positive direction will move the texture in the negative direction.

**x**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**out**

`GafferScene::ScenePlug`

The processed output scene.

**sName**

`Gaffer::StringPlug`

The name of the primitive variable holding the s coordinate.

**tName**

`Gaffer::StringPlug`

The name of the primitive variable holding the t coordinate.

**udim**

`Gaffer::IntPlug`

A specific UDIM to offset the texture coordinates to. The UDIM is converted to an offset which is added to the offset above.

## MapProjection

Applies texture coordinates to meshes via a camera projection. In Gaffer, texture coordinates (commonly referred to as UVs) are represented as primitive variables named "s" and "t".

**Plugs:**

**camera**

`Gaffer::StringPlug`

The location of the camera to use for the projection.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

**sName**

`Gaffer::StringPlug`

The name of the primitive variable to store the s coordinate of the projected texture coordinates. This may be changed in order to store multiple sets of texture coordinates on a single mesh.

**tName**

`Gaffer::StringPlug`

The name of the primitive variable to store the t coordinate of the projected texture coordinates. This may be changed in order to store multiple sets of texture coordinates on a single mesh.

## MeshType

Changes between polygon and subdivision representations for mesh objects, and optionally recalculates vertex normals for polygon meshes.

Note that currently the Gaffer viewport does not display subdivision meshes with smoothing, so the results of using this node will not be seen until a render is performed.

**Plugs:**

**calculatePolygonNormals**

`Gaffer::BoolPlug`

Causes new vertex normals to be calculated for polygon meshes. Has no effect for subdivision surfaces, since those are naturally smooth and do not require surface normals. Vertex normals are represented as primitive variables named "N".

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**meshType**

`Gaffer::StringPlug`

The interpolation type to apply to the mesh.

**out**

`GafferScene::ScenePlug`

The processed output scene.

**overwriteExistingNormals**

`Gaffer::BoolPlug`

By default, vertex normals will only be calculated for polygon meshes which don’t already have them. Turning this on will force new normals to be calculated even for meshes which had them already.

## ObjectToScene

Converts objects to be used with the nodes in the GafferScene module. Typically these objects would come from a GafferCortex OpHolder node or ObjectReader node.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

**name**

`Gaffer::StringPlug`

The name of the object in the output scene.

**object**

`Gaffer::ObjectPlug`

The object to be placed in the output scene.

**out**

`GafferScene::ScenePlug`

The output scene.

**sets**

`Gaffer::StringPlug`

A list of sets to include the object in. The names should be separated by spaces.

**transform**

`Gaffer::TransformPlug`

The transform applied to the object.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## OpenGLAttributes

Applies attributes to modify the appearance of objects in the viewport and in renders done by the OpenGLRender node.

**Plugs:**

**attributes**

`Gaffer::CompoundDataPlug`

The attributes to be applied - arbitrary numbers of user defined attributes may be added as children of this plug via the user interface, or using the CompoundDataPlug API via python.

**primitiveSolid.value**

`Gaffer::BoolPlug`

Whether or not the object is rendered solid, in which case the assigned GLSL shader will be used to perform the shading.

**primitiveWireframe.value**

`Gaffer::BoolPlug`

Whether or not the object is rendered as a wireframe. Use the primitiveWireframeColor and primitiveWireframeWidth plugs for finer control of the wireframe appearance.

**primitiveWireframeColor.value**

`Gaffer::Color4fPlug`

The colour to use for the wireframe rendering. Only meaningful if wireframe rendering is turned on.

**primitiveWireframeWidth.value**

`Gaffer::FloatPlug`

The width in pixels of the wireframe rendering. Only meaningful if wireframe rendering is turned on.

**primitiveOutline.value**

`Gaffer::BoolPlug`

Whether or not an outline is drawn around the object. Use the primitiveOutlineColor and primitiveOutlineWidth plugs for finer control of the outline.

**primitiveOutlineColor.value**

`Gaffer::Color4fPlug`

The colour to use for the outline. Only meaningful if outline rendering is turned on.

**primitiveOutlineWidth.value**

`Gaffer::FloatPlug`

The width in pixels of the outline. Only meaningful if outline rendering is turned on.

**primitivePoint.value**

`Gaffer::BoolPlug`

Whether or not the individual points (vertices) of the object are drawn. Use the primitivePointColor and primitivePointWidth plugs for finer control of the point rendering.

**primitivePointColor.value**

`Gaffer::Color4fPlug`

The colour to use for the point rendering. Only meaningful if point rendering is turned on.

**primitivePointWidth.value**

`Gaffer::FloatPlug`

The width in pixels of the points. Only meaningful if point rendering is turned on.

**primitiveBound.value**

`Gaffer::BoolPlug`

Whether or not the bounding box of the object is drawn. This is in addition to any drawing of unexpanded bounding boxes that the viewer performs. Use the primitiveBoundColor plug to change the colour of the bounding box.

**primitiveBoundColor.value**

`Gaffer::Color4fPlug`

The colour to use for the bounding box rendering. Only meaningful if bounding box rendering is turned on.

**pointsPrimitiveUseGLPoints.value**

`Gaffer::StringPlug`

Points primitives have a render type (set by the PointsType node) which allows them to be rendered as particles, disks, spheres etc. This attribute overrides that type for OpenGL only, allowing a much faster rendering as raw OpenGL points.

**pointsPrimitiveGLPointWidth.value**

`Gaffer::FloatPlug`

The width in pixels of the GL points rendered when the pointsPrimitiveUseGLPoints plug has overridden the point type.

**curvesPrimitiveUseGLLines.value**

`Gaffer::BoolPlug`

Curves primitives are typically rendered as ribbons and as such have an associated width in object space. This attribute overrides that for OpenGL only, allowing a much faster rendering as raw OpenGL lines.

**curvesPrimitiveGLLineWidth.value**

`Gaffer::FloatPlug`

The width in pixels of the GL lines rendered when the curvesPrimitiveUseGLLines plug has overridden the drawing to use lines.

**curvesPrimitiveIgnoreBasis.value**

`Gaffer::BoolPlug`

Turns off interpolation for cubic curves, just rendering straight lines between the vertices instead.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**global**

`Gaffer::BoolPlug`

Causes the attributes to be applied to the scene globals instead of the individual locations defined by the filter.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

## OpenGLRender

Renders scenes using the same OpenGL engine as is used in the viewer. Use the OpenGLShader and OpenGLAttributes nodes to specify the appearance of objects within the render.

**Plugs:**

**dispatcher**

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

**batchSize**

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

**local**

`Gaffer::Plug`

Settings used by the local dispatcher.

**in**

`GafferScene::ScenePlug`

The scene to be rendered.

**out**

`GafferScene::ScenePlug`

A pass-through of the input scene.

**bound**

`Gaffer::AtomicBox3fPlug`

!!!*EMPTY*!!!

**transform**

`Gaffer::M44fPlug`

!!!*EMPTY*!!!

**attributes**

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

**object**

`Gaffer::ObjectPlug`

!!!*EMPTY*!!!

**childNames**

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

**globals**

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

**setNames**

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

**set**

`GafferScene::PathMatcherDataPlug`

!!!*EMPTY*!!!

**requirement**

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

**requirements**

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

## OpenGLShader

Loads GLSL shaders for use in the viewer and the OpenGLRender node. GLSL shaders are loaded from *.frag and *.vert files in directories specified by the IECOREGL_SHADER_PATH environment variable.

Use the ShaderAssignment node to assign shaders to objects in the scene.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**name**

`Gaffer::StringPlug`

The name of the shader being represented. This should be considered read-only. Use the Shader.loadShader() method to load a shader.

**out**

`Gaffer::Plug`

The output from the shader.

**parameters**

`Gaffer::CompoundPlug`

Where the parameters for the shader are represented.

**type**

`Gaffer::StringPlug`

The type of the shader being represented. This should be considered read-only. Use the Shader.loadShader() method to load a shader.

## Parameters

Modifies the parameters of lights, cameras and procedurals. Existing parameters can be tweaked and new parameters be added.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

**parameters**

`Gaffer::CompoundDataPlug`

The parameters to be added - any number of arbitrary parameters may be specified here using either the user interface or the CompoundDataPlug API.

## Parent

Parents one scene hierarchy into another.

**Plugs:**

**child**

`GafferScene::ScenePlug`

The child hierarchy to be parented.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

**parent**

`Gaffer::StringPlug`

The location which the child is parented under.

## ParentConstraint

Constrains objects from one part of the scene hierarchy as if they were children of another part of the hierarchy.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

**relativeTransform**

`Gaffer::TransformPlug`

Transforms the constrained object relative to the target location.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**target**

`Gaffer::StringPlug`

The scene location to which the objects are constrained. The world space transform of this location forms the basis of the constraint target, but is modified by the targetMode and targetOffset values before the constraint is applied.

**targetMode**

`Gaffer::IntPlug`

The precise location of the target transform - this can be derived from the origin or bounding box of the target location.

**targetOffset**

`Gaffer::V3fPlug`

An offset applied to the target transform before the constraint is applied. The offset is measured in the object space of the target location.

## PathFilter

Chooses locations by matching them against a list of paths.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the filter. When it is off, the filter does not match any locations.

**out**

`Gaffer::IntPlug`

The result of the filter. This should be connected into the "filter" plug of a FilteredSceneProcessor.

**paths**

`Gaffer::StringVectorDataPlug`

The list of paths to the locations to be matched by the filter. A path is formed by a sequence of names separated by /, and specifies the hierarchical position of a location within the scene. Paths may use Gaffer's standard wildcard characters to match multiple locations. The *wildcard matches any sequence of characters within an individual name, but never matches across names separated by a /. - /robot/*Arm matches /robot/leftArm, /robot/rightArm and /robot/Arm. But does not match /robot/limbs/leftArm or /robot/arm. The "…" wildcard matches any sequence of names, and can be used to match locations no matter where they are parented in the hierarchy. - /…/house matches /house, /street/house and /city/street/house.

## Plane

Produces scenes containing a plane.

**Plugs:**

**dimensions**

`Gaffer::V2fPlug`

The size of the plane in the X and Y directions.

**x**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::FloatPlug`

!!!*EMPTY*!!!

**divisions**

`Gaffer::V2iPlug`

The number of subdivisions of the plane in the X and Y directions.

**x**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**y**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

**name**

`Gaffer::StringPlug`

The name of the object in the output scene.

**out**

`GafferScene::ScenePlug`

The output scene.

**sets**

`Gaffer::StringPlug`

A list of sets to include the object in. The names should be separated by spaces.

**transform**

`Gaffer::TransformPlug`

The transform applied to the object.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## PointConstraint

Translates objects so that they are constrained to the world space position of the target. Leaves the scale and orientation of the object untouched.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**offset**

`Gaffer::V3fPlug`

A world space translation offset applied on top of the target position.

**out**

`GafferScene::ScenePlug`

The processed output scene.

**target**

`Gaffer::StringPlug`

The scene location to which the objects are constrained. The world space transform of this location forms the basis of the constraint target, but is modified by the targetMode and targetOffset values before the constraint is applied.

**targetMode**

`Gaffer::IntPlug`

The precise location of the target transform - this can be derived from the origin or bounding box of the target location.

**targetOffset**

`Gaffer::V3fPlug`

An offset applied to the target transform before the constraint is applied. The offset is measured in the object space of the target location.

**xEnabled**

`Gaffer::BoolPlug`

Enables the constraint in the world space x axis.

**yEnabled**

`Gaffer::BoolPlug`

Enables the constraint in the world space y axis.

**zEnabled**

`Gaffer::BoolPlug`

Enables the constraint in the world space z axis.

## PointsType

Changes the render type for PointsPrimitive objects. Depending on the renderer, points may be rendered as particles, spheres, disks, patches or blobbies.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

**type**

`Gaffer::StringPlug`

The render type to assign.

## PrimitiveVariables

Adds arbitrary primitive variables to objects. Currently only primitive variables with constant interpolation are supported - see the OSLObject node for a means of creating variables with vertex interpolation.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

**primitiveVariables**

`Gaffer::CompoundDataPlug`

The primitive variables to be applied - arbitrary numbers of user defined primitive variables may be added as children of this plug via the user interface, or using the CompoundDataPlug API via python.

## Prune

A node for removing whole branches from the scene hierarchy.

**Plugs:**

**adjustBounds**

`Gaffer::BoolPlug`

Computes new tightened bounding boxes taking into account the removed locations. This can be an expensive operation - turn on with care.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The branches to prune. The specified locations and all locations below them will be removed from the scene.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

## SceneContextVariables

Adds variables which can be referenced by upstream expressions.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

**variables**

`Gaffer::CompoundDataPlug`

The variables to be added - arbitrary numbers of variables can be added here.

## SceneLoop

Applies a user defined processing loop to a scene. The content of the loop is defined by the node network placed between the previous and next plugs. The input scene is sent around this loop for a set number of iterations and then emerges as the output scene.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**indexVariable**

`Gaffer::StringPlug`

The name of a context variable used to specify the index of the current iteration. This can be referenced from expressions within the loop network to modify the operations performed during each iteration of the loop.

**iterations**

`Gaffer::IntPlug`

The number of times the loop is applied to form the output scene.

**next**

`GafferScene::ScenePlug`

The scene to be used as the start of the next iteration of the loop.

**out**

`GafferScene::ScenePlug`

The processed output scene.

**previous**

`GafferScene::ScenePlug`

The result from the previous iteration of the loop, or the input scene if no iterations have been performed yet. The content of the loop is defined by feeding this previous result through the scene processing nodes of choice and back around into the next plug.

## SceneReader

Reads scenes in any of the formats supported by Cortex's SceneInterface.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

**fileName**

`Gaffer::StringPlug`

The name of the file to be loaded. The file can be in any of the formats supported by Cortex's SceneInterfaces.

**out**

`GafferScene::ScenePlug`

The output scene.

**refreshCount**

`Gaffer::IntPlug`

May be incremented to force a reload if the file has changed on disk - otherwise old contents may still be loaded via Gaffer's cache.

**tags**

`Gaffer::StringPlug`

Limits the parts of the scene loaded to only those with a specific set of tags.

## SceneSwitch

Chooses between multiple input scene, passing through the chosen input to the output.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`Gaffer::ArrayPlug`

The input scenes

**index**

`Gaffer::IntPlug`

The index of the input which is passed through. A value of 0 chooses the first input, 1 the second and so on. Values larger than the number of available inputs wrap back around to the beginning.

**out**

`GafferScene::ScenePlug`

The processed output scene.

## SceneTimeWarp

Changes the time at which upstream nodes are evaluated using the following formula :

`upstreamFrame = frame * speed + offset`

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**offset**

`Gaffer::FloatPlug`

Adds to the current frame value (after multiplication with speed).

**out**

`GafferScene::ScenePlug`

The processed output scene.

**speed**

`Gaffer::FloatPlug`

Multiplies the current frame value.

## SceneWriter

Writes scenes to disk. Supports all formats for which a writeable Cortex SceneInterface exists.

**Plugs:**

**dispatcher**

`Gaffer::Plug`

Container for custom plugs which dispatchers use to control their behaviour.

**batchSize**

`Gaffer::IntPlug`

Maximum number of frames to batch together when dispatching tasks.

#### local

`Gaffer::Plug`

Settings used by the local dispatcher.

### fileName

`Gaffer::StringPlug`

The name of the file to be written. Note that unlike image sequences, many scene formats write animation into a single file, so using # characters to specify a frame number is generally not necessary.

### in

`GafferScene::ScenePlug`

The scene to be written.

### out

`GafferScene::ScenePlug`

A direct pass-through of the input scene.

#### bound

`Gaffer::AtomicBox3fPlug`

!!!*EMPTY*!!!

#### transform

`Gaffer::M44fPlug`

!!!*EMPTY*!!!

#### attributes

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

#### object

`Gaffer::ObjectPlug`

!!!*EMPTY*!!!

#### childNames

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

#### globals

`Gaffer::CompoundObjectPlug`

!!!*EMPTY*!!!

#### setNames

`Gaffer::InternedStringVectorDataPlug`

!!!*EMPTY*!!!

#### set

`GafferScene::PathMatcherDataPlug`

!!!*EMPTY*!!!

### requirement

`Gaffer::ExecutableNode::RequirementPlug`

Output connections to downstream nodes which must not be executed until after this node.

**requirements**

`Gaffer::ArrayPlug`

Input connections to upstream nodes which must be executed before this node.

## Seeds

Scatters points evenly over the surface of meshes. This can be particularly useful in conjunction with the Instancer, which can then apply instances to each point.

**Plugs:**

**density**

`Gaffer::FloatPlug`

The number of points per unit area of the mesh, measured in object space.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**name**

`Gaffer::StringPlug`

The name given to the object generated - this will be placed under the parent in the scene hierarchy.

**out**

`GafferScene::ScenePlug`

The processed output scene.

**parent**

`Gaffer::StringPlug`

The location of the mesh to scatter the points over. The generated points will be parented under this location.

**pointType**

`Gaffer::StringPlug`

The render type of the points. This defaults to "gl:point" so that the points are rendered in a lightweight manner in the viewport.

## Set

Creates and edits sets of objects. Each set contains a list of paths to locations within the scene. After creation, sets can be used by the SetFilter to limit scene operations to only the members of a particular set.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**mode**

`Gaffer::IntPlug`

Create mode creates a new set containing only the specified paths. If a set with the same name already exists, it is replaced. Add mode adds the specified paths to an existing set, keeping the paths already in the set. If the set does not exist yet, this is the same as create mode. Remove mode removes the specified paths from an existing set. If the set does not exist yet, nothing is done.

**name**

`Gaffer::StringPlug`

The name of the set that will be created or edited.

**out**

`GafferScene::ScenePlug`

The processed output scene.

**paths**

`Gaffer::StringVectorDataPlug`

The paths to be added to or removed from the set.

## SetFilter

A filter which uses sets to define which locations are matched.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the filter. When it is off, the filter does not match any locations.

**out**

`Gaffer::IntPlug`

The result of the filter. This should be connected into the "filter" plug of a FilteredSceneProcessor.

**set**

`Gaffer::StringPlug`

The name of a set that defines the locations to be matched.

## ShaderAssignment

Assigns shaders to objects.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

**shader**

`Gaffer::Plug`

The shader to be assigned.

## ShaderSwitch

Chooses between multiple input shaders, passing through the chosen shader to the output. The switching is resolved before rendering begins, so no per-sample overhead is incurred during shading.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

Turns the node on and off.

**in**

`Gaffer::ArrayPlug`

The input shaders - the index plug decides which of these is passed through to the output.

**index**

`Gaffer::IntPlug`

The index of the input which is passed through. A value of 0 chooses the first input, 1 the second and so on. Values larger than the number of available inputs wrap back around to the beginning.

**out**

`Gaffer::Plug`

The output shader.

## Sphere

Produces scenes containing a sphere.

**Plugs:**

**divisions**

`Gaffer::V2iPlug`

Controls tesselation of the sphere when type is Mesh.

**x**

`Gaffer::IntPlug`

!!!*EMPTY*!!!

#### y

`Gaffer::IntPlug`

!!!*EMPTY*!!!

### enabled

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

### name

`Gaffer::StringPlug`

The name of the object in the output scene.

### out

`GafferScene::ScenePlug`

The output scene.

### radius

`Gaffer::FloatPlug`

Radius of the sphere.

### sets

`Gaffer::StringPlug`

A list of sets to include the object in. The names should be separated by spaces.

### thetaMax

`Gaffer::FloatPlug`

Limits the extent of the sphere around the pole axis. Valid values are in the range [0,360].

### transform

`Gaffer::TransformPlug`

The transform applied to the object.

#### translate

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

#### rotate

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

#### scale

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

#### pivot

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

### type

`Gaffer::IntPlug`

The type of object to produce. May be a SpherePrimitive or a Mesh.

### zMax

`Gaffer::FloatPlug`

Limits the extent of the sphere along the upper pole. Valid values are in the range [-1,1] and should always be greater than zMin.

**zMin**

`Gaffer::FloatPlug`

Limits the extent of the sphere along the lower pole. Valid values are in the range [-1,1] and should always be less than zMax.

## StandardAttributes

Modifies the standard attributes on objects - these should be respected by all renderers.

**Plugs:**

**attributes**

`Gaffer::CompoundDataPlug`

The attributes to be applied - arbitrary numbers of user defined attributes may be added as children of this plug via the user interface, or using the CompoundDataPlug API via python.

**visibility.value**

`Gaffer::BoolPlug`

Whether or not the object can be seen - invisible objects are not sent to the renderer at all. Typically more fine grained (camera, reflection etc) visibility can be specified using a renderer specific attributes node. Note that making a parent location invisible will always make all the children invisible too, regardless of their visibility settings.

**doubleSided.value**

`Gaffer::BoolPlug`

Whether or not the object can be seen from both sides. Single sided objects appear invisible when seen from the back.

**transformBlur.value**

`Gaffer::BoolPlug`

Whether or not transformation animation on the object is taken into account in the rendered image. Use the transformBlurSegments plug to specify the number of segments used to represent the motion.

**transformBlurSegments.value**

`Gaffer::IntPlug`

The number of segments of transform animation to pass to the renderer when transformBlur is on.

**deformationBlur.value**

`Gaffer::BoolPlug`

Whether or not deformation animation on the object is taken into account in the rendered image. Use the deformationBlurSegments plug to specify the number of segments used to represent the motion.

**deformationBlurSegments.value**

`Gaffer::IntPlug`

The number of segments of transform animation to pass to the renderer when transformBlur is on.

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

### filter

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

### global

`Gaffer::BoolPlug`

Causes the attributes to be applied to the scene globals instead of the individual locations defined by the filter.

### in

`GafferScene::ScenePlug`

The input scene

### out

`GafferScene::ScenePlug`

The processed output scene.

## StandardOptions

Specifies the standard options (global settings) for the scene. These should be respected by all renderers.

**Plugs:**

### enabled

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

### in

`GafferScene::ScenePlug`

The input scene

### options

`Gaffer::CompoundDataPlug`

The options to be applied - arbitrary numbers of user defined options may be added as children of this plug via the user interface, or using the CompoundDataPlug API via python.

#### renderCamera.value

`Gaffer::StringPlug`

The primary camera to be used for rendering. If this is not specified, then a default orthographic camera positioned at the origin is used.

#### renderResolution.value

`Gaffer::V2iPlug`

The resolution of the image to be rendered. Use the resolution multiplier as a convenient way to temporarily render at multiples of this resolution.

#### pixelAspectRatio.value

`Gaffer::FloatPlug`

The aspect ratio (x/y) of the pixels in the rendered image.

#### resolutionMultiplier.value

`Gaffer::FloatPlug`

Multiplier applied to the render resolution.

**renderCropWindow.value**

`Gaffer::Box2fPlug`

Limits the render to a region of the image. The rendered image will have the same resolution as usual, but areas outside the crop will be rendered black. Coordinates range from 0,0 at the top left of the image to 1,1 at the bottom right. The crop window tool in the viewer may be used to set this interactively.

**overscan.value**

`Gaffer::BoolPlug`

Adds extra pixels to the sides of the rendered image. This can be useful when camera shake or blur will be added as a post process. This plug just enables overscan as a whole - use the overscanTop, overscanBottom, overscanLeft and overscanRight plugs to specify the amount of overscan on each side of the image.

**overscanTop.value**

`Gaffer::FloatPlug`

The amount of overscan at the top of the image. Specified as a 0-1 proportion of the original image height.

**overscanBottom.value**

`Gaffer::FloatPlug`

The amount of overscan at the bottom of the image. Specified as a 0-1 proportion of the original image height.

**overscanLeft.value**

`Gaffer::FloatPlug`

The amount of overscan at the left of the image. Specified as a 0-1 proportion of the original image width.

**overscanRight.value**

`Gaffer::FloatPlug`

The amount of overscan at the right of the image. Specified as a 0-1 proportion of the original image width.

**cameraBlur.value**

`Gaffer::BoolPlug`

Whether or not camera motion is taken into account in the renderered image. To specify the number of segments to use for camera motion, use a StandardAttributes node filtered for the camera.

**transformBlur.value**

`Gaffer::BoolPlug`

Whether or not transform motion is taken into account in the renderered image. To specify the number of transform segments to use for each object in the scene, use a StandardAttributes node with appropriate filters.

**deformationBlur.value**

`Gaffer::BoolPlug`

Whether or not deformation motion is taken into account in the renderered image. To specify the number of deformation segments to use for each object in the scene, use a StandardAttributes node with appropriate filters.

**shutter.value**

`Gaffer::V2fPlug`

The interval over which the camera shutter is open. Measured in frames, and specified relative to the frame being rendered.

**out**

`GafferScene::ScenePlug`

The processed output scene.

## SubTree

A node for extracting a specific branch from a scene.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**in**

`GafferScene::ScenePlug`

The input scene

**includeRoot**

`Gaffer::BoolPlug`

Causes the root location to also be kept in the output scene, in addition to its children. For instance, if the scene contains only /city/street/house and the root is set to /city/street, then the new scene will by default contain only /house - but the includeRoot setting will cause it to contain /street/house instead.

**out**

`GafferScene::ScenePlug`

The processed output scene.

**root**

`Gaffer::StringPlug`

The location to become the new root for the output scene. All locations below this will be kept, and all others will be discarded.

## Text

Creates an object containing a polygon representation of an arbitrary string of text.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs an empty scene.

**font**

`Gaffer::StringPlug`

The font to use - this should be a .ttf font file which is located on the paths specified by the IECORE_FONT_PATHS environment variable.

**name**

`Gaffer::StringPlug`

The name of the object in the output scene.

**out**

`GafferScene::ScenePlug`

The output scene.

**sets**

`Gaffer::StringPlug`

A list of sets to include the object in. The names should be separated by spaces.

**text**

`Gaffer::StringPlug`

The text to output. This is triangulated into a mesh representation using the specified font.

**transform**

`Gaffer::TransformPlug`

The transform applied to the object.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## Transform

Applies a transformation to the local matrix of all locations matched by the filter.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the node. When it is off, the node outputs the input scene unchanged.

**filter**

`Gaffer::IntPlug`

The filter used to control which parts of the scene are processed. A Filter node should be connected here.

**in**

`GafferScene::ScenePlug`

The input scene

**out**

`GafferScene::ScenePlug`

The processed output scene.

**space**

`Gaffer::IntPlug`

The space in which the transformation is specified. Note that no matter which space is chosen, only the local matrices of the filtered locations are ever modified. They are simply modified in such as way as to emulate a modification in the chosen space. Local : The transformation is specified in local space and is therefore post-multiplied onto the local matrix. Parent : The transformation is specified in parent space and is therefore pre-multiplied onto the local matrix. World : The transformation is specified in world space and will therefore be applied as if the whole world was moved. This effect is then applied on a per-location basis to each of the filtered locations. Reset Local : The local matrix is replaced with the specified transform. Reset World : The transformation is specified as an absolute matrix in world space. Each of the filtered locations will be moved to this absolute position.

**transform**

`Gaffer::TransformPlug`

The transform to be applied.

**translate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**rotate**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**scale**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

**pivot**

`Gaffer::V3fPlug`

!!!*EMPTY*!!!

## UnionFilter

Combines several input filters, matching the union of all the locations matched by them.

**Plugs:**

**enabled**

`Gaffer::BoolPlug`

The on/off state of the filter. When it is off, the filter does not match any locations.

**in**

`Gaffer::ArrayPlug`

The filters to be combined. Any number of inputs may be added here.

**out**

`Gaffer::IntPlug`

The result of the filter. This should be connected into the "filter" plug of a FilteredSceneProcessor.

**Version 0.18.00**
**Last updated 2015-11-12 17:45:29 GMT**