



User Manual

Contents

Installation Guide.....	4
Getting Started.....	5
Home Window	5
Light Manager Menu.....	6
Model Manager Menu	7
Importing a Model	7
Transforming a Model.....	8
Creating an Instance of a Model.....	8
Removing Models from the Scene.....	9
Environment Map Settings Menu	10
Setting an Environment Map	10
Changing the Influence of an Environment Map.....	11
Camera Settings Menu.....	12
Changing camera attributes.....	12
Depth of Field.....	12
Resetting the Camera	14
Material Library Menu	15
Applying Materials to a Model.....	15
Removing a Material from the Library.....	16
OSL Hypershader Menu	17
The Node Graph.....	17
Adding Nodes.....	17
Shader Node.....	18
Float Node.....	18
Int Node	19
Vector Node	19
Color Node	19
Normal Node.....	20
Point Node	20
Image Node.....	20
Connecting nodes	21
Removing Nodes and Connections	22
Creating Shaders	22

Adding shaders to the Material Library	22
Applying a Material to a Model	23
Loading a Material from the Material library to the Node Graph	23
Saving changes to a node graph	24
Importing external Node Graphs	24
More complicated Node Graphs.....	24
Tab Menu's	26
File.....	26
Render.....	26
Settings.....	27
OSL	28
Types Supported	28
Mathematical Operations	28
Multiplications	28
Additions	28
Divisions	28
Subtractions	29
Cross product	29
Dot Product.....	29
Length	29
Distance.....	29
Mix	29
Power	30
Not Equal To.....	30
Equal To.....	30
Assignment.....	30
Array Assignment.....	31
Referencing an Array	31
Floor	31
Min	31
Max	31
Normalize	31
Greater Than	31
Less Than or Equal To.....	32
Syntax.....	32
For loops	32

If Statements.....	32
Utilities.....	33
Closures.....	33
Example shaders	35

Installation Guide

Dependencies:

- Qt 5.0 or higher
- NVidia's OptiX 3.7
- CUDA 6.5
- Flex
- Bison
- Open Shader Language 1.0
- Assimp
- Boost 1.54
- OpenImageIO

Installation:

This system builds on mac and linux. However our application depends on you having an Nvidia GPU.

1. Install all of the dependencies listed above.
2. Set an environment variable for the build location of Open Shader language under the name OSL_DIR e.g. export OSL_DIR=~/.OpenShaderLanguage
3. Clone/Download Helios
4. In the Helios directory open the helios.pro file and modify the following according to your system setup
 - a. CUDA_DIR
 - b. OPTIX_DIR
 - c. GENCODE
5. qmake
6. make clean
7. make
8. You should now have Helios installed.

Getting Started

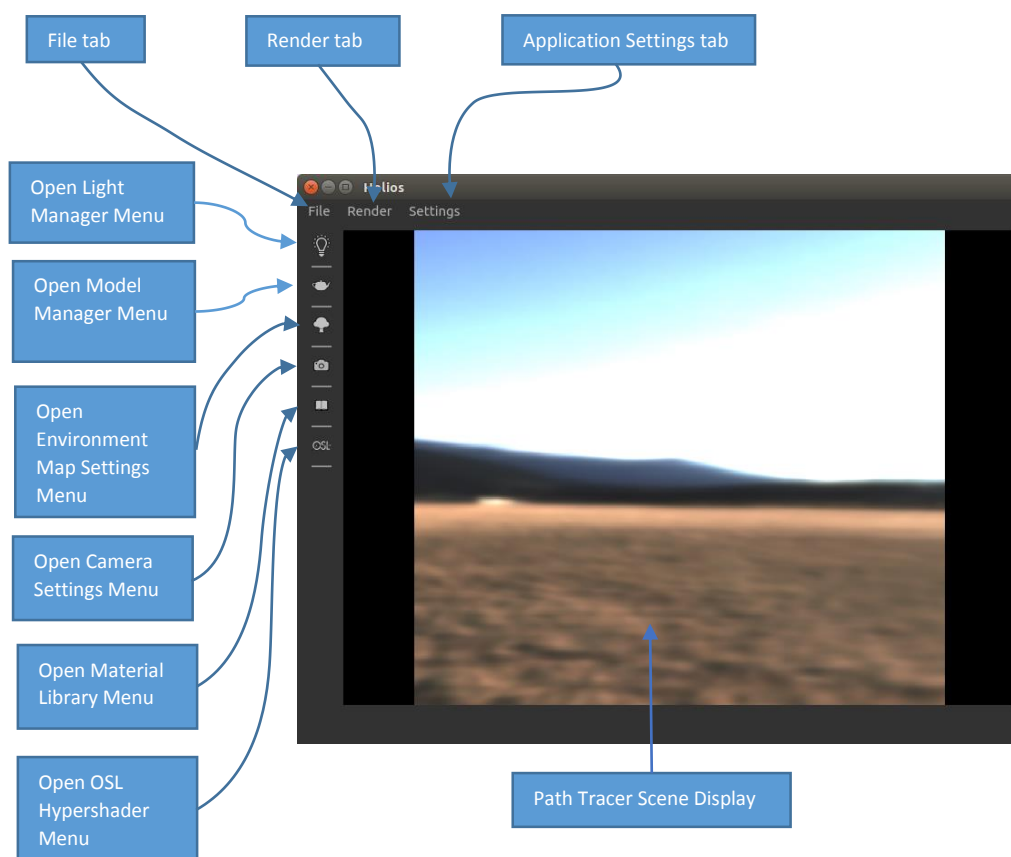
We have provided a number of example materials and scenes with the installation of this application see importing sections below on how to access these.

Home Window


When you launch Helios the first window you will encounter will be the home window. This is where our path traced scene lives and from here you will be able to launch all other menus to interface with it. Simply click the relative button for desired menu you wish to interface with to open it.

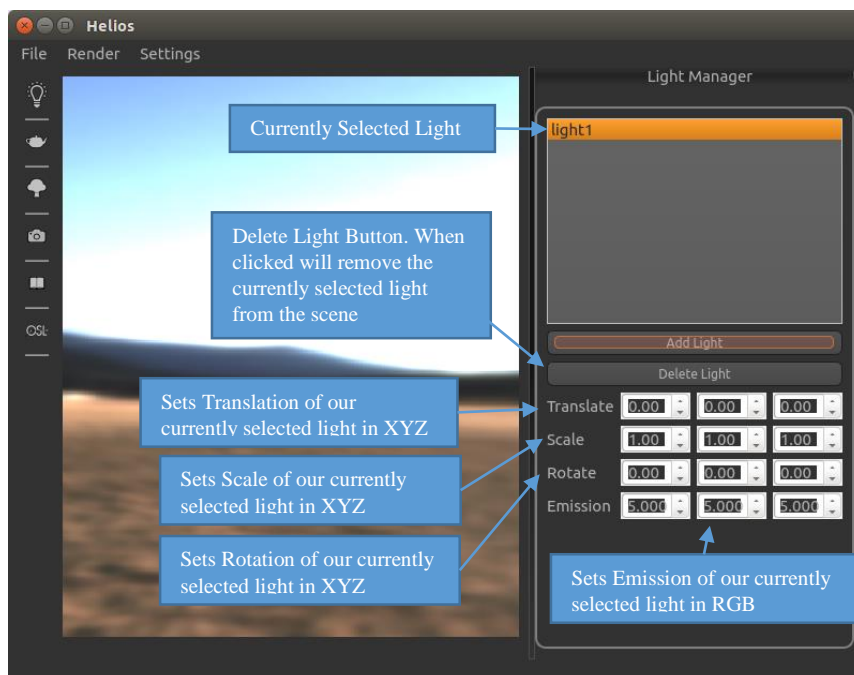
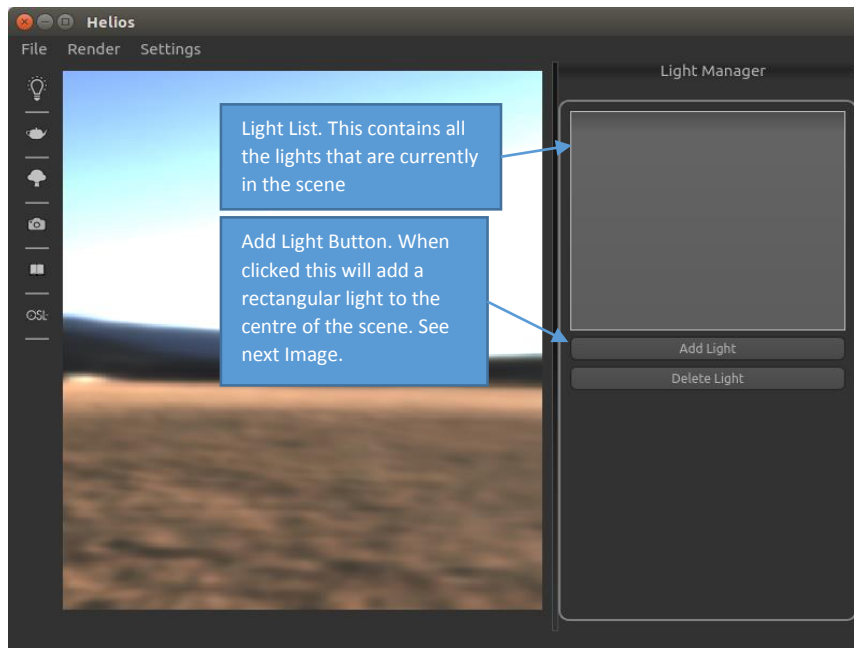
There are 4 mouse controls to traverse around the path tracer scene.

- Left click & Drag to rotate the scene
- Right click & Drag to pan around the scene
- Mouse wheel to zoom in and out
- Middle click & Drag to rotate the environment map




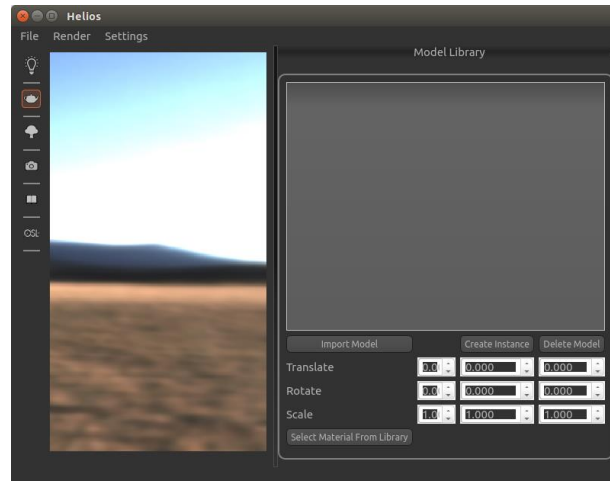
Light Manager Menu

To launch the light manager click the lightbulb button  on the left tool bar. This will open up a dockable menu on the right side of the home window.



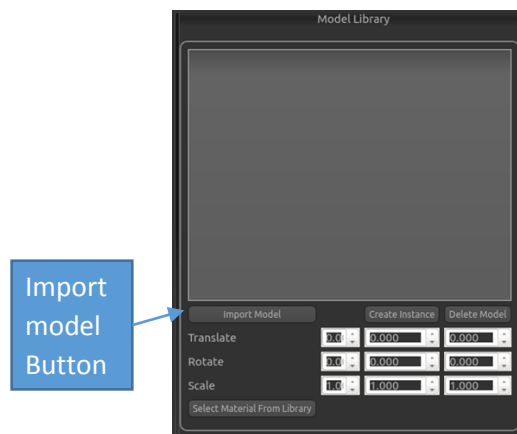
Model Manager Menu

To launch the model manager click the teapot button  on the left tool bar. This will open up a dockable menu on the right side of the home window. This menu provides the controls to import a model into our path tracer. It also allows you to edit attributes models in the scene.

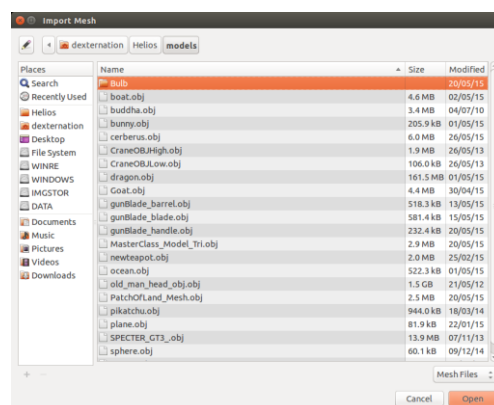


Importing a Model

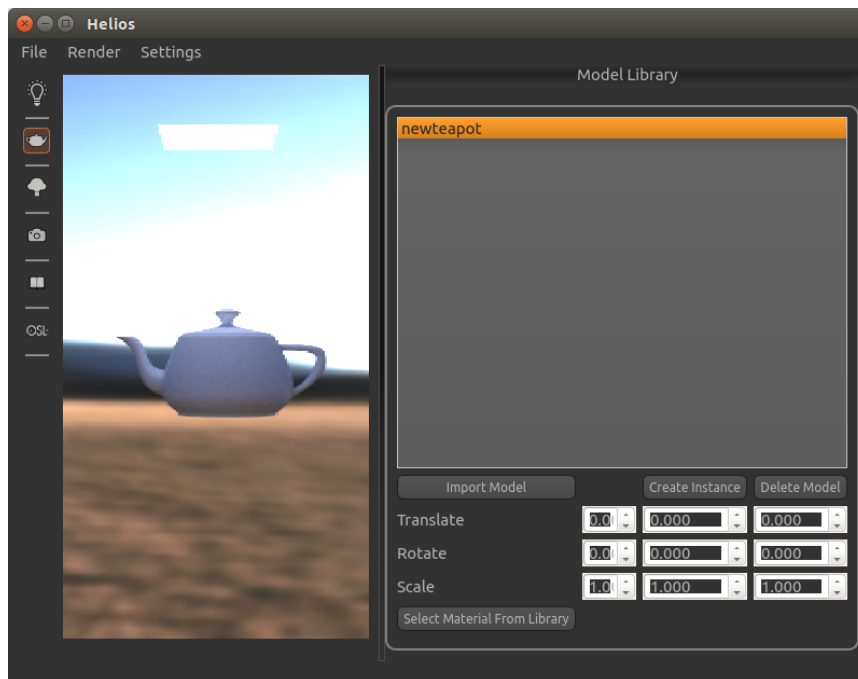
To import a model the firstly click the import button located half way down the model menu.



This will then open a browser for you to select the model that you wish to import from your computer. Simply select a model and click “Open”.



Your model should now appear as the currently selected in the model list in the top half of the menu.

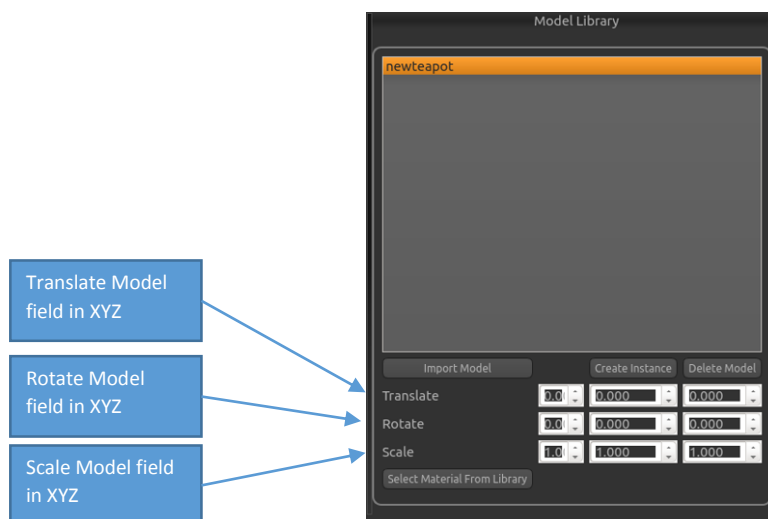


Transforming a Model

To transform a model in our scene we provide 3 different transform types.

- Translation
- Scale
- Rotation

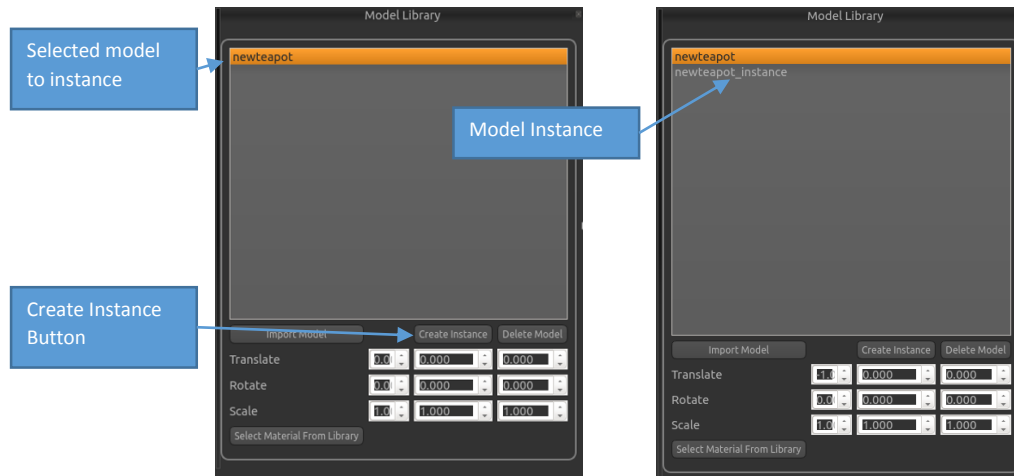
Simple edit the spin box's located near the bottom of the menu associated with the desired transform field to transform your selected model.



Creating an Instance of a Model

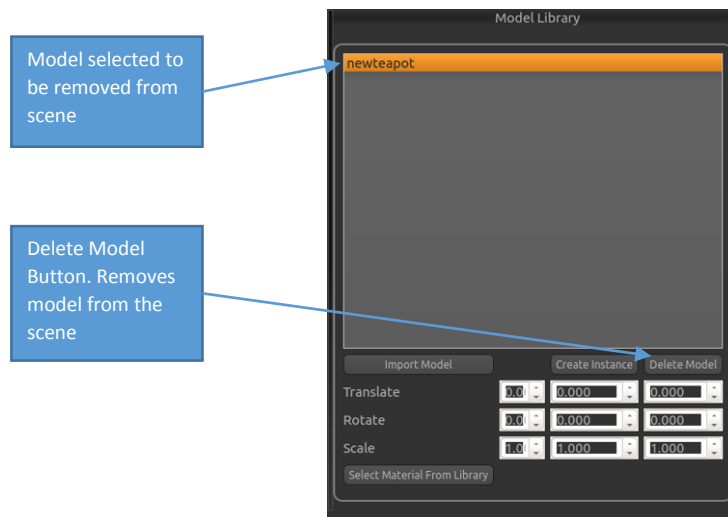
Creating instances is an important feature in our path tracer to the memory limitations with GPU path tracers. If you have large amounts of geometry for a single scene its advisable take advantage of this as much as possible to improve performance and reliability.

To create an instance simply select the model you wish to instance from the model list and click the “Create Instance” Button located above the transform fields. This will then create an instance of the model in the model list




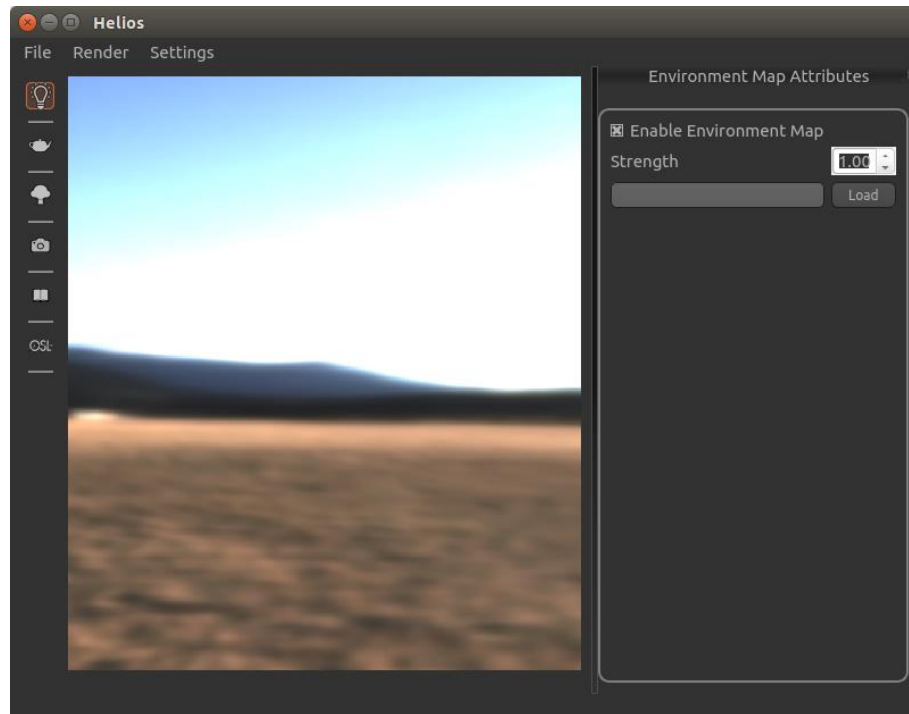
Removing Models from the Scene

If you wish to remove a model from the scene. Simple select the model from the model list that you wish to remove and click the “Delete Model” button located to the right above the transform fields.



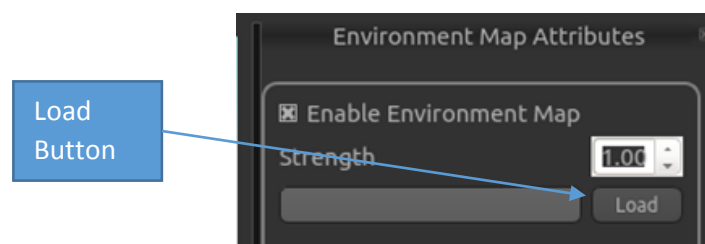
Environment Map Settings Menu

To launch the environment map settings menu click the tree button  on the left tool bar. This will open up a dockable menu on the right side of the home window. From here you will be able to set your own environment map and change the influence that it has on the scene.



Setting an Environment Map

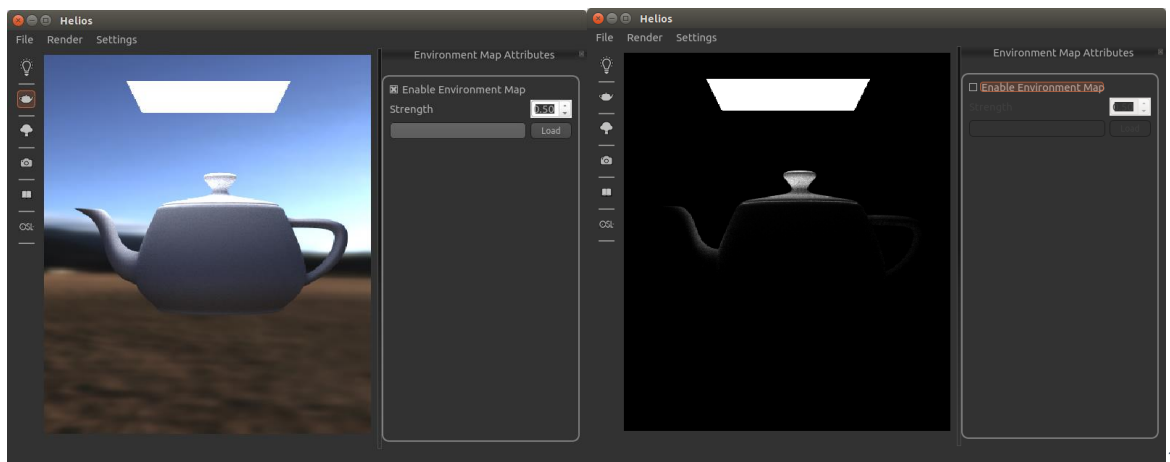
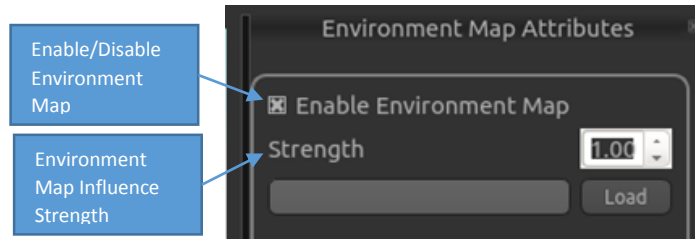
You may wish to use your own custom environment map for your scene. To set your own environment map first simply click the “Load” button located on the right side of the menu.



This will prompt a browser window in which you can select your environment map for your scene. Note that Helios only supports HDR images for environment maps.


Changing the Influence of an Environment Map

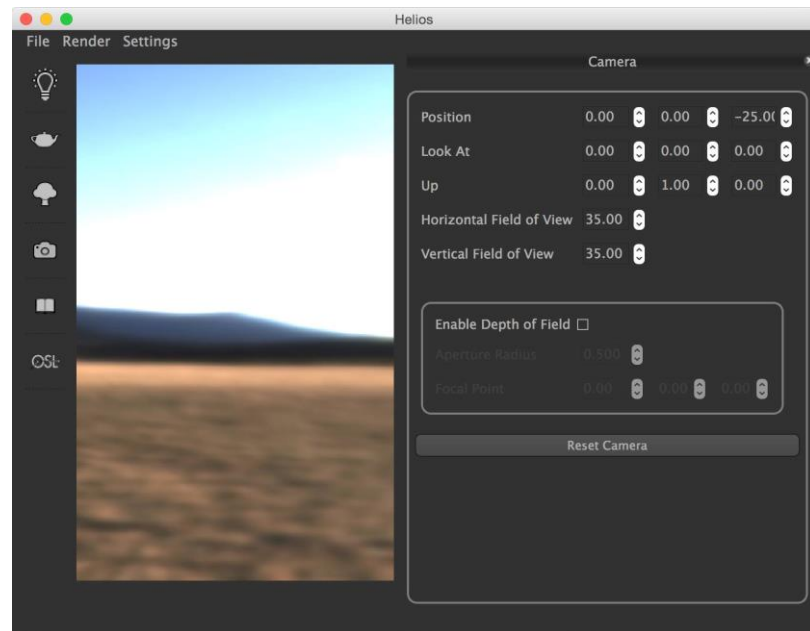
You changing the influence of an environment map can be achieved in 2 ways. You can either completely disable the environment map by unchecking the “Enable Environment Map” check box. Alternatively you may change the influence of the environment map by modifying the “Strength” parameter”.



(left) Environment Map at 0.5 Strength (Right) Environment Map disabled

Camera Settings Menu

To launch the camera settings menu click the camera button  on the left tool bar. This will open up a dock able menu on the right side of the home window.

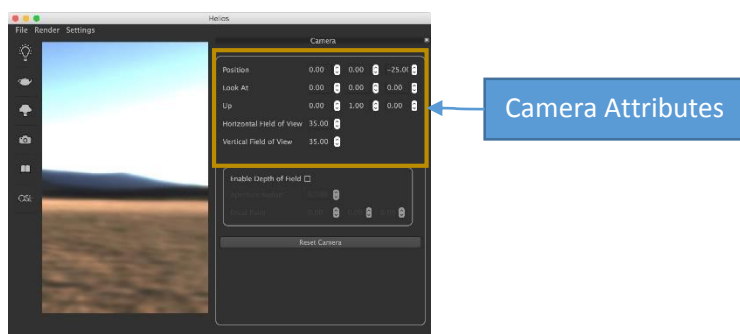


Changing camera attributes

You can change attributes of our scene camera. We currently support,

- Position
- Look At direction
- Up direction
- Horizontal field of view
- Vertical field of view

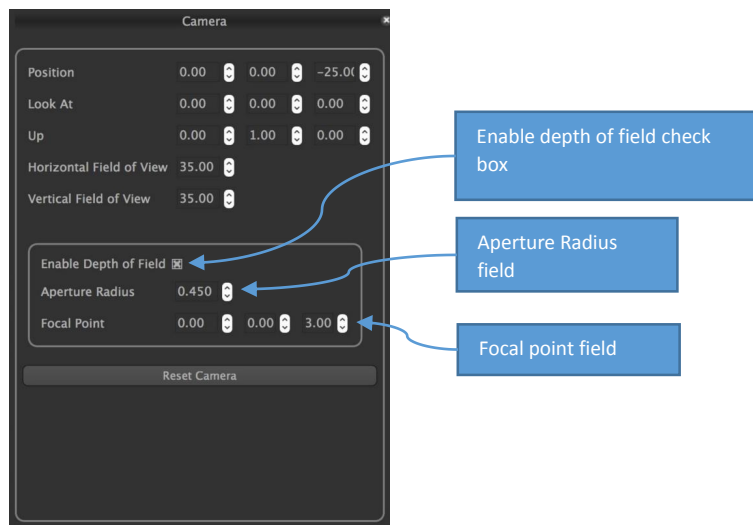
These are found at the top of our camera settings menu.



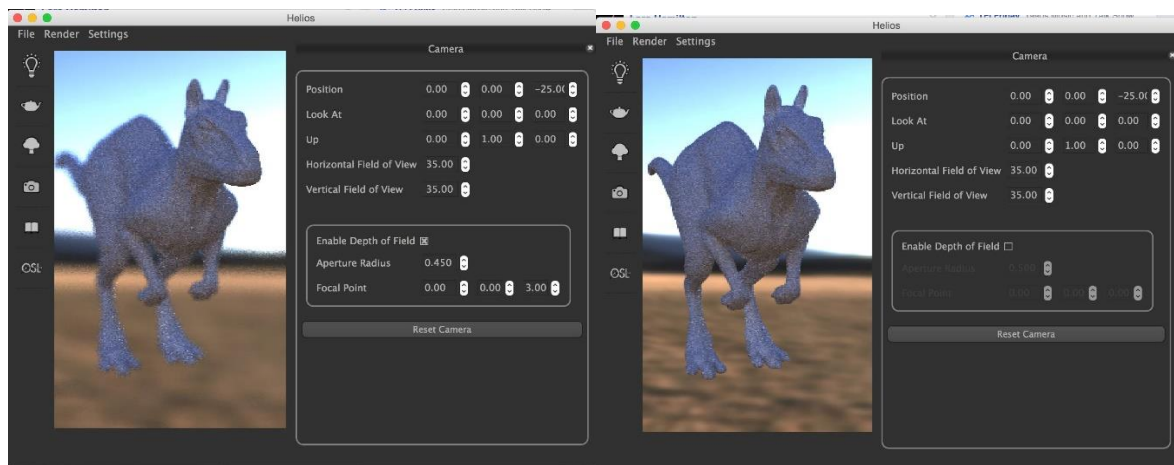
Depth of Field

Our camera supports depth of field. To enable this just check the depth of field check box. With depth of field we provide you with 2 attributes,

- Aperture Radius
- Focal Point

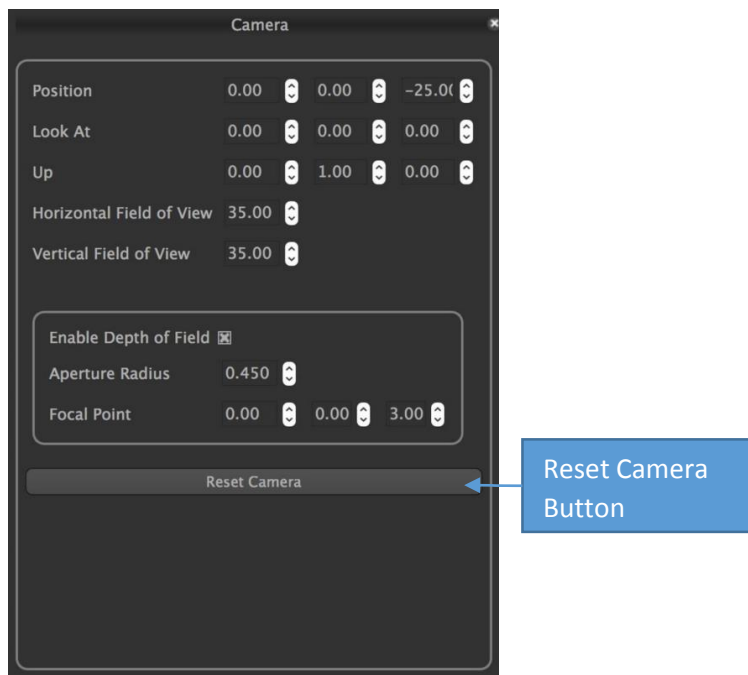


(Left) Camera with Depth of field enabled. (Right) Camera with Depth of field disabled




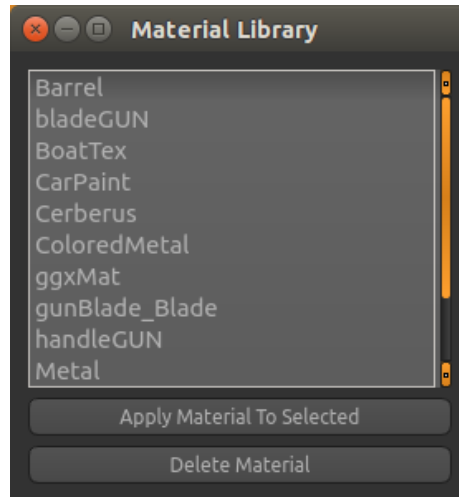
Resetting the Camera

If you need to reset your camera then this is achievable by clicking the “Reset Camera” button located at the bottom of the Camera settings menu.



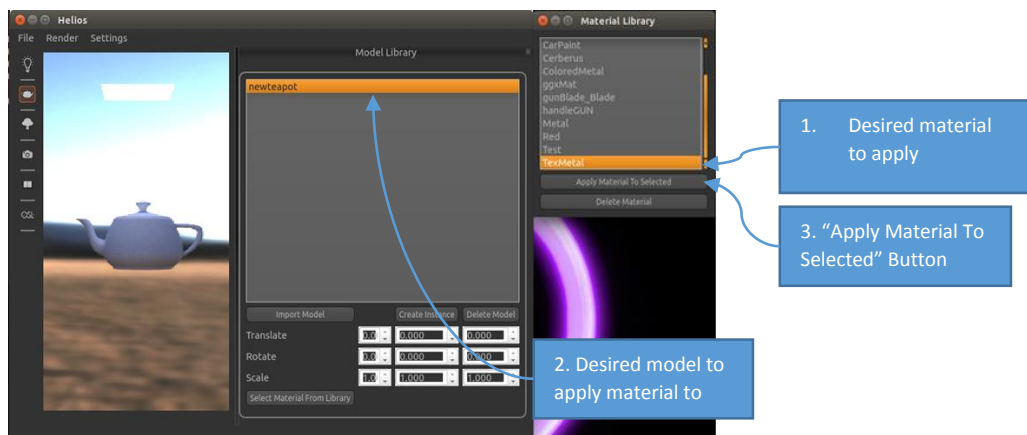
Material Library Menu

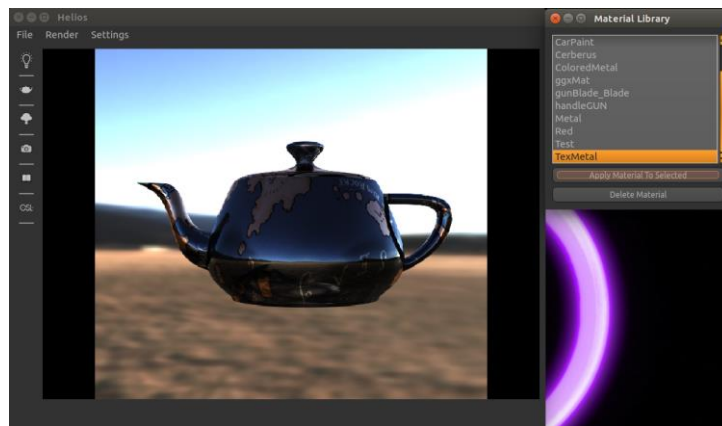
To launch the Material Library menu click the book button  on the left tool bar. This will open up a dockable menu on the right side of the home window. This menu contains a list of compiled materials. You can either apply these materials to a model in the scene or remove them from the library.



Applying Materials to a Model

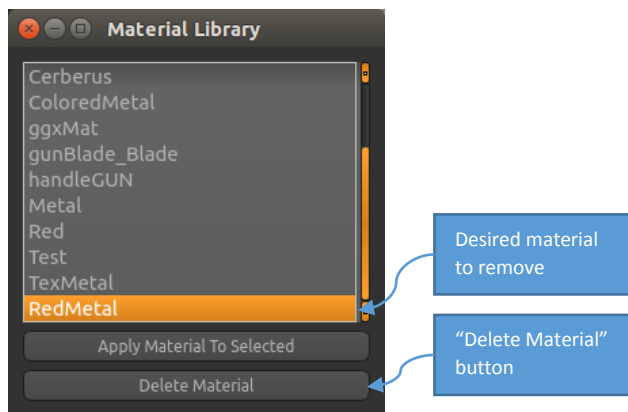
If you wish to apply a material to a model firstly select your desired material in the material library to apply. Secondly select the desired model in the Model Manager Menu to apply the material to. Now click the "Apply Material To Selected" Button.






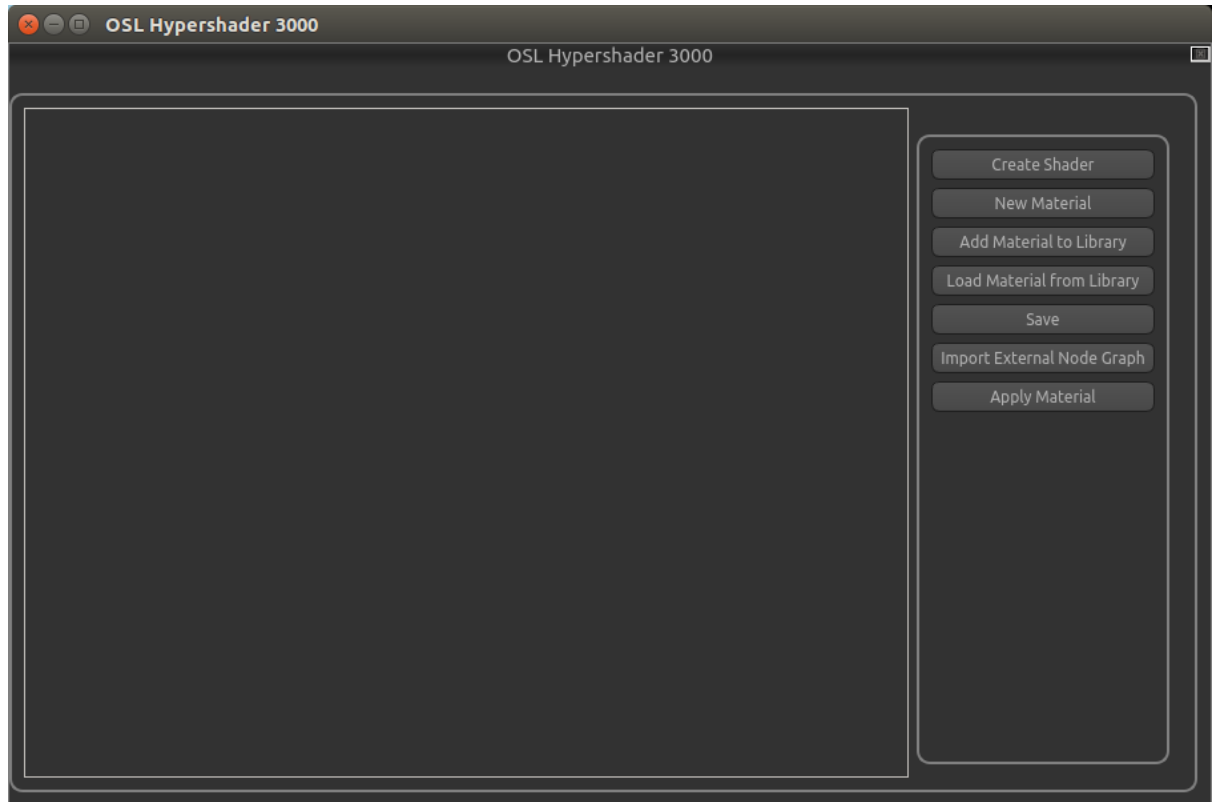
Removing a Material from the Library

If you wish to remove a material from the library simply select the desired material to remove from the list and click the “Delete Material” button.



OSL Hypershader Menu

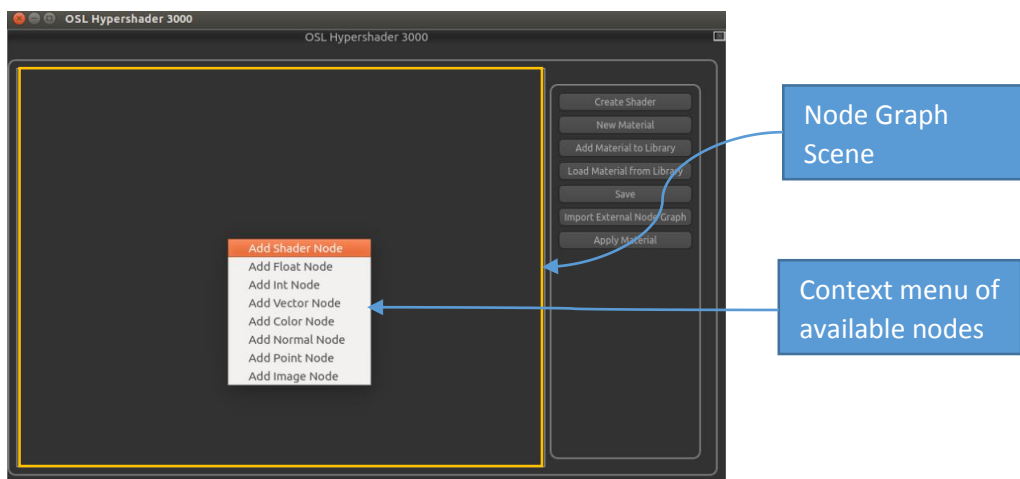
To launch the OSL Hypershader menu click the OSL logo button  on the left tool bar. This will open up a dockable menu on the right side of the home window. This menu will allow you to create your own custom OSL shaders that may be applied to models in the scene. Our shader creation is achieved through the creation of node graphs that can then be compiled and used as a material.



The Node Graph

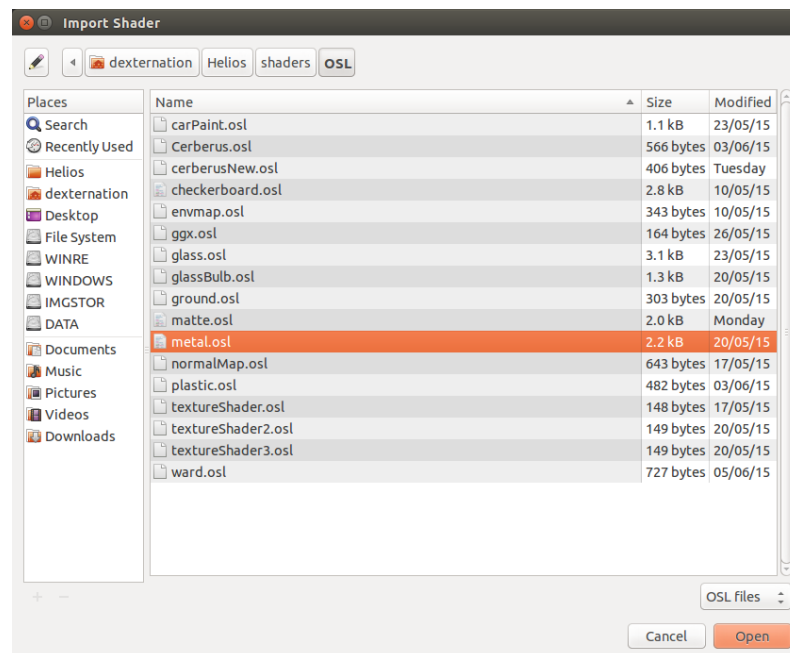
Adding Nodes

We have a variety of nodes available for you to create your OSL shader in our node graph. To add a node to the node graph simply right click within the node graph scene. A context menu will now appear where you can select your desired node to add to the scene.

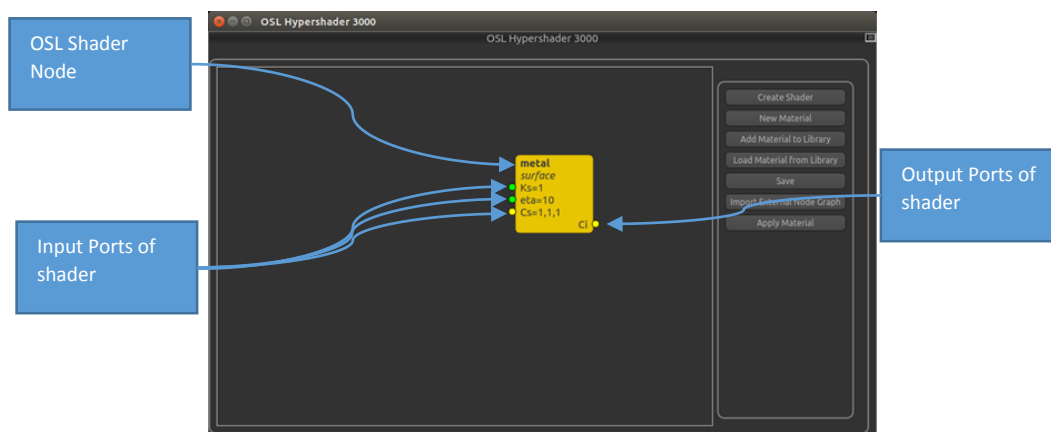


Shader Node

When adding a shader node to the node graph a browser will appear. From here you can select your prewritten OSL shader to import into the scene. Simply navigate to your OSL shader, select it and click “Open”. Note that all OSL shaders must be saved with the “.osl” file extension.



You will now have a shader node in appear in the node graph build from the shader selected. The following image is a shader node generated from importing the metal shader from the image above.



For more information about our OSL support see the OSL section. Something to keep in mind with this node is that if you change the OSL program that you have written you will need to reimport it for the changes to be made to the node.

Float Node

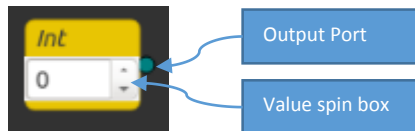
When adding a float node to the node graph the following node will appear.



This has one output port that may be connected to multiple shader nodes inputs that are also of type float. These can be recognised by having a **green** coloured port. You can change the value of this node by editing the spin box located in the centre.

Int Node

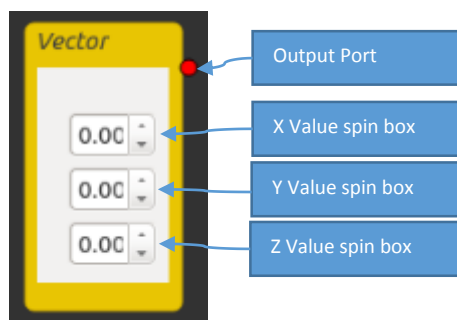
When adding an Int node to the node graph the following node will appear.



This has one output port that may be connected to multiple shader nodes inputs that are also of type int. These can be recognised by having a **blue** coloured port. You can change the value of this node by editing the spin box located in the centre.

Vector Node

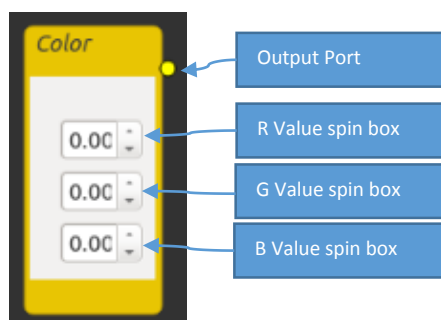
When adding a Vector node to the node graph the following node will appear.



This has one output port that may be connected to multiple shader nodes inputs that are also of type vector. These can be recognised by having a **Red** coloured port. You can change the value of the node by editing the 3 spin boxes located in the centre. These relate the to X,Y and Z components of the vector.

Color Node

When adding a Color node to the node graph the following node will appear.

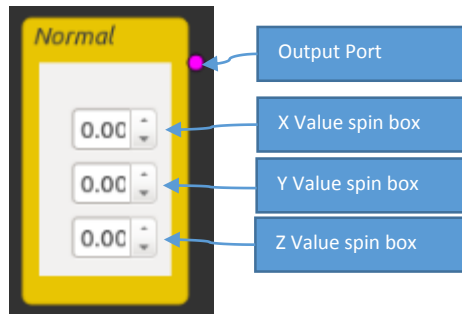


This has one output port that may be connected to multiple shader nodes inputs that are also of type colour. These can be recognised by having a **Yellow** coloured port. You can change the value of

the node by editing the 3 spin boxes located in the centre. These relate the to R,G and B components of the colour.

Normal Node

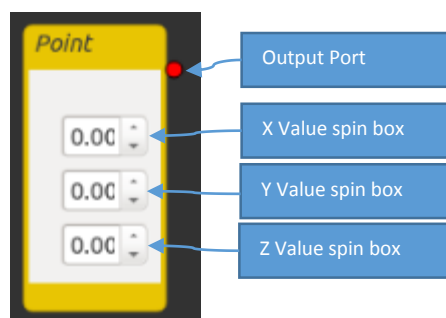
When adding a Normal node to the node graph the following node will appear.



This has one output port that may be connected to multiple shader nodes inputs that are also of type normal. These can be recognised by having a **Pink** coloured port. You can change the value of the node by editing the 3 spin boxes located in the centre. These relate the to X,Y and Z components of the normal.

Point Node

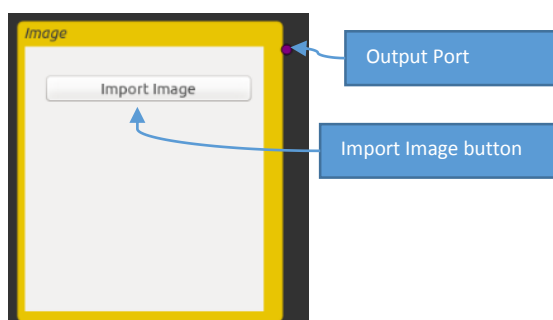
When adding a Point node to the node graph the following node will appear.



This has one output port that may be connected to multiple shader nodes inputs that are also of type point. These can be recognised by having a **Red** coloured port. You can change the value of the node by editing the 3 spin boxes located in the centre. These relate the to X,Y and Z components of the point.

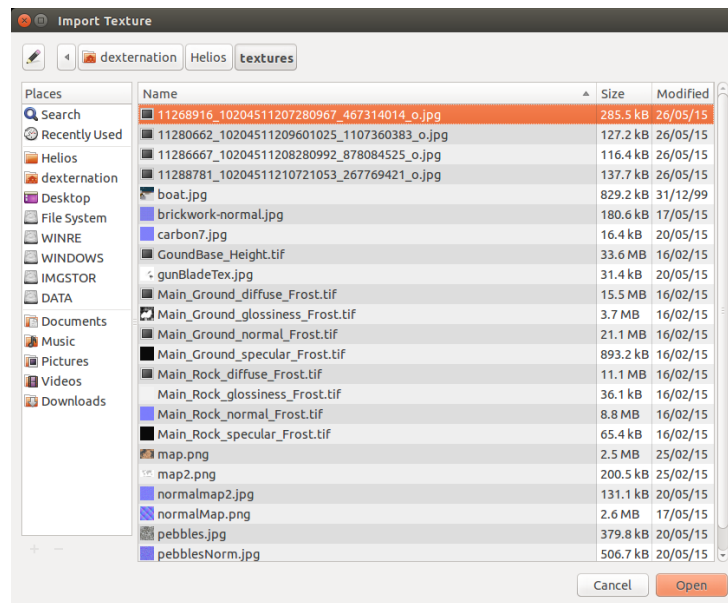
Image Node

When adding an Image node to the node graph the following node will appear.



This has one output port that may be connected to multiple shader nodes inputs that are of type string. These can be recognised by having a **Purple** coloured port. You can select the image for the

node to represent by clicking the “Import Image” Button at the top of the node. This will open a browser for you to select you image.

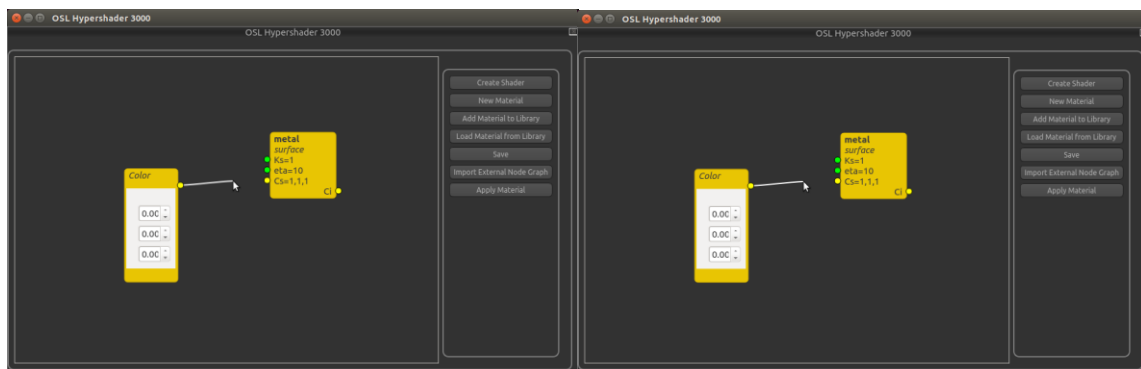


Simply locate your image and click “Open”. Your Image node should now look like something similar to this if you have imported the image correctly.



Connecting nodes

Nodes connect together via their ports. An output port can only be connected to an input port of the same type. To connect two ports together simply left click the desired input/output port and drag it to the desired connected output/input port respectively.

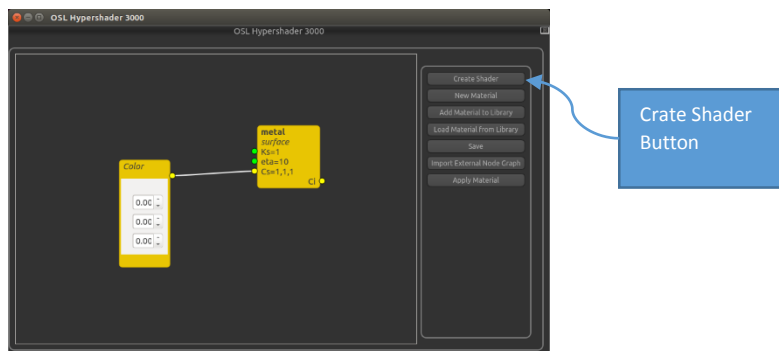


Removing Nodes and Connections

To remove a node or connection simply middle mouse click upon the desired node or connection.

Creating Shaders

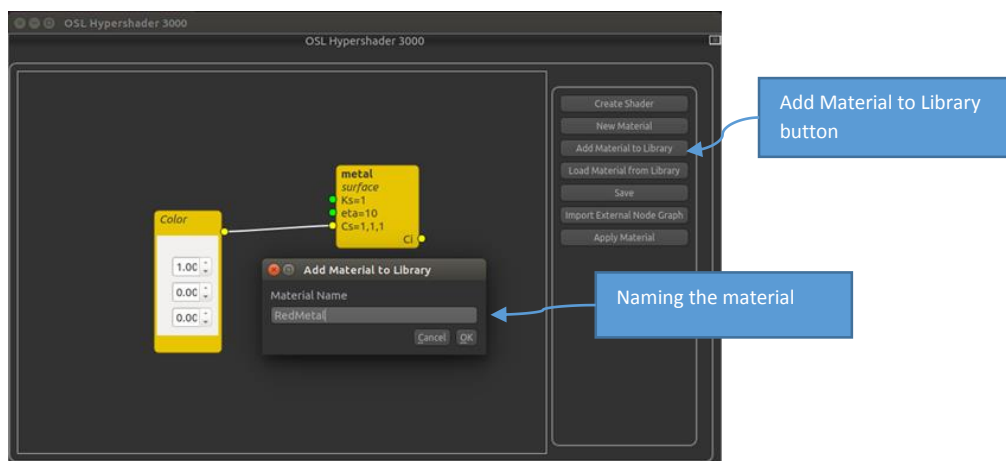
Creating shaders is done by connecting shader programs together in a desired order of execution. For a node graph to be a valid shader it must have the OSL output value "Ci". This means for the simplest case you will need at least one shader node. See below I have connected a colour node to a metal shader.



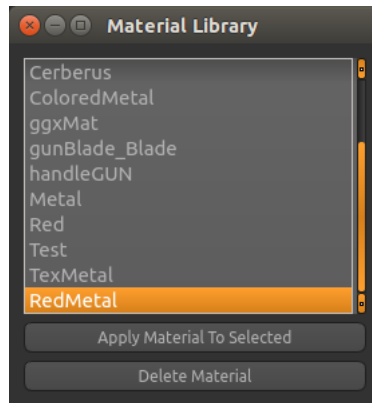
To compile this shader you must then click the "Create shader Button located at the top right of the menu.

Adding shaders to the Material Library

Once compiled shaders can be added to the material library. The material library stores you material so you can load the material to the node graph again or to allow you to apply the material to other models in the scene. Continuing from the previous example I have changed the colour node to have a red colour. To add the material to the library you must click the "Add Material to Library" button. The library requires a name to save the material under and you will be prompted to input one when adding a material to the library. I have suitably called mine "RedMetal".

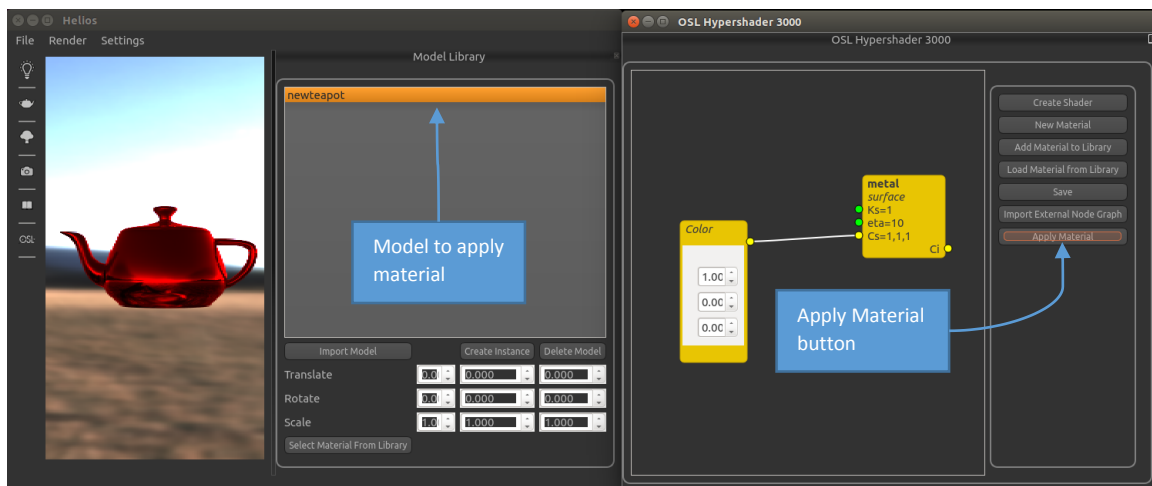


Once an appropriate name is chosen click "ok". If you now open up the Material Library Menu you will now see that you material has been added to the library.



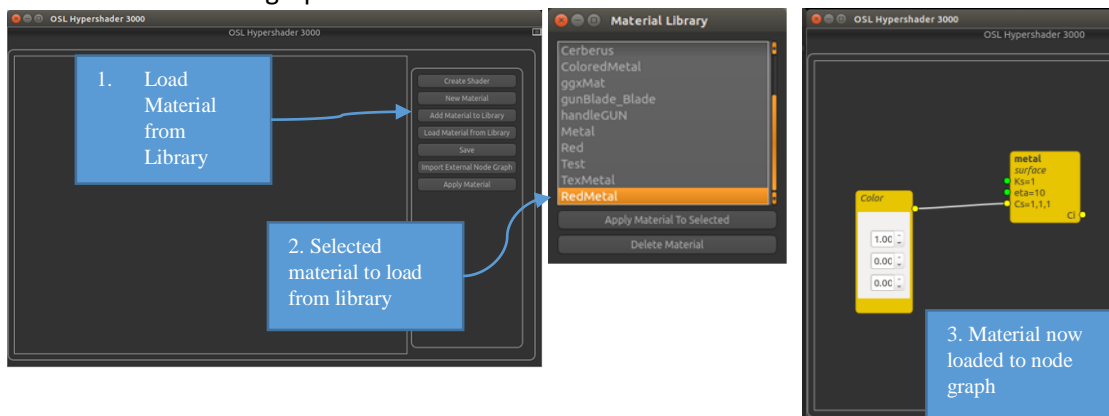
Applying a Material to a Model

To apply a material to a model simply click the “Apply Material” with you desired model selected in the model manager menu. For a material to be applied it is required that it is first compiled and secondly added to the material library. If these steps have not been executed you will be promoted to do so.



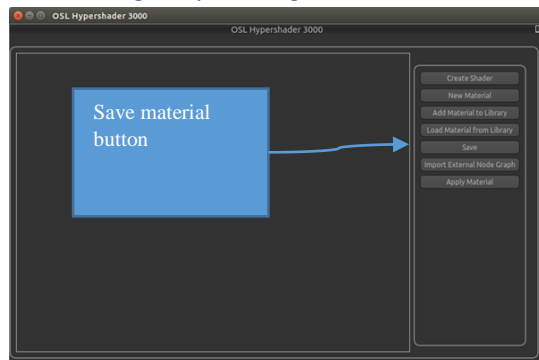
Loading a Material from the Material library to the Node Graph

You can load a material from the material library to the node graph. Simply select “Load Material from library”. The material library will be shown where you can select the desired material to load into the node graph.



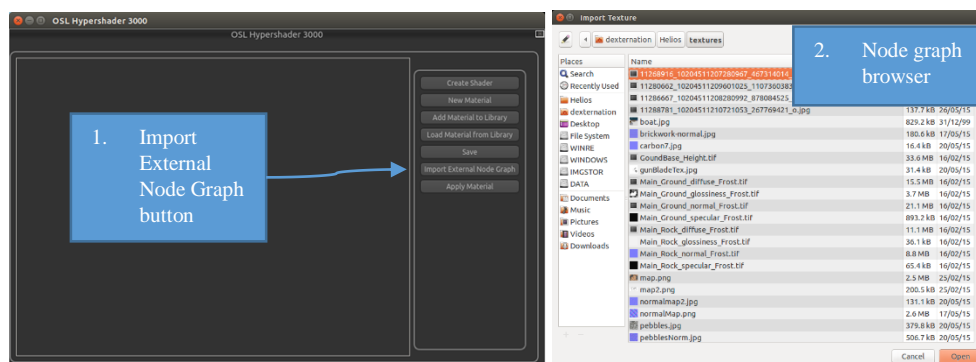
Saving changes to a node graph

If you have changed a material in the node graph that has been added to the library you can save these changes by clicking the “Save” button on the right of the OSL Hypershader.



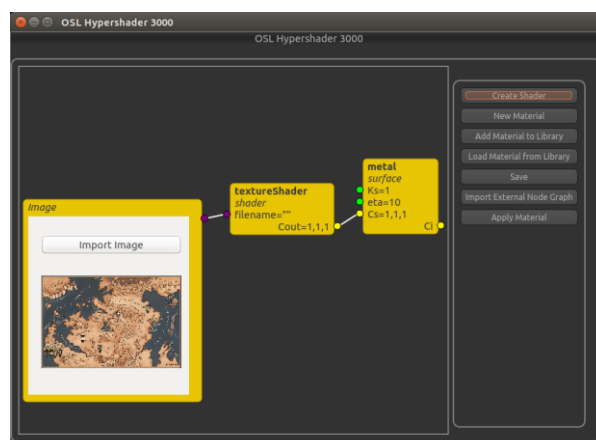
Importing external Node Graphs

You can import a node graph from a file. This is achieved by clicking the “Import External Node Graph” button. This will prompt a browser for you to select your external node graph. Note that all node graph files are saved with the “.hel” extension. Another point to be aware of is that all node graphs that are saved to the library are saved in a folder called “NodeGraphs” in the executable directory of the application.

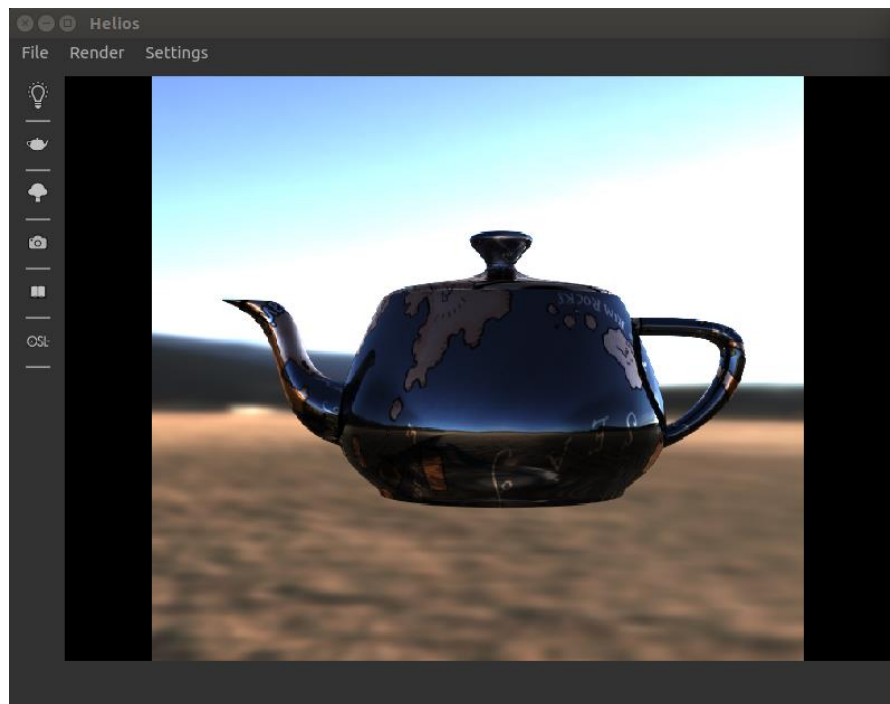


More complicated Node Graphs

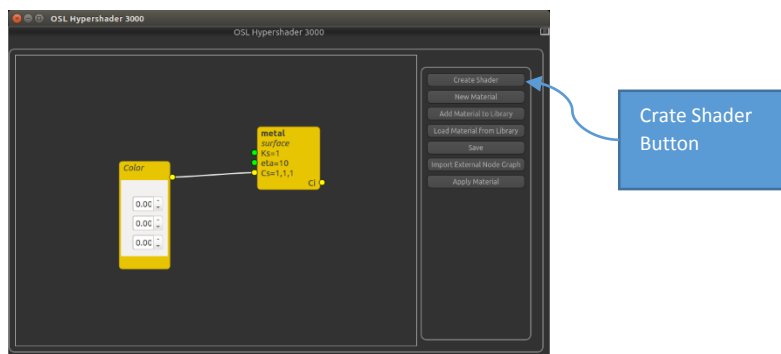
In shader creation you may want to connect multiple OSL shaders together. This is very much possible in our application,



As you can see I have used two shader nodes. One shader to convert a texture to a colour which is then plugged into our specular colour in our metal shader. This creates the following results,



However something you will need to keep in mind while creating shaders like this. Unlike connecting/disconnecting variable nodes if you change the order of connection of shader nodes you will need to re-compile your shader for the changes to be applied. This can be done with the “Create shader” button.



Tab Menu's

We have 3 tab menus within our application, this section will be a brief discussion of what the cover.

These 3 menus are located at the top of the application,



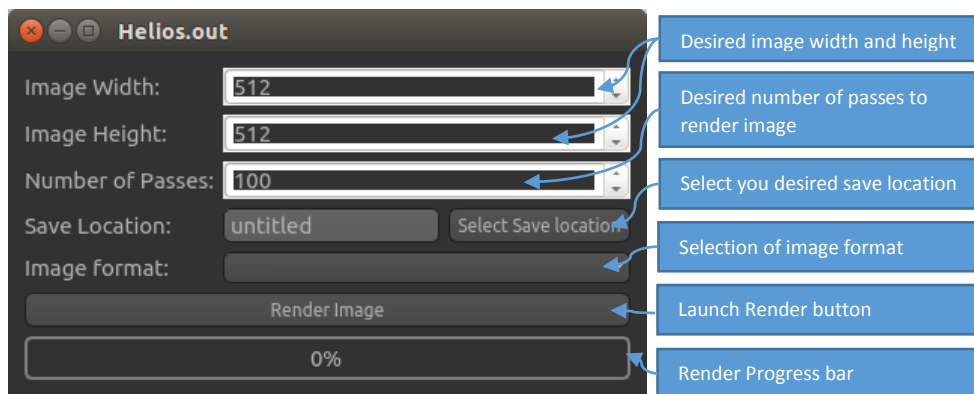
File

Under the file tab you will find 4 options,

- New scene
 - o Clears the scene
- Save Scene
 - o Saves the current scene and materials to a file of your choosing
 - o Note that all scene files are save with the “.sun” file extension
- Load Scene
 - o Loads a previously saved scene file to the path tracer
 - o Note that all scene files are saved with the “.sun” file extension
- Import
 - o Another short cut to import a mesh to the scene

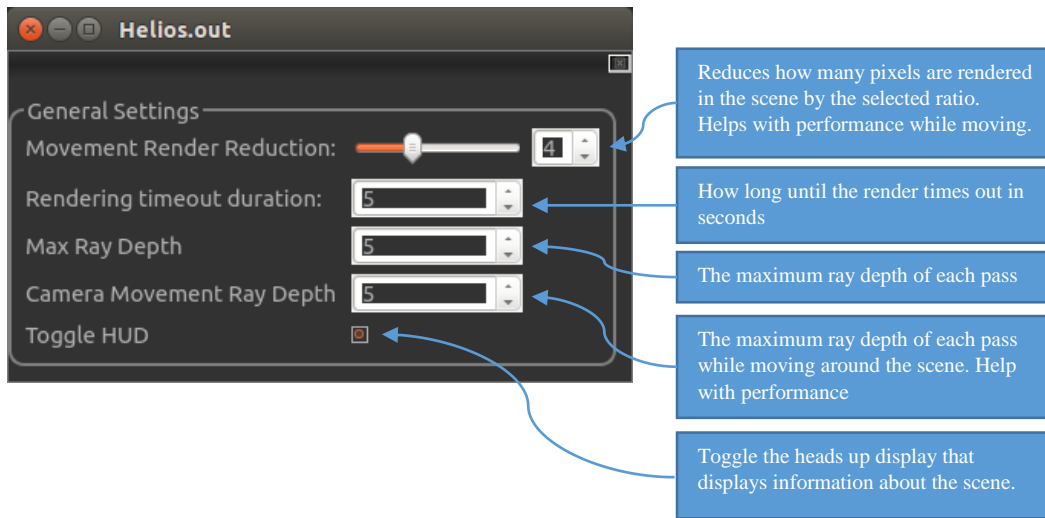
Render

Under the render tab you will find the “Image” button. Clicking this button will open the render to image menu. This allows you to render an image to a format of your choosing.



Settings

Under the settings tab you will find “General Settings”. This button will launch the general settings menu. Here you will be able to alter the general settings of the Helios application.



The screenshot shows the 'Helios.out' application window with the 'General Settings' tab selected. The settings are as follows:

- Movement Render Reduction:** A slider set to 4. Callout: Reduces how many pixels are rendered in the scene by the selected ratio. Helps with performance while moving.
- Rendering timeout duration:** A numeric input field set to 5. Callout: How long until the render times out in seconds.
- Max Ray Depth:** A numeric input field set to 5. Callout: The maximum ray depth of each pass.
- Camera Movement Ray Depth:** A numeric input field set to 5. Callout: The maximum ray depth of each pass while moving around the scene. Help with performance.
- Toggle HUD:** A checkbox that is currently checked. Callout: Toggle the heads up display that displays information about the scene.

OSL

Our renderer supports a wide variety of Open Shading Language features. The version we currently support is 1.0.

Types Supported

- Int
- *Float*
- String
- Color
- Matrix
- Normal
- Point
- Vector

Mathematical Operations

Multiplications

- Float * Float;
- Int * Int
- Float * Int
- Color * Color;
- Normal * Normal
- Point * Point
- Float * Color
- Float * Normal
- Float * Point
- Int * Color
- Int * Normal
- Int * Point

Additions

- Float + Float;
- Int + Int
- Float + Int
- Color + Color;
- Normal + Normal
- Point + Point

Divisions

- Float / Float;

- Int / Int
- Float / Int
- Int / Float
- Color / Color;
- Normal / Normal
- Point / Point
- Color / Float
- Normal / Float
- Point / Float
- Color / Int
- Normal / Int
- Point / Int

Subtractions

- Float - Float;
- Int - Int
- Float - Int
- Color - Color;
- Normal - Normal
- Point - Point

Cross product

vector **cross** (vector A, vector B) - Returns the cross product of two vectors (or normals), i.e., $A \times B$.

Dot Product

float **dot** (vector A, vector B) - Returns the inner product of the two vectors (or normals), i.e., $A \cdot B = A_x B_x + A_y B_y + A_z C_z$.

Length

float **length** (vector V)

float **length** (normal V)

Returns the length of a vector or normal.

Distance

float **distance** (point P0, point P1) - Returns the distance between two points.

Mix

type **mix** (*type* x, *type* y, *type* alpha)

type **mix** (*type* x, *type* y, float alpha)

The mix function returns a linear blending: $x * (1 - a) + y * (a)$

Power

type **pow** (*type* x, *type* y)

type **pow** (*type* x, float y)

Computes x^y . This function will return 0 for “undefined” operations, such as pow(-1,0.5).

Not Equal To

- Float != Float
- Float != Int
- Int != Int
- Int != Float
- Vector != Vector
- Color != Color
- Normal != Normal

Equal To

- Float == Float
- Float == Int
- Int == Int
- Int == Float
- Vector == Vector
- Color == Color
- Normal == Normal

Assignment

- Float = Float
- Float = Int
- Int = Int
- Int = Float
- Vector = Vector
- Color = Color
- Normal = Normal
- Vector = Float
- Vector = Int
- Color = Float
- Color = Int
- Point = Float
- Point = Int

Array Assignment

- Float[i] = Float
- Float[i] = Int
- Int[i] = Float;
- Int[i] = Int
- Color[i] = Colour
- Point[i] = Point
- Normal[i] = Normal

Referencing an Array

To reference the float array array[10] first component - array[0]

Floor

type **floor** (*type* x) - returns the largest integer less than or equal to x

Min

type **min** (*type* a, *type* b) - returns the minimum value either a or b

Max

type **max** (*type* a, *type* b) - returns the maximum value either a or b

Normalize

vector **normalize** (vector V)

normal **normalize** (normal V)

Return a vector in the same direction as V but with length 1, that is, $V / \text{length}(V)$

Less Than

Float < Float

Float < Int

Int < Int

Int < Float

Returns 1 if what's on the left side is less than what's on the right side 0 otherwise.

Greater Than

Float > Float

Float > Int

Int > Int

Int > Float

Returns 1 if what's on the left side is more than what's on the right side 0 otherwise.

Less Than or Equal To

Float <= Float

Float <= Int

Int <= Int

Int <= Float

Returns 1 if what's on the left side is less than or equal to what's on the right side 0 otherwise.

Greater Than or Equal To

Float >= Float

Float >= Int

Int >= Int

Int >= Float

Returns 1 if what's on the left side is more than or equal to what's on the right side 0 otherwise.

Syntax

For loops

Works in the same way as C++

```
for(int i=0; i<3; ++i){  
    ...  
}
```

If Statements

Conditionals in Open Shading Language just like in C or C++:

```
if ( condition ){  
    truestatement  
}
```

and


```

if ( condition ) {
    truestatement
}
else {
    falsestatement
}

```

Utilities

int **isnan** (float x) – function returns 1 if x is a not-a-number (NaN) value, 0 otherwise

int **isinf** (float x) – function returns 1 if x is an infinite (Inf or –Inf) value, 0 otherwise.

int **isfinite** (float x) - function returns 1 if x is an ordinary number (neither infinite nor NaN), 0 otherwise.

vector **random** () – function returns a uniform random vector with every component in the range of [0, 1]

int **backfacing** () - Returns 1 if the surface is being sampled as if “seen” from the back of the surface (or the “inside” of a closed object). Returns 0 if seen from the “front” or the “outside” of a closed object.

type **texture** (string filename, float s, float t) - Perform a texture lookup of an image file, indexed by 2D coordinates (s,t)

Closures

closure color diffuse(normal N)

Represents the Lambertian reflectance of a smooth surface, where *N* is a unit length forward-facing normal.

closure color phong(normal N, float exponent)

Represents the specular reflectance of the surface using the Phong BRDF. N is a unit length forward-facing normal. The *exponent* represents how smooth or rough the surface is. Higher values of exponent indicate a smoother surface.

closure color oren_nayar(normal N, float sigma)

Represents the diffuse reflectance of a rough surface using the Oren Nayar BRDF. N is a unit length forward-facing normal. The *sigma* parameter indicates how smooth or rough the surface is. 0 is perfectly smooth which is equivalent to the *diffuse(normal N)* BRDF.

closure color ward(normal N, vector T, float xrough, float yrough)

Represents the anisotropic specular reflectance of the surface using the Ward BRDF. N is a unit length forward-facing normal. The parameter T is a tangent to the surface. *xrough* and *yrough* are parameters to specify the amount of roughness in the tangent (T) and bi-tangent ($N * T$) directions respectively.

closure color microfacet_beckmann(normal N, float roughness, float eta)

Represents scattering on a surface using the Beckmann shadow-masking and distribution functions. N is a unit length forward-facing normal. The roughness parameter represents how rough or smooth the surface is. The *eta* parameter is the index of refraction of the material.

closure color microfacet_ggx(normal N, float roughness, float eta)

Represents scattering on a surface using the GGX shadow-masking and distribution functions. N is a unit length forward-facing normal. The roughness parameter represents how rough or smooth the surface is. The *eta* parameter is the index of refraction of the material.

closure color reflection(normal N, float eta)

Represents a sharp mirror-like reflection from the surface. N is a unit length forward-facing normal. *eta* is the index of refraction of the material.

closure color refraction(normal N, float eta)

Represents a sharp glass-like refraction of the objects behind the surface. N is a unit length forward-facing normal. *eta* is the index of refraction of the material.

closure color translucence(normal N)

Represents the Lambertian diffuse translucence of a smooth surface, which is much like `diffuse()` except that it gathers light from the *far* side of the surface. N is a unit length forward-facing normal.

Example shaders

```
surface metal
(
    float Ks = 1,
    float eta = 10,
    color Cs = 1,
)
{
    Ci = Ks * Cs * reflection(N, eta);
}
```

```
surface carPaint(
    color baseColour1 = color(0.08, 0.08, 0.77),
    color baseColour2 = color(0.46, 0.46, 0.82),
    color specularColour = color(0.87, 0.87, 1.0),
    float diffuseWeight = 0.5,
    float specularWeight = 0.3,
    float specularGlossiness = 0.5,
    float reflectionWeight = 0.2,
    float baseFalloffAmount = 2.0,
    float flakeSpread = 0.6,
    float e = 80.0
)
{
    float cosNI = dot(N, -I);
    float baseBlendFactor = pow(cosNI, baseFalloffAmount);

    color diffuseColour = mix(baseColour1, baseColour2, baseBlendFactor);
    float h = (reflectionWeight * (1.0 - cosNI));
    color reflectionColour = color(h, h, h);

    normal n = random() * 2.0 - 1.0;
    float cosFlake = dot(n, -I);
    if(cosFlake < 0.0){
        cosFlake = -cosFlake;
        n = -n;
    }

    n = normalize(N + flakeSpread * n);

    //diffuseColour = color(1, 1, 1);

    Ci = diffuseColour * diffuseWeight * diffuse(n) + specularColour * phong(n, e) +
    reflectionColour * reflection(N, 1.0);
}
```

```

surface glass
(
    float Ks = 0.15,
    float Cs = 0.80,
    color col = color(1, 1, 1),
    float eta = 1.5,
    int caustics = 0,
    int TIR = 0,
)
{
    if (caustics || !raytype("glossy") && !raytype("diffuse")) {
        // Take into account backfacing to invert eta accordingly
        if (backfacing()) {
            Ci = Cs * refraction(N, 1.0 / eta);
            // If Total Internal Reflection is enabled, we also return a
            // reflection closure, which might make rays bounce too much
            // inside an object. That's why we make it optional.
            //if (TIR){
            //    Ci += Ks * reflection(N, 1.0 / eta);
            //}
        }
        else {
            Ci = col * (Cs * refraction(N, eta) + Ks * reflection(N, eta));
        }
    }
}

```