



# Indigo Renderer Manual

For Indigo Renderer version 1.1.x

By Nicholas Chapman

Last updated 8th August 2008



Render by Bertrand Benoit

# Table of Contents

Indigo Renderer Manual.....	1
Indigo Overview.....	10
Running Indigo.....	10
Network Rendering.....	10
Progressive rendering.....	11
Command line parameter Reference.....	12
Scene XML format.....	14
renderer_settings.....	15
width.....	15
height.....	15
bih_tri_threshold.....	15
metropolis.....	15
large_mutation_prob.....	16
max_change.....	16
max_depth.....	16
max_num_consec_rejections.....	16
logging.....	17
bidirectional.....	17
save_untoneMapped_exr.....	17
save_toneMapped_exr.....	17
save_igi.....	17
image_save_period.....	17
halt_time.....	18
halt_samples_per_pixel.....	18
hybrid.....	18
frame_upload_period.....	18
auto_choose_num_threads.....	19
num_threads.....	19
super_sample_factor.....	19
display_period.....	19
ray_origin_nudge_distance.....	20
watermark.....	20
info_overlay.....	20
cache_trees.....	20
aperture_diffraction.....	20
post_process_diffraction.....	20
render_region.....	21
render_region::x1.....	21
render_region::y1.....	21
render_region::x2.....	21
render_region::y2.....	21
render_foreground_alpha.....	22
splat_filter.....	22
splat_filter::box.....	22
splat_filter::gaussian.....	22
splat_filter::mn_cubic.....	22
splat_filter::mn_cubic::blur.....	22
splat_filter::mn_cubic::ring.....	22
downsize_filter.....	23
background.....	24

skylight.....	25
sundir.....	25
turbidity.....	25
extra_atmospheric.....	25
env_map.....	27
env_map :: lat_long.....	27
env_map :: lat_long :: path.....	27
env_map :: lat_long :: gain.....	27
env_map :: spherical.....	28
env_map :: spherical :: path.....	28
env_map :: spherical :: width.....	28
env_map :: spherical :: gain.....	28
Tonemapping.....	30
tonemapping :: linear.....	30
tonemapping :: linear :: scale.....	30
tonemapping :: reinhard.....	30
tonemapping :: reinhard :: pre_scale.....	30
tonemapping :: reinhard :: post_scale.....	30
tonemapping :: reinhard :: burn.....	31
tonemapping :: camera.....	31
tonemapping :: camera :: response_function_path.....	31
tonemapping :: camera :: ev_adjust.....	31
tonemapping :: camera :: film_iso.....	32
Camera.....	33
pos.....	33
up.....	33
forwards.....	33
aperture_radius.....	33
focus_distance.....	34
aspect_ratio.....	34
sensor_width.....	34
lens_sensor_dist.....	34
white_balance.....	34
exposure_duration.....	35
autofocus.....	35
obstacle_map.....	35
aperture_shape.....	35
aperture_shape::circular.....	36
aperture_shape::image.....	36
aperture_shape::image::path.....	36
aperture_shape::generated.....	36
aperture_shape::generated::num_blades.....	36
aperture_shape::generated::start_angle.....	37
aperture_shape::generated::blade_offset.....	37
aperture_shape::generated::blade_curvature_radius.....	37
Camera element example XML: .....	38
Materials.....	40
Texture.....	40
texture::uv_set.....	40
texture::path.....	41
texture::exponent.....	41
texture::a.....	41

texture::b.....	41
texture::c.....	41
material.....	42
material :: name.....	42
diffuse.....	43
albedo.....	43
bump.....	43
displacement.....	43
base_emission.....	44
emission.....	44
layer.....	44
specular.....	46
internal_medium_name.....	46
transparent.....	46
bump.....	46
displacement.....	46
base_emission.....	47
emission.....	47
layer.....	47
phong.....	48
diffuse_albedo.....	48
ior.....	48
exponent.....	49
nk_data.....	49
specular_reflectivity.....	49
bump.....	50
displacement.....	50
base_emission.....	50
emission.....	50
layer.....	50
glossy_transparent.....	52
internal_medium_name.....	52
exponent.....	52
bump.....	52
displacement.....	52
base_emission.....	52
emission.....	52
layer.....	52
diffuse_transmitter.....	53
albedo.....	53
displacement.....	53
base_emission.....	53
emission.....	54
layer.....	54
blend.....	55
blend.....	55
step_blend.....	55
a_name.....	55
b_name.....	56
null_material.....	58
oren_nayar.....	59
sigma.....	59

albedo.....	60
bump.....	60
displacement.....	60
base_emission.....	60
emission.....	60
layer.....	60
Medium.....	61
medium.....	61
name.....	61
precedence.....	61
medium::epidermis.....	62
melanin_fraction .....	62
melanin_type_blend .....	62
medium::dermis.....	63
hemoglobin_fraction .....	63
medium::basic.....	64
ior.....	64
cauchy_b_coeff.....	64
absorption_coefficient_spectrum.....	64
subsurface_scattering.....	64
subsurface_scattering::scattering_coefficient_spectrum .....	65
subsurface_scattering:: phase_function .....	65
Phase Function.....	66
uniform.....	66
henyey_greenstein.....	66
henyey_greenstein::g_spectrum .....	66
Spectrum.....	67
spectrum::peak.....	67
spectrum::peak::peak_min.....	67
spectrum::peak::peak_width.....	67
spectrum::peak::base_value.....	67
spectrum::peak::peak_value.....	67
spectrum::blackbody.....	68
spectrum::blackbody::temperature .....	68
spectrum::blackbody::gain.....	68
spectrum::rgb.....	68
spectrum::rgb::rgb.....	68
spectrum::rgb::gamma.....	68
spectrum::uniform.....	69
spectrum::uniform::value.....	69
spectrum::regular_tabulated.....	69
spectrum::regular_tabulated::start_wavelength.....	69
spectrum::regular_tabulated::end_wavelength.....	69
spectrum::regular_tabulated::num_values.....	69
wavelength-dependent material parameter.....	71
constant.....	71
texture.....	71
texture::texture_index.....	71
shader.....	71
shader::shader.....	72
wavelength-independent material parameter.....	73
constant.....	73

texture.....	73
texture::texture_index.....	73
shader.....	73
shader::shader.....	73
rectanglelight.....	74
pos.....	74
width.....	74
height.....	74
spectrum.....	74
efficacy_scale.....	74
efficacy_scale::power_drawn .....	75
efficacy_scale::overall_luminous_efficiency .....	75
exit_portal.....	76
pos.....	76
scale.....	76
rotation.....	76
rotation :: matrix.....	77
mesh_name.....	77
meshlight.....	78
pos.....	78
scale.....	78
rotation.....	79
rotation :: matrix.....	79
mesh_name.....	79
spectrum.....	79
efficacy_scale.....	79
efficacy_scale::power_drawn .....	80
efficacy_scale::overall_luminous_efficiency .....	80
texture.....	80
ies_profile.....	80
ies_profile :: path.....	81
mesh.....	82
mesh :: name.....	82
mesh :: scale.....	82
mesh :: path.....	82
mesh :: normal_smoothing.....	82
mesh :: external.....	82
mesh :: external :: path.....	82
mesh :: embedded.....	82
mesh :: embedded :: expose_uv_set.....	82
mesh :: embedded :: expose_uv_set :: index.....	83
mesh :: embedded :: expose_uv_set :: name.....	83
mesh :: embedded :: vertex.....	83
mesh :: embedded :: vertex :: pos (attribute).....	83
mesh :: embedded :: vertex :: normal (attribute).....	83
mesh :: embedded :: vertex :: uvN (attribute).....	83
mesh :: embedded :: triangle_set.....	84
mesh :: embedded :: triangle_set :: material_name.....	84
mesh :: embedded :: triangle_set :: tri.....	84
model.....	86
pos.....	86
scale.....	86

rotation.....	86
rotation :: matrix.....	86
mesh_name.....	87
emission_scale.....	87
emission_scale::material_name.....	87
emission_scale::measure.....	87
emission_scale::value.....	87
plane.....	89
sphere.....	90
sphere :: center.....	90
sphere :: radius.....	90
sphere :: material_name.....	90
Include.....	91
include :: pathname.....	91
Indigo Shader Language Reference.....	92
Built-in types.....	92
real.....	92
int.....	92
bool.....	92
vec2.....	92
vec3.....	92
mat2x2.....	92
mat3x3.....	92
Literal values.....	92
Function Definitions.....	92
Built-in functions – Conditional functions.....	93
if(bool p, int a, int b) int.....	93
if(bool p, real a, real b) real.....	93
Built-in functions – Comparison functions.....	93
eq(int a, int b) bool.....	93
eq(real a, real b) bool.....	93
eq(bool a, bool b) bool.....	93
neq(int a, int b) bool.....	93
neq(real a, real b) bool.....	93
neq(bool a, bool b) bool.....	93
lt(int a, int b) bool.....	94
lt(real a, real b) bool.....	94
lte(int a, int b) bool.....	94
lte(real a, real b) bool.....	94
gt(int a, int b) bool.....	94
gt(real a, real b) bool.....	94
gte(int a, int b) bool.....	94
gte(real a, real b) bool.....	94
Built-in functions – Boolean functions.....	94
not(bool a) bool.....	94
or(bool a, bool b) bool.....	94
and(bool a, bool b) bool.....	94
Built-in functions – Maths utility functions.....	95
add(real x, real y) real.....	95
add(int x, int y) int.....	95
add(vec2 x, vec2 y) vec2.....	95
add(vec3 x, vec3 y) vec3.....	95

sub(real x, real y) real.....	95
sub(int x, int y) int.....	95
sub(vec2 x, vec2 y) vec2.....	95
sub(vec3 x, vec3 y) vec3.....	95
mul(real x, real y) real.....	95
mul(int x, int y) int.....	95
div(real x, real y) real.....	95
div(int x, int y) int.....	95
mod(real x, real y) real.....	95
mod(int x, int y) int.....	95
sin(real x) real.....	96
asin(real x) real.....	96
cos(real x) real.....	96
acos(real x) real.....	96
tan(real x) real.....	96
atan(real x) real.....	96
abs(real x) real.....	96
abs(int x) int.....	96
exp(real x) real.....	96
pow(real x, real y) real.....	96
sqrt(real x) real.....	96
log(real x) real.....	97
Built-in functions – Vector constructors.....	97
vec2(real x, real y) vec2.....	97
vec2(real x) vec2.....	97
vec3(real x, real y, real z) vec3.....	97
vec3(real x) vec3.....	97
Built-in functions – Vector functions.....	97
dot(vec2 a, vec2 b) real.....	97
dot(vec3 a, vec3 b) real.....	97
cross(vec3 a, vec3 b) vec3.....	97
neg(vec3 a) vec3.....	97
length(vec2 a) real.....	98
length(vec3 a) real.....	98
normalise(vec3 a) vec3.....	98
doti(vec3 a) real.....	98
dotj(vec3 a) real.....	98
dotk(vec3 a) real.....	98
doti(vec2 a) real.....	98
dotj(vec2 a) real.....	98
mul(vec2 a, real b) vec2.....	98
mul(vec3 a, real b) vec3.....	98
Built-in functions – Matrix constructors.....	98
mat2x2(real e11, real e12, real e21, real e22) mat2x2.....	98
mat3x3(real e11, real e12, real e13, real e21, real e22, real e23, real e31, real e32, real e33) mat3x3.....	99
Built-in functions – Matrix operations.....	99
mul(mat2x2 A, mat2x2 B) mat2x2.....	99
mul(mat3x3 A, mat3x3 B) mat3x3.....	99
mul(mat2x2 A, vec2 b) vec2.....	99
mul(mat3x3 A, vec3 b) vec3.....	99
transpose(mat2x2 A) mat2x2.....	99



transpose(mat3x3 A) mat3x3.....	99
inverse(mat2x2 A) mat2x2.....	99
inverse(mat3x3 A) mat3x3.....	99
Built-in functions – procedural noise functions.....	100
noise(real x) real.....	100
noise(vec2 x) real.....	100
noise(vec3 x) real.....	100
fbm(real x, int oc) real.....	100
fbm(vec2 x, int oc) real.....	100
fbm(vec3 x, int oc) real.....	100
Built-in functions – texture sampling functions.....	100
getTexCoords(int texcoord_set_index) vec2.....	100
sample2DTextureVec3(int texture_index, vec2 st) vec3.....	100
Appendix A: Modelling a Liquid in a Glass For Indigo.....	102
Appendix B: Scattering Properties for Liquids.....	105

## Indigo Overview

Indigo is a stand-alone application. Indigo renders one image at a time, reading the scene description from Indigo scene file (.igs), and writing the resulting image as a PNG file in the 'renders' directory. Indigo can be used in conjunction with a 3d-modelling package such as 3dsMax by using one of the exporters being coded by various people. The process in this case is

1. model scene in the 3d modelling package.
2. Use exporter script to export .3ds models and scene xml file to some directory.
3. Start indigo, specifying the path to the scene file to render.

Alternatively, the scene file can be hand-edited.

## Running Indigo

There are two executable files included in the Indigo distribution. Indigo.exe is the graphical user-interface (GUI) version of Indigo. Indigo\_console.exe is the command line, non-GUI version.

## Network Rendering

Indigo supports distributed rendering over a TCP/IP network. One Indigo process is started as a network master. Multiple Indigo processes (usually on other boxes) are then started as network slaves. The network slaves work on their own local version of the render, and periodically upload their buffer to the network master, where they are combined into the master render and saved to disk in the 'renders' directory as per usual.

The steps to run a network render are as follows:

1. Choose a scene to render, eg. somescene.xml.
2. Start the network master process like this:

```
indigo.exe somescene.xml -n m
```

The *-n m* switch tells the process to run as a network master

3. On another box, start a network slave process like this:

```
indigo.exe -n s -h lust:7777
```

The *-n s* switch runs indigo in network slave mode.

The *-h lust:7777* switch tells the slave to connect to the network master running on the host 'lust' on port 7777 (the indigo port)

You should of course substitute the host name running the network master for 'lust'.

The *-h* switch is optional. If it is not present, the local network is scanned for any masters, and if one is found, the slave connects to the master automatically.

## **Progressive rendering**

As a consequence of the unbiased nature of Indigo, the render will gradually converge over time to the correct solution. The longer you leave the render, the less noise will remain in the image. Indigo can be left to render a given scene for an arbitrary amount of time, and will never terminate by itself. Simply close Indigo when you are satisfied with the render (or you don't want to wait any longer).

## Command line parameter Reference

### **indigo [scenepathname]**

Starts Indigo. If *scenepathname* is present, then it will attempt to load that scene file.

### **-h hostname:port**

Sets the hostname and port that a network slave indigo process will try and connect to, e.g.

indigo -n s -h masterhostname:7777

### **-halt halttime**

Stops the Indigo process after *halttime* seconds. By default Indigo does not halt.

### **-haltspp X**

Stops the Indigo process after X samples per pixel have been reached.

### **-igio pathname**

Writes the Indigo Image (IGI) output to *pathname* instead of the usual generated pathname. Only used if IGI output is enabled.

### **-image\_save\_period N**

Sets the image save period to N seconds.

### **-frame\_upload\_period N**

Sets the frame upload period to N seconds

### **-n s**

Start in network render slave mode.

### **-n m**

Start in network render master mode.

### **-n wm**

Start in network render working master mode. Working master mode is like master mode, except rendering work is done on the master as well as the slaves.

### **-o pathname**

Writes the render to *pathname* instead of the usual generated pathname.

### **-p port**

Instructs a network render master to listen on a particular port.

**-r igi\_path**

Resume render using Indigo Image (.igi) found at *igi\_path*.

**-t numthreads**

Runs the Indigo process with *numthreads* threads.

**-texro**

Path to write the tone mapped EXR file to. Only used if tone mapped EXR output is enabled.

**-uexro**

Path to write the un-tone mapped EXR file to. Only used if un-tone mapped EXR output is enabled.

**--ptest**

Runs performance test for the given scene.

**--dumpmetadata png\_path**

Dump PNG meta data from file at *png\_path*

**--pack scene.igs out.pigs**

Pack Indigo scene *scene.igs* to *out.pigs*

**--unpack scene.pigs out**

Unpack Indigo PIGS or PIGM to the directory *out*

**--tonemap scene.igs in.igi out.png**

Tonemap *in.igi* using parameters in *scene.igs*, to *out.png*

## **Scene XML format**

Indigo scene files are stored in an XML format. The filename extension is .igs, for 'Indigo Scene'.

Root element should be called 'scene'.

## renderer\_settings

This element provides a means of overriding various settings from inifile.xml. This element is optional, and so are each of its children. Any setting defined here overrides the respective setting from inifile.xml.

**element status:** optional

### width

Sets the width (horizontal resolution) of the output image.

**type:** integer

**restrictions:** must be  $> 0$

**units:** pixels

**default value:** 600

### height

Sets the height (vertical resolution) of the output image.

**type:** integer

**restrictions:** must be  $> 0$

**units:** pixels

**default value:** 450

### bih\_tri\_threshold

If the number of triangles in a single mesh exceeds this threshold, then a BIH will be used for intersection acceleration for that mesh, otherwise a Kd-tree is used.

**type:** integer

**restrictions:** must be  $> 0$

**units:** number of triangles

**default value:** 1100000

### metropolis

Enables or disables Metropolis-Hastings sampling

**type:** boolean

**default value:** true

## large\_mutation\_prob

Probability of selecting a large mutation type. Only used if metropolis is true.

**type:** scalar real

**restrictions:** must in range [0, 1]

**units:** dimensionless

**default value:** 0.4

## max\_change

Radius of the perturbation mutation distribution.

**type:** scalar real

**restrictions:** must in range [0, 1]

**units:** dimensionless

**default value:** 0.01

## max\_depth

Maximum ray bounce depth.

**type:** integer

**restrictions:** must be  $> 0$

**units:** number of bounces

**default value:** 10000

## max\_num\_consec\_rejections

Maximum number of consecutive rejection of tentative new samples when Metropolis-Hastings transport is used. Note that any non-infinite number technically causes biased sampling.

**type:** integer

**restrictions:** must be  $> 0$

**units:** number of rejections



**default value:** 1000

## logging

If true, a log of the console output is written to log.txt

**type:** boolean

**default value:** true

## bidirectional

If true, bidirectional path tracing is used to construct paths. Otherwise, backwards path tracing is used.

**type:** boolean

**default value:** true

## save\_untoneMapped\_exr

If true, an untoneMapped EXR image is saved in the renders directory.

**type:** boolean

**default value:** false

## save\_toneMapped\_exr

If true, a toneMapped EXR image is saved in the renders directory.

**type:** boolean

**default value:** false

## save\_igi

If true, an untoneMapped Indigo Image (.igi) file is saved in the renders directory.

**type:** boolean

**default value:** false

## image\_save\_period

The rendered image(s) will be saved to the renders directory every *image\_save\_period* seconds.

**type:** scalar real

**restrictions:** must be  $> 0$

**units:** seconds

**default value:** 10

## halt\_time

If positive, indigo will halt after *halt\_time* seconds.

**type:** scalar real

**restrictions:**

**units:** seconds

**default value:** -1

## halt\_samples\_per\_pixel

If positive, indigo will halt after *halt\_samples\_per\_pixel* samples per pixel have been reached.

**type:** scalar real

**restrictions:**

**units:** samples / pixel

**default value:** -1

## hybrid

If true, direct illumination is sampled with QMC sampling, and indirect illumination with Metropolis-Hastings sampling.

**type:** boolean

**default value:** false

## frame\_upload\_period

Period between uploads of the image buffer from slave to master when rendering in network mode.

**type:** scalar real

**restrictions:** must be  $> 0$

**units:** seconds

**default value:** 40

## **auto\_choose\_num\_threads**

If true, the number of render threads used is set based on the number of logical cores detected.

**type:** boolean

**default value:** true

## **num\_threads**

Number of render threads used. This setting is only used if *auto\_choose\_num\_threads* is false.

**type:** integer

**restrictions:** must be > 0

**units:** number of threads

**default value:** 1

## **super\_sample\_factor**

If this factor is greater than 1, then the image is rendered at a higher resolution internally, then downsampled using the downsize filter before the render is saved to disk. This can help to reduce aliasing around high contrast edges.

Note that higher factors require more memory (RAM).

**type:** integer

**restrictions:** must be > 0

**units:** dimensionless

**default value:** 2

## **display\_period**

The internal HDR buffer is tonemapped and displayed on screen every *display\_period* seconds.

**type:** scalar real

**restrictions:** must be > 0

**units:** seconds

**default value:** 10

## **ray\_origin\_nudge\_distance**

Ray origins are offset by this distance after intersection, in order to avoid false self-intersections.

**type:** scalar real

**restrictions:** must be  $\geq 0$

**units:** meters

**default value:** 1.0e-4

## **watermark**

If true, an 'Indigo Renderer' logo is drawn on the bottom right hand corner of the output render.

**type:** boolean

**default value:** false

## **info\_overlay**

If true, a line of text is drawn on the bottom of each render, containing some statistics about the current render process.

**type:** boolean

**default value:** false

## **cache\_trees**

If true, kd-trees are cached to disk after construction, in the tree\_cache directory.

**type:** boolean

**default value:** true

## **aperture\_diffraction**

If true, diffraction of light passing through the camera aperture is simulated.

**type:** boolean

**default value:** true

## **post\_process\_diffraction**

If true, aperture\_diffraction is simulated using a filter applied to the image buffer, instead of perturbation of rays. This technique is generally faster and less noisy, but slightly less accurate.

**type:** boolean

**default value:** true

## **render\_region**

If this element is present, only a certain region of the usual image is rendered.

Only pixels (x, y) such that  $x_1 \leq x < x_2$  and  $y_1 \leq y < y_2$  are rendered.

## **render\_region::x1**

X Coordinate of top left pixel of rendered region.

**type:** integer

**restrictions:** must be  $\geq 0$

**units:** pixels

## **render\_region::y1**

Y Coordinate of top left pixel of rendered region.

**type:** integer

**restrictions:** must be  $\geq 0$

**units:** pixels

## **render\_region::x2**

X Coordinate of pixel immediately to the right of rendered region.

**type:** integer

**restrictions:**  $x_1 < x_2 \leq \text{width}$

**units:** pixels

## **render\_region::y2**

Y Coordinate of pixel immediately below rendered region.

**type:** integer

**restrictions:**  $y_1 < y_2 \leq \text{height}$

**units:** pixels

## **render\_foreground\_alpha**

If this is true, the output image is just a greyscale image, where the foreground is white, and the background (physical sky, env map, constant background, void background etc..) is black.

**type:** boolean

**default value:** false

## **splat\_filter**

Controls the filter used for splatting contributions to the image buffer.

Can be one of *box*, *gaussian*, or *mn\_cubic*.

### **splat\_filter::box**

Box filter. Causes bad aliasing, don't use :)

### **splat\_filter::gaussian**

Gaussian filter with standard deviation of 0.35 pixels.

### **splat\_filter::mn\_cubic**

Mitchell-Netravali cubic filter. Good all-round filter with little aliasing.

Please refer to the paper 'Reconstruction Filters in Computer Graphics' by Mitchell and Netravali, 1988, for more information.

### **splat\_filter::mn\_cubic::blur**

The 'B' parameter from the paper. Higher blur values cause more blurring of the image

**type:** scalar real

**restrictions:** will give best results in range [0, 1]

**units:** dimensionless

**default value:** 0.6

### **splat\_filter::mn\_cubic::ring**

The 'C' parameter from the paper. Higher ring values cause more 'ringing'. (alternating bands of black and white around high contrast edges).

Note that Mitchell and Netravali recommend choosing B and C such that  $2C + B = 1$ .

**type:** scalar real

**restrictions:** will give best results in range [0, 1]

**units:** dimensionless

**default value:** 0.2

## downsize\_filter

Controls the filter used for downsizing super-sampled images.

Only used when `super_sample_factor` is greater than one.

Takes exactly the same parameters as *splat\_filter*.

Example XML for `renderer_settings`:

```
<renderer_settings>
  <metropolis>true</metropolis>
  <bidirectional>true</bidirectional>

  <width>800</width>
  <height>600</height>

  <downsize_filter>
    <mn_cubic>
      <ring>0.2</ring>
      <blur>0.6</blur>
    </mn_cubic>
  </downsize_filter>
  <splat_filter>
    <gaussian/>
  </splat_filter>

  <super_sample_factor>2</super_sample_factor>

  <aperture_diffraction>true</aperture_diffraction>
  <post_process_diffraction>true</post_process_diffraction>
</renderer_settings>
```

## background

Illuminates scene with a uniform environment light.

*element status:* optional

Must have exactly one 'spectrum' child element.

example xml:

```
<background>
  <spectrum>
    <blackbody>
      <temperature>3500</temperature>
      <gain>1.0</gain>
    </blackbody>
  </spectrum>
</background>
```



## skylight



Illuminates scene with sunlight and scattered skylight.

*element status:* optional

## sundir

The *sundir* element defines the 3-vector direction towards the sun. the Z axis is up, e.g. (0,0,1) places the sun directly overhead. Need not be normalised

**type:** *real 3-vector*

**restrictions:** *z component must be  $> 0$*

**units:** *dimensionless*

## turbidity

The *turbidity* defines the haziness/clearness of the sky. Lower turbidity means a clearer sky. Should be set to something between 2 and ~5.

**type:** *scalar real*

**restrictions:**  *$> 0$*

**units:** *dimensionless*

## extra\_atmospheric

If *extra\_atmospheric* is true, then the skylight is computed as if it was outside the atmosphere.

This means that the sun spectrum is not attenuated by atmospheric scattering, and the sky will be black, since

there is no atmospheric scattering.

**Element status:** optional

**type:** boolean

**default:**false

xml example:

```
<skylight>  
  <sundir>0 0.6 1</sundir>  
  <turbidity>2</turbidity>  
</skylight>
```

## env\_map

*element status:* optional



Illuminates scene with a HDR environment map.

Currently Indigo can load two types of environment maps.

The first type is .exr maps in lat-long format:

**env\_map :: lat\_long**

**env\_map :: lat\_long :: path**

The path to the .exr file.

**type:** *string*

**restrictions:** *must be a valid path*

**units:**

**env\_map :: lat\_long :: gain**

The map is scaled by this factor when it is loaded.

**type:** *scalar real*

**restrictions:** *> 0*

**units:** *dimensionless*

The second type of Env map supported by Indigo is .float maps in spherical format. .float is a simple format exported by the HDR Shop program, with 3 32bit floats per pixel, one per colour channel, and no other information in the file.

### **env\_map :: spherical**

#### **env\_map :: spherical :: path**

The path to the .exr file.

**type:** *string*

**restrictions:** *must be a valid path*

**units:**

#### **env\_map :: spherical :: width**

The width of the map. Must be equal to the height

**type:** *scalar integer*

**restrictions:** *> 0*

**units:** *pixels*

#### **env\_map :: spherical :: gain**

The map is scaled by this factor when it is loaded.

**type:** *scalar real*

**restrictions:** *> 0*

**units:** *dimensionless*

Example XML:

```
<env_map>
  <spherical>
    <path>probes/kitchen_probe.float</path>
    <width>640</width>
    <gain>0.9</gain>
  </spherical>
</env_map>
```

```
<!--latlong>  
  <path>probes/skylight-day.exr</path>  
  <gain>1.0</gain>  
</latlong-->  
</env_map>
```

# Tonemapping

The tonemapping element should have one child element, either 'linear', 'reinhard', or 'camera'.

*element status:* required

## tonemapping :: linear

### tonemapping :: linear :: scale

A constant by which the pixel values are multiplied.

**type:** *scalar real*

**restrictions:**  $\geq 0$

**units:** *dimensionless*

## tonemapping :: reinhard

### tonemapping :: reinhard :: pre\_scale

The pixel buffer is scaled by this factor before the non-linear stage of the tonemapping takes place. Stands in for the middle-grey luminance scaling in part 3.1 of Reinhard et. al.'s [Photographic Tone Reproduction for Digital Images](#) paper.

**type:** *scalar real*

**restrictions:**  $\geq 0$

**units:** *dimensionless*

### tonemapping :: reinhard :: post\_scale

This scaling factor is applied after the rest of the tone mapping stages. By default, the pixel with max luminance is mapped to white, so setting this scale to  $> 1$  will result in pixels with less luminance being mapped to white. example:

**type:** *scalar real*

**restrictions:**  $\geq 0$

**units:** *dimensionless*

## **tonemapping :: reinhard :: burn**

Determines the luminance at which clipping occurs.

A smaller value means more severe burn, no burn will occur in the limit as the value goes to infinity.

*element status:* optional

**type:** *scalar real*

**restrictions:**  $> 1$

**units:** *dimensionless*

**default value:** *10*

## **tonemapping :: camera**

Camera tone mapping is an attempt to model the image generation process of a digital camera, and shares some parameters with a real camera.

The response function uses data from

<http://www1.cs.columbia.edu/CAVE/software/softlib/dorf.php>

, as such many cameras should be able to be modelled.

## **tonemapping :: camera :: response\_function\_path**

Path to response function data file, e.g. 'data/camera\_response\_functions/dscs315.txt'

Path can be absolute or relative; if relative, it is taken relative to the Indigo executable base path.

**type:** string

**restrictions:**

**units:**

## **tonemapping :: camera :: ev\_adjust**

ev\_adjust is exposure-value adjustment; increasing this value by 1 will effectively double the 'sensor output'.

**type:** real scalar

**restrictions:**

**units:** *dimensionless*

## tonemapping :: camera :: film\_iso

film speed (film ISO) has much the same effect as `ev_adjust`, except it's a linear factor. Doubling the film ISO will double the 'sensor output'.

**type:** real scalar

**restrictions:** *must be > 0*

**units:** *dimensionless*

xml example:

```
<tonemapping>
  <reinhard>
    <pre_scale>1.0</pre_scale>
    <post_scale>1.0</post_scale>
  </reinhard>
</tonemapping>
```



# Camera

**element status:** required

## pos

Defines the position of the camera.

**type:** real 3-vector

**restrictions:**

**units:** meters

## up

Defines the up vector of the camera. This and the forwards vector uniquely determine the right vector. Need not be normalised.

*type:* real 3-vector

**restrictions:**

**units:** *dimensionless*

## forwards

Defines the forwards vector of the camera, i.e. which direction it is facing. Need not be normalised.

*type:* real 3-vector

**restrictions:**

**units:** *dimensionless*

## aperture\_radius

Defines the radius of the camera aperture. Larger radius means more depth of field.

If a non-circular aperture is used, then aperture\_radius defines the half-width of the rectangle in which the aperture shape is defined.

**type:** scalar real

**restrictions:** Must be greater than zero.

**units:** meters

## focus\_distance

Distance from the camera, along the camera forwards direction, to the focal plane. Objects lying on the focal plane will be in focus. Value not used if autofocus is set.

**type:** scalar real

**restrictions:** Must be greater than zero.

**units:** meters

## aspect\_ratio

Influences the directions in which rays are traced. Should be set to the image width divided by the image height.

**type:** scalar real

**restrictions:** Must be greater than zero.

**units:** dimensionless

## sensor\_width

Width of the sensor element of the camera. A reasonable default is 0.036. (36mm)

Determines the angle of view (FOV), together with the lens\_sensor\_dist.

**type:** scalar real

**restrictions:** Must be greater than zero.

**units:** meters

## lens\_sensor\_dist

Distance from the camera sensor to the camera lens. A reasonable default is 0.02. (20mm)

**type:** scalar real

**restrictions:** Must be greater than zero.

**units:** meters

## white\_balance

Sets the white balance of the camera.

Possible values are D50, D55, D65 etc..

all the illuminants from [http://en.wikipedia.org/wiki/White\\_point](http://en.wikipedia.org/wiki/White_point) are supported.

What's this for?

Well lets say you're rendering a room illuminated by a 5000K blackbody emitter.

In real life, your eyes would adjust to the lighting conditions, and you would perceive the light as white.

The same would occur in a room lit by a 6500K blackbody emitter.

A whitebalance setting allows the camera to adjust in the same way that the eyes do.

So if you set the white balance to D50 and render the room with a 5000K emitter, the light should appear white.

If you set the white balance to D65 it will come out kinda orange.

The D65 white point is designed for outdoors and is a good general setting to use if you're not sure what to use.

**type:** string

**restrictions:** Must be one of 'D65', 'D50', 'E' etc..

## exposure\_duration

How long the exposure will be. The longer the exposure duration, the greater the light energy registered by the sensor.

**type:** scalar real

**restrictions:** Must be greater than zero.

**units:** seconds

## autofocus

If this (empty) element is present, a ray will be traced from the camera position in the camera forwards direction. The camera focus distance will then be set to the distance the ray travels before striking an object, or to infinity if no object is hit.

Element status: optional

## obstacle\_map

If this element is present, then an obstacle map texture is used when calculating the diffraction though the camera aperture.

An obstacle map will only have an effect if aperture\_diffraction is enabled.

Path must be relative to the scene working directory.

**Element status:** optional

**type:** string

## aperture\_shape

This element allows a particular shape of camera aperture to be specified. The allowable shapes are *image*,

*generated*, or *circular*.

If the `aperture_shape` element is not present, then a default circular aperture shape is used.

Note that a preview of the final aperture shape will be saved in the working directory as *aperture\_preview.png*.

**Element status:** optional

### **aperture\_shape::circular**

Makes the camera use a circular shaped aperture.

### **aperture\_shape::image**

Allows the aperture shape to be loaded from an image file.

### **aperture\_shape::image::path**

The path to the aperture image file.

The image must be of PNG format.

The image is interpreted as a greyscale image.

The image must be square, and have power-of-two dimensions of at least 512 x 512.

White portions of the image are interpreted as transparent, and black parts of the image are interpreted as stopping light.

The white part of the aperture image should be as large as possible (i.e. It should just touch the edges of the square image), to allow for efficient sampling.

Path must be relative to the scene working directory.

**type:** string

### **aperture\_shape::generated**

Allows the aperture shape to be defined using a few parameters. See the attached diagram for more information.

### **aperture\_shape::generated::num\_blades**

Number of diaphragm blades.

**type:** integer

**restrictions:** Must be  $\geq 3$

**units:** dimensionless

### **aperture\_shape::generated::start\_angle**

Initial angle of first diaphragm blade.

**type:** scalar real

**restrictions:**

**units:** radians

### **aperture\_shape::generated::blade\_offset**

Distance from center of aperture shape to edge of diaphragm.

**type:** scalar real

**restrictions:** must be  $> 0$

**units:** fraction of aperture shape width

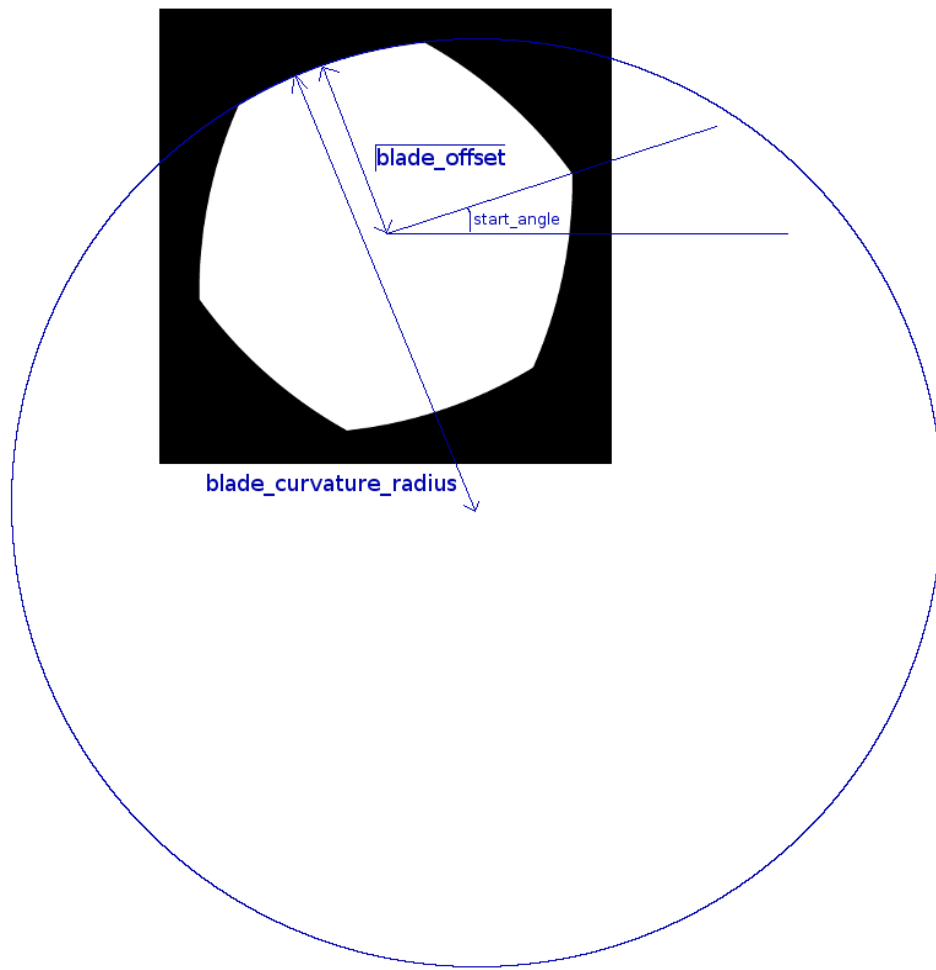
### **aperture\_shape::generated::blade\_curvature\_radius**

Distance from edge of diaphragm to effective center of diaphragm curvature circle.

**type:** scalar real

**restrictions:** must be  $> 0$

**units:** fraction of aperture shape width



### Camera element example XML:

```
<camera>
  <pos>0 -2 1</pos>
  <up>0 0 1</up>
  <forwards>0 1 0</forwards>
  <aperture_radius>0.001</aperture_radius>
  <focus_distance>3.0</focus_distance>
  <aspect_ratio>1.33</aspect_ratio>
  <sensor_width>0.036</sensor_width>
  <lens_sensor_dist>0.02</lens_sensor_dist>
  <white_balance>E</white_balance>

  <autofocus/>
</camera>
```



# Materials

## Texture

The final, used value for each component is calculated like so:

$$f(x) = a * g(x)^2 + b * g(x) + c$$

and

$$g(x) = x^{\text{exponent}}$$

Where  $x$  is the colour component from the map (e.g. The greyscale value or one of the R, G, B), normalised to  $[0, 1]$ ,

$f(x)$  is the final used value, and  $a$ ,  $b$ ,  $c$ , and *exponent* are as described below.

In the case where a scalar value is required from a RGB map, for example when using a bump map, the final value is calculated as  $(f(r) + f(g) + f(b)) / 3$

### Supported texture formats:

**JPEG (.jpg, .jpeg):** greyscale, RGB supported.

**PNG (.png):** greyscale (8 or 16 bits per pixel) , RGB (24 or 48 bits per pixel) supported. Note that 16 bits per channel data will be internally converted to 8 bits per channel data.

**Truevision TGA (.tga):** greyscale (8 bits per pixel) or RGB (24 bits per pixel) supported. RLE compression is not supported.

**Windows Bitmap (.bmp):** greyscale (8 bits per pixel) or RGB (24 bits per pixel) supported. RLE compression is not supported.

**Open EXR (.exr)**

## texture::uv\_set

Name of the set of uv coordinates used for texture lookup.

**type:** string

**restrictions:** must be a uv set that has been exported by a mesh, if the material is used on that mesh.

**units:**



## **texture::path**

Path to the texture on disk. Path must be absolute or relative to the scene working directory.

**type:** string

## **texture::exponent**

Used for converting texture RGB values to display values. A typical value is thus 2.2.

**type:** scalar real

**restrictions:** must be  $> 0$

**units:** dimensionless

## **texture::a**

'a' coefficient for quadratic texture function.

Element status: optional

## **texture::b**

'b' coefficient for quadratic texture function.

Element status: optional

## **texture::c**

'c' coefficient for quadratic texture function.

Element status: optional

# material

Defines a material.

A material element must have one child element called 'name', and another element which can be either 'specular', 'phong', or 'diffuse' etc..

## material :: name

The name of the material

**type:** string

**restrictions:**

**units:**

example xml:

```
<material>
  <name>white</name>
  <diffuse>
    <colour>0.9 0.9 0.9</colour>
  </diffuse>
</material>
```

## diffuse



Diffuse is a Lambertian diffuse material.

### albedo

Sets the reflectance (albedo)

Values will be clamped to the range [0, 1].

**Type:** wavelength-dependent material parameter.

**units:** dimensionless

### bump

The bump map is used to perturb the shading normal of the surface.

**element\_status:** optional

**type:** wavelength-independent material parameter.

**units:** meters

### displacement

The displacement distance in meters that mesh vertices are displaced along the shading normal.

**element\_status:** optional

**type:** wavelength-independent material parameter.

**units:** meters

## base\_emission

The spectral radiance emitted from this material surface.

Values will be clamped to [0, infinity)

**element\_status:** optional

**type:** wavelength-dependent material parameter.

**units:**  $\text{W m}^{-3} \text{sr}^{-1}$

## emission

Modulates the base\_emission value: the final emitted spectral radiance is calculated as the product of base\_emission and emission.

Values will be clamped to [0, infinity)

**element\_status:** optional

**type:** wavelength-dependent material parameter.

**Units:** dimensionless

## layer

Defines the index of the layer that light emitted from this material will be drawn to. The index is zero-based.

**element\_status:** optional

**type:** integer

**default:** 0

example xml:

```
<material>
  <name>mat1</name>
  <diffuse>
    <texture>
      <uv_set>albedo</uv_set>
      <path>indigo_logo.png</path>
      <exponent>2.2</exponent>
    </texture>
  </diffuse>
</material>
```

```
                </texture>

                <base_emission>
                    <constant>
                        <rgb>
                            <rgb>
                                20000000000 20000000000
20000000000
                            </rgb>
                        <gamma>1</gamma>
                    </constant>
                </base_emission>
            </diffuse>
        </material>
```

## specular



Specular is a material that can be both a perfect specular reflector and a perfect specular transmitter.

### internal\_medium\_name

Should be the name of a medium already defined in the scene file.

*type:* string

*unit:*

*restrictions:*

### transparent

true or false. If true, light can be transmitted, if not, only reflected light is simulated.

*type:* boolean

### bump

See `diffuse::bump`

### displacement

See `diffuse::displacement`

## **base\_emission**

See `diffuse::base_emission`

## **emission**

See `diffuse::emission`

## **layer**

See `diffuse::layer`

xml example:

```
<specular>
  <transparent>true</transparent>
  <internal_medium_name>glass</internal_medium_name>
</specular>
```

## phong



*Image by Xman*

Phong is a physically based glossy reflection model using a Phong lobe. It has a Lambertian diffuse substrate.

## diffuse\_albedo

The reflectance of the diffuse substrate.

Values will be clamped to the range [0, 1].

**type:** wavelength-dependent material parameter

**units:** dimensionless

## ior

Index of refraction of the dielectric coating or substance making up the material.

**type:** real scalar

**units:** dimensionless

**restrictions:**  $\geq 1$



## exponent

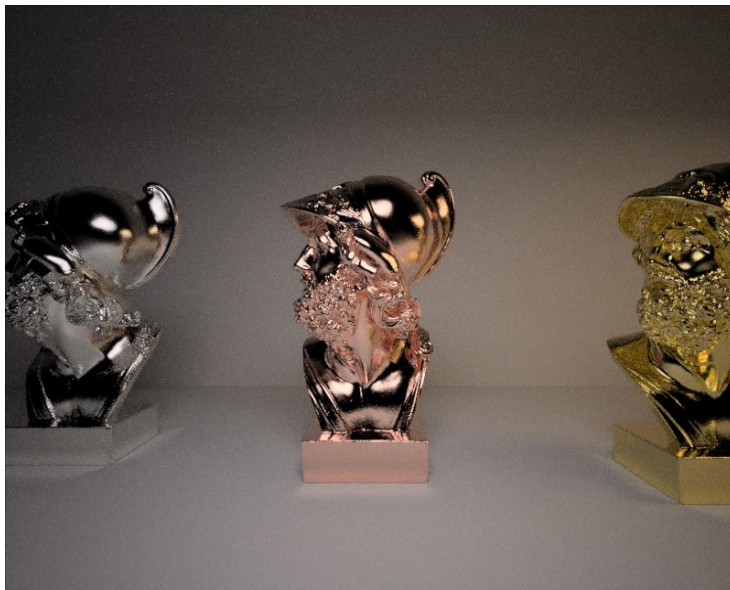
Sets the exponent of the Phong lobe controlling specular reflection. Higher exponent means more 'perfect' reflection, lower exponent leads to more glossy+diffuse highlights. Can range from ~1 to up to 10000 or more.

**type:** real scalar

*units:* dimensionless

**restrictions:** Must be greater than zero.

## nk\_data



The nk\_data element specifies that the phong material should use measured complex IOR data to compute the reflection at various angles.

If nk\_data is specified, then the diffuse and specular elements are not taken into account.

The value of the nk\_data element should be a path to a .nk file, e.g. 'nkdata/au.nk'

**element status:** optional

**type:** string

**restrictions:** Must be a valid path to a .nk file. Path should be a relative path from Indigo root directory.

## specular\_reflectivity

The specular reflectivity at normal incidence.

If this element is present, then the specular reflectivity at various angles is based on these values.

If this element is present, then the diffuse albedo of the material is set to zero, therefore, if this element is present, only metals can be simulated.

If this element is present, then the *ior* and *diffuse\_albedo* elements are ignored.

Values will be clamped to lie in the range [0, 1]

**type:** wavelength-dependent material parameter

**units:** dimensionless

## bump

See `diffuse::bump`

## displacement

See `diffuse::displacement`

## base\_emission

See `diffuse::base_emission`

## emission

See `diffuse::emission`

## layer

See `diffuse::layer`

xml example:

```
<material>
  <name>phong1</name>
  <phong>
    <ior>1.5</ior>
    <exponent>
      <constant>
        10.0
      </constant>
    </exponent>
    <diffuse_albedo>
      <constant>
        <uniform>
          <value>0.3</value>
        </uniform>
      </constant>
    </diffuse_albedo>
  </phong>
</material>
or
<material>
  <name>aluminium</name>
  <phong>
    <nk_data>nkdata/al.nk</nk_data>
    <exponent>
```

```
                <constant>1000</constant>
            </exponent>
        </phong>
</material>

or

<material>

    <name>phong2</name>

    <phong>

        <exponent>

            <constant>100</constant>

        </exponent>

        <specular_reflectivity>

            <constant>

                <rgb>

                    <rgb>0.8 0.2 0.2</rgb>

                    <gamma>1</gamma>

                </rgb>

            </constant>

        </specular_reflectivity>

    </phong>

</material>
```

## glossy\_transparent

The glossy transparent is a material that simulates a rough surface of a transparent dielectric medium. It's good for simulating stuff like frosted glass, human skin etc...

### internal\_medium\_name

Should be the name of a medium already defined in the scene file.

**type:** string

### exponent

See phong::exponent

### bump

See diffuse::bump

### displacement

See diffuse::displacement

### base\_emission

See diffuse::base\_emission

### emission

See diffuse::emission

### layer

See diffuse::layer

XML example:

```
<material>
  <name>frosty_glass</name>

  <glossy_transparent>
    <internal_medium_name>glass</internal_medium_name>
    <exponent>
      <constant>1000</constant>
    </exponent>
  </glossy_transparent>
</material>
```

## diffuse\_transmitter



This material is a very simple BSDF that basically scatters incoming light into the opposite hemisphere, with a cosine weighted distribution.

Although it doesn't really have any exact physical basis, it could be thought of as the limit of many sub-surface scatters inside a thin, highly scattering material. As such it should be useful for simulating such materials as curtains, lampshades etc..

It's meant to be used on single-layer geometry, and it does not have an associated internal medium (it's not an interface material).

It will probably be a good idea to blend this material with a normal diffuse material, so that some backscattered light is visible, not just transmitted light.

### albedo

Sets the transmission fraction (albedo)

Values will be clamped to the range [0, 1.0].

**Type:** wavelength-dependedent material parameter

**units:** dimensionless

### displacement

See `diffuse::displacement`

### base\_emission

See `diffuse::base_emission`

## emission

See `diffuse::emission`

## layer

See `diffuse::layer`

xml example:

```
<material>
  <name>diff_tran</name>

  <diffuse_transmitter>
    <texture>
      <uv_set>albedo</uv_set>
      <path>ColorChecker_sRGB_from_Ref.jpg</path>
      <exponent>2.2</exponent>
    </texture>

    <albedo>
      <texture>
        <texture_index>0</texture_index>
      </texture>
    </albedo>
  </diffuse_transmitter>
</material>
```

## blend

The blend material allows two materials to be blended together, with the weighting factor a given constant, or controlled by an image map.

More than two materials can be blended, by using a hierarchical arrangement of blend materials.

There is one restriction that applies to what materials can be blended together (in the same blend tree composed of one or more blend materials) – At most one constituent material can be a BSDF containing a delta distribution. Materials with delta distributions are the specular and null\_material material types.

### blend

Controls the fraction of each constituent material used.

A value of 0 means only material *a* is used, a value of 1 means only material *b* is used.

Will be clamped to [0, 1]

**type:** wavelength-independent material parameter

**units:** dimensionless

### step\_blend

If step\_blend is true, then the blend parameter will have the following step function applied to it:

$f(x) = 1$  if  $x \geq 0.5$ , else 0

Enabling step blend is recommended when using a texture as a 'clip mask', in order to reduce noise.

**type:** boolean

**default:** false

### a\_name

Name of constituent material *a*.

**type:** string

**restrictions:** must be the name of a material already defined.

## **b\_name**

Name of constituent material *b*.

**type:** string

**restrictions:** must be the name of a material already defined.

xml example:

```
<material>
  <name>mata</name>
  <phong>
    <ior>3</ior>
    <diffuse>1 0 0</diffuse>
    <exponent>10000</exponent>

    <bump_map>
      <uv_set>bump</uv_set>
      <path>indigo.jpg</path>
      <b>0.003</b>
      <exponent>1.0</exponent>
    </bump_map>
  </phong>
</material>

<material>
  <name>matb</name>
  <diffuse>
    <colour>0 1 0</colour>

    <bump_map>
      <uv_set>bump</uv_set>
      <path>spherebump.jpg</path>
      <b>0.1</b>
      <exponent>1.0</exponent>
    </bump_map>
  </diffuse>
</material>

<material>
  <name>blendmat</name>

  <blend>
    <a_name>a</a_name>
    <b_name>b</b_name>
    <texture>
      <uv_set>albedo</uv_set>
      <path>checker.jpg</path>
      <exponent>1.0</exponent>
    </texture>
    <blend>
      <texture>
        <texture_index>0</texture_index>
      </texture>
    </blend>
  </blend>
</material>
```





## **null\_material**

The null material is a very simple material that doesn't scatter light at all. It's effectively invisible.

The null material has no parameters.

Xml example:

```
<material>
  <name>b</name>
  <null_material/>
</material>
```

## oren\_nayar

The Oren-Nayar material models very rough surfaces that have no specular reflection.

It's appropriate for materials like clay, sprayed concrete, porous rock, Moon surface, etc..

See the paper 'Generalization of the Lambertian Model and Implications for Machine Vision' (1995) , Michael Oren, Shree K. Nayar, for more details.



Diffuse

Oren Nayar - Sigma 0.3

Oren Nayar - Sigma 0.5

Oren Nayar - Sigma 0.7

Dependence of the Oren-Nayar material on the sigma parameter. Render by Zom-B, model from The Stanford 3D Scanning Repository

## sigma

Controls the roughness of the material. A higher sigma gives a rougher material with more backscattering.

Standard deviation of the microfacet groove slope angles.

Values will be clamped to  $[0, \infty)$

**type:** wavelength-independent material parameter

**units:** radians

## albedo

See diffuse::albedo

## bump

See diffuse::bump

## displacement

See diffuse::displacement

## base\_emission

See diffuse::base\_emission

## emission

See diffuse::emission

## layer

See diffuse::layer

Xml example:

```
<material>
  <name>3</name>
  <oren_nayar>
    <albedo>
      <constant>
        <uniform>
          <value>0.7</value>
        </uniform>
      </constant>
    </albedo>
    <sigma>
      <constant>0.2</constant>
    </sigma>
  </oren_nayar>
</material>
```

# Medium

## medium

Defines a new medium. A medium has a type (much like material have types).

Media types include basic, epidermis, and dermis.

## name

Name of the medium. Used when specifying the internal `_medium_name` in specular etc.. materials.

**type:** string

**unit:**

**restrictions:**

## precedence

Precedence is used to determine which medium is considered to occupy a volume when two or more media occupy the volume. The medium with the highest precedence value is considered to occupy the medium, 'displacing' the other media.

The predefined and default scene medium, 'air', has precedence 1.

**type:** integer

**unit:**

**restrictions:** Should be  $> 1$

## medium::epidermis

Medium for simulating the outer layer of skin.

See Jensen and Donner's paper for more details and example values.

<http://graphics.ucsd.edu/papers/egsr2006skin/egsr2006skin.pdf>

## melanin\_fraction

Fraction of melanin present in tissue.

Typical range: 0 – 0.5

**type:** real scalar

**unit:** dimensionless

**restrictions:** Should be in range [0, 1]

## melanin\_type\_blend

Controls the amount of eumelanin relative to pheomelanin in the tissue.

Typical range: 0 - 1

**type:** real scalar

**unit:** dimensionless

**restrictions:** Should be in range [0, 1]

## medium::dermis

### hemoglobin\_fraction

Controls the amount of hemoglobin present.

Typical range: 0.001 – 0.1

**type:** real scalar

**unit:** dimensionless

**restrictions:** Should be in range [0, 1]

## medium::basic

### ior

Index of refraction. Should be  $\geq 1$ .

Glass has an IOR (index of refraction) of about 1.5, water about 1.33.

The IOR of plastic varies, 1.5 would be a reasonable guess.

*type*: scalar real

*unit*: dimensionless

*restrictions*:  $\geq 1$

### cauchy\_b\_coeff

Sets the 'b' coefficient in [Cauchy's equation](#), which is used in Indigo to govern dispersive refraction. Units are micrometers squared. Setting to 0 disables dispersion. Note: the render can be slower to converge when dispersion is enabled, because each ray refracted through a dispersive medium can represent just one wavelength. So only set `cauchy_b_coeff != 0` if you really want to see dispersion :)

Typical values for glass and water lie in the range 0.003 – 0.01

(see [http://en.wikipedia.org/wiki/Cauchy%27s\\_equation](http://en.wikipedia.org/wiki/Cauchy%27s_equation) for some coefficients)

**type**: scalar real

**unit**: micrometers<sup>2</sup>

**restrictions**: Should be  $\geq 0$  for physical correctness

### absorption\_coefficient\_spectrum

Controls the rate at which light is absorbed as it passes through the medium.

**type**: spectrum element

**unit**: meter<sup>-1</sup>

**restrictions**: Should be  $\geq 0$  for physical correctness

### subsurface\_scattering

Use this element to make the medium scatter light as it passes through it.



element status: optional

### **subsurface\_scattering::scattering\_coefficient\_spectrum**

**type:** spectrum element

**unit:** meter<sup>-1</sup>

**restrictions:** Should be  $\geq 0$  for physical correctness

### **subsurface\_scattering::phase\_function**

Chooses the phase function used for the scattering.

Should contain one phase\_function element (see below).

**type:** phase function element

xml example:

```
<medium>
  <name>scattering_medium</name>

  <ior>1.5</ior>
  <cauchy_b_coeff>0.0</cauchy_b_coeff>
  <absorption_coefficient_spectrum>
    <rgb>
      <rgb>10000.0 5 5</rgb>
    </rgb>
  </absorption_coefficient_spectrum>

  <subsurface_scattering>
    <scattering_coefficient_spectrum>
      <uniform>
        <value>10</value>
      </uniform>
    </scattering_coefficient_spectrum>

    <phase_function>
      <uniform/>
    </phase_function>
  </subsurface_scattering>
</medium>
```

# Phase Function

The phase function controls in what direction light is scattered, when a scattering event occurs.

Must be one of:

## uniform

Takes no parameters

xml example:

```
<phase_function>
  <uniform/>
</phase_function>
```

## henyey\_greenstein

The Henyey-Greenstein phase function can be forwards or backwards scattering, depending on the 'g' parameter.

### henyey\_greenstein::g\_spectrum

The g parameter may vary with wavelength, and is therefore specified using a spectrum element.

Spectrum values will be silently clamped to  $[-0.99, 0.99]$  .

*type*: spectrum element

*units*: dimensionless (average cosine of phase function scattering angle)

*restrictions*: spectrum values should lie in range  $[-1, 1]$

xml example:

```
<phase_function>
  <henyey_greenstein>
    <g_spectrum>
      <uniform>
        <value>0.9</value>
      </uniform>
    </g_spectrum>
  </henyey_greenstein>
</phase_function>
```

# Spectrum

Should have exactly one child, either *peak*, *blackbody*, *rgb*, or *uniform*.

## **spectrum::peak**

### **spectrum::peak::peak\_min**

The wavelength in nm of the start of the spectrum peak.

*type*: real scalar

*units*: nanometers

*restrictions*:  $< \text{peak\_max}$

### **spectrum::peak::peak\_width**

The width of the spectrum peak, in nm.

*type*: real scalar

**units**: nanometers

**restrictions**:  $> 0$

### **spectrum::peak::base\_value**

Exitant radiance for wavelengths outside the peak part of the spectrum.

*type*: real scalar

*units*: spectral radiance,  $\text{W m}^{-3} \text{sr}^{-1}$

*restrictions*:  $\geq 0$

### **spectrum::peak::peak\_value**

Exitant radiance for wavelengths inside the peak part of the spectrum.

**type**: real scalar

*units*: spectral radiance,  $\text{W m}^{-3} \text{sr}^{-1}$

*restrictions*:  $\geq 0$

## **spectrum::blackbody**

### **spectrum::blackbody::temperature**

**type:** real scalar

*units:* Kelvin

*restrictions:*  $> 0$

### **spectrum::blackbody::gain**

Exitant radiance is scaled by this

**type:** real scalar

**units:** dimensionless

**restrictions:**  $> 0$

## **spectrum::rgb**

### **spectrum::rgb::rgb**

**type:** real 3-vector

*units:* spectral radiance,  $\text{W m}^{-3} \text{sr}^{-1}$

*restrictions:* each component must be  $\geq 0$

### **spectrum::rgb::gamma**

The gamma value is used to convert rgb values from image values into intensity-linear display values.

The rgb components are raised by this exponent.

Use 2.2 as a suitable default.

**type:** real scalar

*units:* dimensionless

*restrictions:* must be  $> 0$

## **spectrum::uniform**

### **spectrum::uniform::value**

**type:** real scalar

*units:* spectral radiance,  $\text{W m}^{-3} \text{sr}^{-1}$

*restrictions:* must be  $\geq 0$

## **spectrum::regular\_tabulated**

Allows nearly arbitrary spectra to be defined. The spectrum value is given at regular wavelength intervals, and linear interpolation is used to sample at intermediate wavelengths.

### **spectrum::regular\_tabulated::start\_wavelength**

Wavelength of the first spectrum value.

**type:** real scalar

**units:** meters.

**restrictions:** must be  $\geq 0$

### **spectrum::regular\_tabulated::end\_wavelength**

Wavelength of the last spectrum value.

**type:** real scalar

**units:** meters.

**restrictions:** must be  $\geq 0$

### **spectrum::regular\_tabulated::num\_values**

Number of tabulated values

**type:** integer

**units:**

**restrictions:** must be  $\geq 2$

xml example:

```
<material>
  <name>mat1</name>
```

```
<diffuse>
  <albedo_spectrum>
    <regular_tabulated>
      <start_wavelength>0.4E-06</start_wavelength>
      <end_wavelength>0.7E-06</end_wavelength>
      <num_values>10</num_values>
      <values>
        1 0.9 0.5 0.345 0 0 0 0 0 0
      </values>
    </regular_tabulated>
  </albedo_spectrum>
</diffuse>
</material>
```

## wavelength-dependent material parameter

A wavelength-dependent material parameter may either be *constant*, in which case it does not vary spatially, or it may be controlled by a texture, or it may be controlled by a shader.

A wavelength-dependent material parameter element must have exactly one child element, with the name 'constant', 'texture', or 'shader'.

### constant

A *constant* wavelength-dependent material parameter defines a material parameter that does not vary spatially. However, it can still vary with wavelength, so therefore a spectrum element is used to define the parameter.

**Type:** spectrum element

**units:** depends on context

### texture

A *texture* wavelength-dependent material parameter defines a material parameter that is controlled by a texture map.

### texture::texture\_index

**type:** integer

**restrictions:** must be the 0-based index of a texture defined in the current material.

### shader

A *shader* wavelength-dependent material parameter defines a material parameter that is controlled by a shader program.

For a wavelength-dependent material parameter, a shader program can be defined two different ways. In the first way, the shader is executed once for each wavelength. This allows the most control when creating wavelength-dependent parameters.

To define a shader in this way, you must define a function called 'eval' with signature and return type:

```
eval(real wavelen, vec3 pos) real
```

In the second way, the shader is executed only once for all wavelengths, and returns a RGB 3-vector, that is in turn converted into a spectrum internally in Indigo.

To define a shader in this way, you must define a function called 'eval' with signature and return type:

```
def eval(vec3 pos) vec3
```

## **shader::shader**

**type:** string

**restrictions:** must define a valid shader program.



## wavelength-independent material parameter

A wavelength-independent material parameter may either be *constant*, in which case it does not vary spatially, or it may be controlled by a texture, or it may be controlled by a shader.

A wavelength-independent material parameter element must have exactly one child element, with the name 'constant', 'texture', or 'shader'.

### constant

A *constant* wavelength-dependent material parameter defines a material parameter that does not vary spatially.

**Type:** real scalar

**units:** depends on context

### texture

A *texture* wavelength-dependent material parameter defines a material parameter that is controlled by a texture map.

### texture::texture\_index

**type:** integer

**restrictions:** must be the 0-based index of a texture defined in the current material.

### shader

A *shader* wavelength-independent material parameter defines a material parameter that is controlled by a shader program.

To define a shader in this way, you must define a function called 'eval' with signature and return type:

```
def displacement(vec3 pos) real
```

### shader::shader

**type:** string

**restrictions:** must define a valid shader program.

## rectanglelight

The rectangle light element defines a horizontal area light with normal (0,0,-1).

### pos

The (x, y, z) position of the middle of the rectangle area light.

**type:** real 3-vector

**units:** meters

**restrictions:**

### width

Width in x direction.

**type:** real scalar

**units:** meters

**restrictions:**  $> 0$

### height

Width in y direction.

**type:** real scalar

**units:** meters

**restrictions:**  $> 0$

### spectrum

Emission spectrum for the rectangle light; spectrum element is described above

**type:** spectrum element

### efficacy\_scale

The efficacy\_scale element allows light sources of a given wattage and efficacy to be simulated.

The efficacy\_scale element is optional, if it is used, it overrides the gain. Otherwise the gain works as

normal.

The overall luminous efficiency is the luminous flux per Watt of power drawn.

There are some values on the wikipedia page [http://en.wikipedia.org/wiki/Luminous\\_efficiency](http://en.wikipedia.org/wiki/Luminous_efficiency)

element status: optional

### **efficacy\_scale::power\_drawn**

Power drawn by the light source, e.g. 100 Watts

**type:** real scalar

**units:** Watts

**restrictions:** > 0

### **efficacy\_scale::overall\_luminous\_efficiency**

The overall luminous efficiency is the luminous flux per Watt of power drawn.

**type:** real scalar

**units:** Lumens per Watt (lm/W)

**restrictions:** > 0

example xml:

```
<rectanglelight>
  <pos>0.0 0 1.9</pos>
  <width>0.2</width>
  <height>0.2</height>

  <spectrum>
    <peak>
      <peak_min>300</peak_min>
      <peak_width>550</peak_width>
      <base_value>0</base_value>
      <peak_value>200</peak_value>
    </peak>
  </spectrum>

  <efficacy_scale>
    <power_drawn>100</power_drawn>
    <overall_luminous_efficiency>17.5</overall_luminous_efficiency>
  </efficacy_scale>
</rectanglelight>
```

## exit\_portal

Exit portals are useful for speeding up the rate of convergence of interior renderings, when the interior is lit by an environmental light source, such as the sun/sky model.

Exit portals are placed over the openings between the interior and the exterior environment. These openings are the 'portals' in the scene.

Exit portals make the rendering process more efficient, because paths passing through such openings can be more efficiently sampled when explicitly marked with an exit portal.

Requirements for exit portal usage:

- If exit portals are present in the scene, then all openings must be covered by exit portals. In other words, all possible paths that start on the camera, and then travel through space or a transparent object, and then escape out of the scene into the environment, must be blocked by one or more exit portals.
- The geometric normal (defined by triangle winding order) of an exit portal mesh triangle, where reachable by some path from the camera, must point into the interior of the scene. (i.e. The front side of the mesh faces should be visible by the camera)

## pos

Translation applied to the mesh vertex positions, when transforming from object space into world space. This is also the origin of the object coordinated system in world coordinates.

*type*: real 3-vector

*units*: meters

**restrictions**:

## scale

Uniform scale applied to the mesh vertex positions, when transforming from object space into world space.

*type*: real scalar

**units**: dimensionless

*restrictions*:  $> 0$

## rotation

Optional element that defines a linear transformation that is applied to the mesh vertex positions, when transforming from object space into world space.

Note that position vectors in Indigo are considered to be column vectors.

As of Indigo 0.9, the orthogonality requirement on this matrix has been relaxed, the matrix now must merely be invertible.

### **rotation :: matrix**

Defines a 3x3 matrix, in row-major format.

**type:** real 3x3 matrix

**units:** dimensionless

**restrictions:** Must be invertible.

### **mesh\_name**

Name of a mesh object already defined in the scene file.

**type:** string

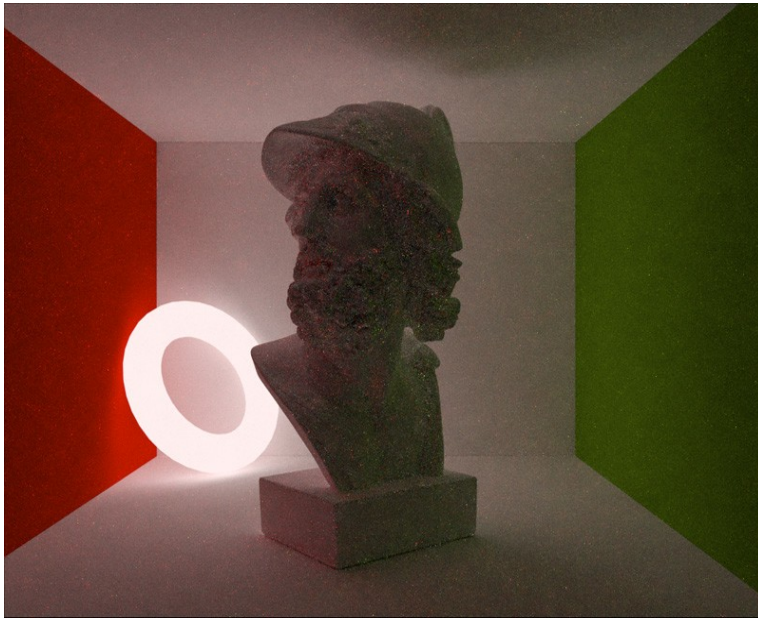
XML example:

```
<exit_portal>
  <pos>0.591146 0.361125 0.957886</pos>
  <scale>1</scale>
  <rotation>
    <matrix>
      1.000 0.000 0.000 0.000 1.0000 0.000 0.000 0.000 1.000
    </matrix>
  </rotation>
  <mesh_name>Plane.011</mesh_name>
</exit_portal>
```

## meshlight

**NOTE:** Meshlight is deprecated, use emitting materials instead.

A mesh light is an emitter that uses triangle mesh geometry. Any mesh that has already been defined can be used to define a mesh light.



### pos

Translation applied to the mesh vertex positions, when transforming from object space into world space. This is also the origin of the object coordinated system in world coordinates.

*type:* real 3-vector

*units:* meters

**restrictions:**

### scale

Uniform scale applied to the mesh vertex positions, when transforming from object space into world space.

*type:* real scalar

**units:** dimensionless

*restrictions:*  $> 0$

## rotation

Optional element that defines a linear transformation that is applied to the mesh vertex positions, when transforming from object space into world space.

Note that position vectors in Indigo are considered to be column vectors.

As of Indigo 0.9, the orthogonality requirement on this matrix has been relaxed, the matrix now must merely be invertible.

## rotation :: matrix

Defines a 3x3 matrix, in row-major format.

**type:** real 3x3 matrix

**units:** dimensionless

**restrictions:** Must be invertible.

## mesh\_name

Name of a mesh object already defined in the scene file.

**type:** string

## spectrum

Emission spectrum for the mesh light; spectrum element is described above

**type:** spectrum element

## efficacy\_scale

The efficacy\_scale element allows light sources of a given wattage and efficacy to be simulated.

The efficacy\_scale element is optional, if it is used, it overrides the gain. Otherwise the gain works as normal.

The overall luminous efficiency is the luminous flux per Watt of power drawn.

There are some values on the wikipedia page [http://en.wikipedia.org/wiki/Luminous\\_efficacy](http://en.wikipedia.org/wiki/Luminous_efficacy)

element status: optional

## **efficacy\_scale::power\_drawn**

Power drawn by the light source, e.g. 100 Watts

**type:** real scalar

**units:** Watts

**restrictions:**  $> 0$

## **efficacy\_scale::overall\_luminous\_efficiency**

The overall luminous efficiency is the luminous flux per Watt of power drawn.

**type:** real scalar

**units:** Lumens per Watt (lm/W)

**restrictions:**  $> 0$

## **texture**

If this element is present, then the light emitted from the mesh light is modulated by an image map.

**element\_status:** optional

**type:** texture element

## **ies\_profile**

If this element is present, then a directional distribution of light will be emitted from the emitter.

The directional distribution is loaded from a file satisfying the ANSI/IESNA LM-63-2002 data system (IES) for describing photometric light distributions.

If the `ies_profile` is present, then the spectral radiance of the emission spectrum will be scaled so that the light emits a luminous flux as defined in the IES file.

When using an IES profile, each triangle of the mesh light will emit light with a directional distribution determined by the IES data, using the normal of the triangle as the 'principle direction'. So you can make the triangle face in any direction and it will work just fine.

When modelling a meshlight that will be used as an IES emitter, make sure it is completely flat, so that all triangle normals are the same.

Only IES files of photometric type 'C' are supported.

Only IES files with vertical angles starting at 0 degrees and ending at 90 degrees are supported.



**element\_status:** optional

## **ies\_profile :: path**

Path to the IES file. Can be absolute or relative. If relative, the path is taken as relative to the scene base directory.

**type:** string

XML example:

```
<meshlight>
  <pos>0 0 2.4</pos>
  <scale>1</scale>
  <spectrum>
    <blackbody>
      <temperature>3500</temperature>
      <gain>1</gain>
    </blackbody>
  </spectrum>
  <mesh_name>prism</mesh_name>

  <efficacy_scale>
    <power_drawn>150</power_drawn>
    <overall_luminous_efficacy>20</overall_luminous_efficacy>
  </efficacy_scale>
</meshlight>
```

# mesh

## **mesh :: name**

Name of the mesh

## **mesh :: scale**

Scales the vertex positions.

## **mesh :: path**

The path to the .3ds mesh, eg 'prism\prism.3ds'. The path is relative to the deepest directory containing the scene file. Either backslashes or forward slashes can be used as directory separators.

## **mesh :: normal\_smoothing**

If this is set to false, normal smoothing will be disabled, and the geometric normal will always be used for this model.

## **mesh :: external**

An external mesh type allows a mesh defined in another file to be loaded and used.

## **mesh :: external :: path**

Path to mesh data file.

Path can be absolute or relative. If relative, it is take as relative to the scene file base path.

Allowed file types are .obj, .3ds, and .ply.

## **mesh :: embedded**

An embedded mesh type allows the mesh to be defined directly in the .igs file.

## **mesh :: embedded :: expose\_uv\_set**

Names a given index of uv (texture) coordinate information. Materials can then bind to the uv coordinates using the given name.

### **mesh :: embedded :: expose\_uv\_set :: index**

Index of the uv coordinates as defined in the embedded mesh data.

**type:** integer

**restrictions:** must be  $\geq 0$ , must be  $<$  than the total number of uv coordinates defined in the mesh data.

**unit:** dimensionless

### **mesh :: embedded :: expose\_uv\_set :: name**

Name with which the uv set will be exposed to the materials.

**type:** string

**restrictions:**

**unit:**

### **mesh :: embedded :: vertex**

Defines a single vertex.

#### **mesh :: embedded :: vertex :: pos (attribute)**

Position of the vertex, in the local coordinate system of the mesh

**type:** real 3-vector

**restrictions:**

**unit:** meters

#### **mesh :: embedded :: vertex :: normal (attribute)**

Normal of the vertex, in the local coordinate system of the mesh

**type:** real 3-vector

**restrictions:** vector must be normalised (have length  $\approx 1.0$ )

**unit:** meters

#### **mesh :: embedded :: vertex :: uvN (attribute)**

N-th uv coordinates. N must be  $\geq 0$  and  $\leq 3$ .

**type:** real 2-vector

**restrictions:**

**unit:** normalised texture coordinates.

### **mesh :: embedded :: triangle\_set**

Defines a group of triangles sharing a common material.

### **mesh :: embedded :: triangle\_set :: material\_name**

Name of the material triangles in this group will use.

**type:** string

**restrictions:** must be the name of an already-defined material.

**Unit:**

### **mesh :: embedded :: triangle\_set :: tri**

Defines a single triangle in terms of its constituent vertices, as a 3-vector of vertex indices.

The indices index into the vertices already defined in the current mesh.

**type:** integer 3-vector

**restrictions:** each vertex index must be  $\geq 0$  and  $<$  the total number of vertices already defined for the current mesh.

**unit:**

Xml example of an external mesh:

```
<mesh>
  <name>hand</name>
  <normal_smoothing>>false</normal_smoothing>
  <scale>0.0005</scale>
  <external>
    <path>..\hand\gipshand2-273k.obj</path>
  </external>
</mesh>
```

An example of an internally defined mesh:

```
<mesh>
```

```
<name>mesh1</name>

<normal_smoothing>false</normal_smoothing>
<embedded>
    <expose_uv_set>
        <index>0</index>
        <name>albedo</name>
    </expose_uv_set>

    <expose_uv_set>
        <index>0</index>
        <name>bump</name>
    </expose_uv_set>

    <vertex pos="-10 -10 0" normal="0 0 1" uv0="0 0" />
    <vertex pos="-10 10 0" normal="0 0 1" uv0="0 10" />
    <vertex pos="10 10 0" normal="0 0 1" uv0="10 10" />
    <vertex pos="10 -10 0" normal="0 0 1" uv0="10 0" />

    <triangle_set>
        <material_name>white</material_name>
        <tri>0 1 2</tri>
        <tri>0 2 3</tri>
    </triangle_set>

    <vertex pos="-10 3 0" normal="0 -1 1" uv0="0 0" />
    <vertex pos="-10 3 20" normal="0 -1 1" uv0="0 10" />
    <vertex pos="10 3 20" normal="0 -1 1" uv0="10 10" />
    <vertex pos="10 3 0" normal="0 -1 1" uv0="10 0" />

    <triangle_set>
        <material_name>checker</material_name>
        <tri>4 5 6</tri>
        <tri>4 6 7</tri>
    </triangle_set>
</embedded>
</mesh>
```

## model

Places a mesh instance into the scene.

### pos

Translation applied to the model vertex positions, when transforming from object space into world space. This is also the origin of the object coordinated system in world coordinates.

*type:* real 3-vector

*units:* meters

**restrictions:**

### scale

Uniform scale applied to the model vertex positions, when transforming from object space into world space.

*type:* real scalar

**units:** dimensionless

*restrictions:*  $> 0$

### rotation

Optional element that defines a linear transformation that is applied to the model vertex positions, when transforming from object space into world space.

Note that position vectors in Indigo are considered to be column vectors.

As of Indigo 0.9, the orthogonality requirement on this matrix has been relaxed, the matrix now must merely be invertible.

### rotation :: matrix

Defines a 3x3 matrix, in row-major format. For example, the identity matrix / null rotation can be defined like this:

```
<rotation>  
  <matrix> 1 0 0 0 1 0 0 0 1 </matrix>  
</rotation>
```

**type:** real 3x3 matrix

**units:** dimensionless

**restrictions:** Must be invertible.

## mesh\_name

Name of a mesh object already defined in the scene file.

**type:** string

## emission\_scale

Scales the amount of light emitted by a certain material as applied to the current model, according to one of several different photometric measures.

**Element status:** optional.

## emission\_scale::material\_name

The name of a material.

## emission\_scale::measure

The following table gives the acceptable values for measure, in the name column:

Name	Unit
luminous_flux	lm
luminous_intensity	cd = lm sr <sup>-1</sup>
luminance	nits = lm sr <sup>-1</sup> m <sup>-2</sup>
luminous_emittance	lux = lm m <sup>-2</sup>

## emission\_scale::value

Defines the value of the corresponding measure.

xml example:

```
<model>
  <rotation>
    <matrix>
      1 0 0 0 0 -1 0 1 0
    </matrix>
  </rotation>
  <pos>0 0 0</pos>
  <scale>1</scale>
```

```
<mesh_name>hand</mesh_name>

<b>emission_scale</b>
  <material_name>mat1</material_name>
  <measure>luminous_flux</measure>
  <value>100000</value>
</emission_scale>
</model>
```

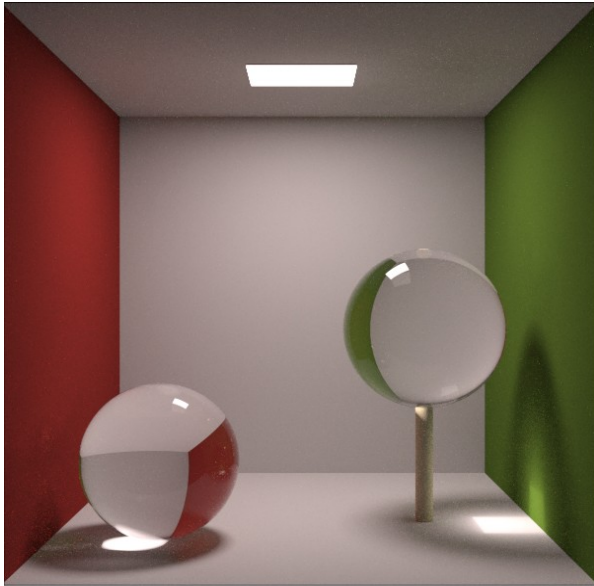


## plane

'Plane' is an infinite plane.

Plane has been removed from Indigo.

## sphere



(c) <http://www.jotero.com> and <http://homepages.paradise.net.nz/nickamy/>

### sphere :: center

**type:** real 3-vector

**units:** meters

**restrictions:**

### sphere :: radius

**type:** real scalar

**units:** meters

**restrictions:**  $> 0$

### sphere :: material\_name

**type:** string

**restrictions:** must be the name of an already specified material.

## Include

The *include* element allows a different .igs file to be loaded and processed during the processing of the main .igs file.

### **include :: pathname**

Path to another .igs file to be processed. Path is absolute or taken as relative to the current scene file directory.

**type:** *string*

**units:**

**restrictions:**

# Indigo Shader Language Reference

## Built-in types

### **real**

A real, scalar number, stored as a floating-point number.

### **int**

A 32-bit signed integer

### **bool**

A boolean value.

### **vec2**

A real 2-vector.

### **vec3**

A real 3-vector.

### **mat2x2**

A 2 x 2 real matrix.

### **mat3x3**

A 3 x 3 real matrix.

## Literal values

Real literal values, integer and boolean literal values can be written with the syntax from C++, e.g. '-1.56e4', '4564', 'true'.

## Function Definitions

Functions are defined like so:

```
def myfunc(vec3 pos) vec3 :  
    vec3 (
```

```

        fbm(pos, 10),
        sin(mul(doti(getTexCoords(0)), 100.0)),
        sin(mul(dotj(getTexCoords(0)), 100.0))
    )

```

The *def* keyword starts a function definition. Next is the name of the function, then the list of arguments to the function, where each argument name is preceded by the argument type. Then the return type of the function follows. After the colon, the body of the function is defined.

## Built-in functions – Conditional functions

**if(bool p, int a, int b) int**

**if(bool p, real a, real b) real**

If p is true, returns a, otherwise returns b.

## Built-in functions – Comparison functions

**eq(int a, int b) bool**

**eq(real a, real b) bool**

**eq(bool a, bool b) bool**

Returns true if a = b, false otherwise.

**neq(int a, int b) bool**

**neq(real a, real b) bool**

**neq(bool a, bool b) bool**

Returns false if a = b, true otherwise.

**lt(int a, int b) bool**

**lt(real a, real b) bool**

Returns  $a < b$ .

**lte(int a, int b) bool**

**lte(real a, real b) bool**

Returns  $a \leq b$ .

**gt(int a, int b) bool**

**gt(real a, real b) bool**

Returns  $a > b$ .

**gte(int a, int b) bool**

**gte(real a, real b) bool**

Returns  $a \geq b$ .

## Built-in functions – Boolean functions

***not(bool a) bool***

Returns  $\sim a$ .

***or(bool a, bool b) bool***

Returns  $a \vee b$ .

***and(bool a, bool b) bool***

Returns  $a \wedge b$ .

## Built-in functions – Maths utility functions

```
add(real x, real y) real
```

```
add(int x, int y) int
```

```
add(vec2 x, vec2 y) vec2
```

```
add(vec3 x, vec3 y) vec3
```

Returns  $x + y$ .

```
sub(real x, real y) real
```

```
sub(int x, int y) int
```

```
sub(vec2 x, vec2 y) vec2
```

```
sub(vec3 x, vec3 y) vec3
```

Returns  $x - y$ .

```
mul(real x, real y) real
```

```
mul(int x, int y) int
```

Returns  $x * y$ .

```
div(real x, real y) real
```

```
div(int x, int y) int
```

Returns  $x / y$ .

```
mod(real x, real y) real
```

```
mod(int x, int y) int
```

Returns the remainder of  $x / y$ .

**sin(real x) real**

Returns  $\sin(x)$ .

**asin(real x) real**

Returns  $\sin^{-1}(x)$ .

**cos(real x) real**

Returns  $\cos(x)$ .

**acos(real x) real**

Returns  $\cos^{-1}(x)$ .

**tan(real x) real**

Returns  $\tan(x)$ .

**atan(real x) real**

Returns  $\tan^{-1}(x)$ .

**abs(real x) real**

**abs(int x) int**

Returns the absolute value of  $x$ .

**exp(real x) real**

Returns  $e^x$ .

**pow(real x, real y) real**

Returns  $x^y$ .

**sqrt(real x) real**

Returns  $x^{1/2}$ .



## **log(real x) real**

Returns the natural logarithm of  $x$ ,  $\ln(x)$ .

## Built-in functions – Vector constructors

### **vec2(real x, real y) vec2**

Returns the 2-vector  $(x, y)$ .

### **vec2(real x) vec2**

Returns the 2-vector  $(x, x)$ .

### **vec3(real x, real y, real z) vec3**

Returns the 3-vector  $(x, y, z)$ .

### **vec3(real x) vec3**

Returns the 3-vector  $(x, x, x)$ .

## Built-in functions – Vector functions

### **dot(vec2 a, vec2 b) real**

### **dot(vec3 a, vec3 b) real**

Returns the dot product of **a** and **b**.

### **cross(vec3 a, vec3 b) vec3**

Returns the cross product of **a** and **b**.

### **neg(vec3 a) vec3**

Returns **-a**.

`length(vec2 a) real`

`length(vec3 a) real`

Returns the length of **a**,  $\|\mathbf{a}\|$ .

`normalise(vec3 a) vec3`

Returns  $\mathbf{a} / \|\mathbf{a}\|$

`doti(vec3 a) real`

Returns  $\mathbf{i} \cdot \mathbf{a}$ , where **i** is the standard basis vector (1,0,0)

`dotj(vec3 a) real`

Returns  $\mathbf{j} \cdot \mathbf{a}$ , where **j** is the standard basis vector (0,1,0)

`dotk(vec3 a) real`

Returns  $\mathbf{k} \cdot \mathbf{a}$ , where **k** is the standard basis vector (0,0,1)

`doti(vec2 a) real`

Returns  $\mathbf{i} \cdot \mathbf{a}$ , where **i** is the standard basis vector (1,0)

`dotj(vec2 a) real`

Returns  $\mathbf{j} \cdot \mathbf{a}$ , where **j** is the standard basis vector (0,1)

`mul(vec2 a, real b) vec2`

`mul(vec3 a, real b) vec3`

Returns  $\mathbf{a} * b$

## Built-in functions – Matrix constructors

`mat2x2(real e11, real e12, real e21, real e22) mat2x2`

Returns the 2 x 2 matrix

$e_{11} \ e_{12}$

e21 e22

mat3x3(real e11, real e12, real e13, real e21, real e22, real e23, real e31, real e32, real e33) mat3x3

Returns the 3 x 3 matrix

e11 e12 e13

e21 e22 e23

e31 e32 e33

## Built-in functions – Matrix operations

mul(mat2x2 A, mat2x2 B) mat2x2

Returns the 2 x 2 matrix **AB**.

mul(mat3x3 A, mat3x3 B) mat3x3

Returns the 3 x 3 matrix **AB**.

mul(mat2x2 A, vec2 b) vec2

Returns the 2-vector (2 x 1 matrix) **Ab**.

mul(mat3x3 A, vec3 b) vec3

Returns the 3-vector (3 x 1 matrix) **Ab**.

transpose(mat2x2 A) mat2x2

transpose(mat3x3 A) mat3x3

Returns the transpose of **A**.

inverse(mat2x2 A) mat2x2

inverse(mat3x3 A) mat3x3

Returns the inverse of **A**. Undefined if **A** is not invertible.

## Built-in functions – procedural noise functions

**noise(real x) real**

Returns 1-D Perlin noise evaluated at **x**.

**noise(vec2 x) real**

Returns 2-D Perlin noise evaluated at **x**.

**noise(vec3 x) real**

Returns 3-D Perlin noise evaluated at **x**.

**fbm(real x, int oc) real**

Returns *oc* octaves of 1-D Fractal Brownian Motion noise evaluated at **x**.

**fbm(vec2 x, int oc) real**

Returns *oc* octaves of 2-D Fractal Brownian Motion noise evaluated at **x**.

**fbm(vec3 x, int oc) real**

Returns *oc* octaves of 3-D Fractal Brownian Motion noise evaluated at **x**.

## Built-in functions – texture sampling functions

**getTexCoords(int texcoord\_set\_index) vec2**

Gets the *i*-th texture coordinates at the shading point, where *i* = *texcoord\_set\_index*.

**sample2DTextureVec3(int texture\_index, vec2 st) vec3**

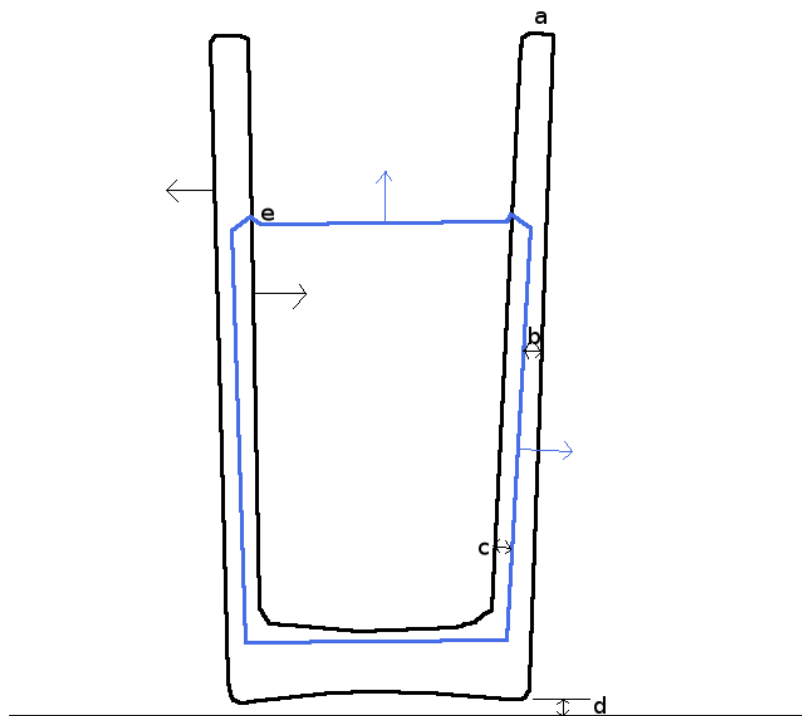
Samples the *i*-th texture defined in the current material, where *i* = *texture\_index* is a 0-based index, at the normalised coordinates (*s*, *t*). Returns a (R, G, B) triplet, where each component will be in the range [0, 1].



## Appendix A: Modelling a Liquid in a Glass For Indigo

If you want to create a realistic rendering of a liquid in a glass vessel in Indigo, you must model it in a rather particular way, to take advantage of the medium precedence system, so that all light scattering and transmission processes are simulated (reasonably) accurately.

In the requirements given below, a distance of 0.2mm is often mentioned. This distance is chosen because the default ray origin 'nudge distance' (used to avoid false self-intersection due to limited floating point precision) in Indigo is 0.1mm.



Nick C. 06

### Modelling Requirements:

a) need sufficient polygonisation of the curve here because sharp edges cause shading normal artifacts. For example 10-20 points may be needed on piecewise curve.

b) distance should be  $> 0.2\text{mm}$

c) distance should be  $> 0.2\text{mm}$ .

Note that while the distance between the glass and **water** surfaces in the walls of the glass needs only to be greater than 0.2mm, it should probably be about 1mm, depending on the width of the glass

d) distance between glass and ground plane should be in range  $[0.2\text{mm}, 1\text{mm}]$

e) meniscus should be modelled

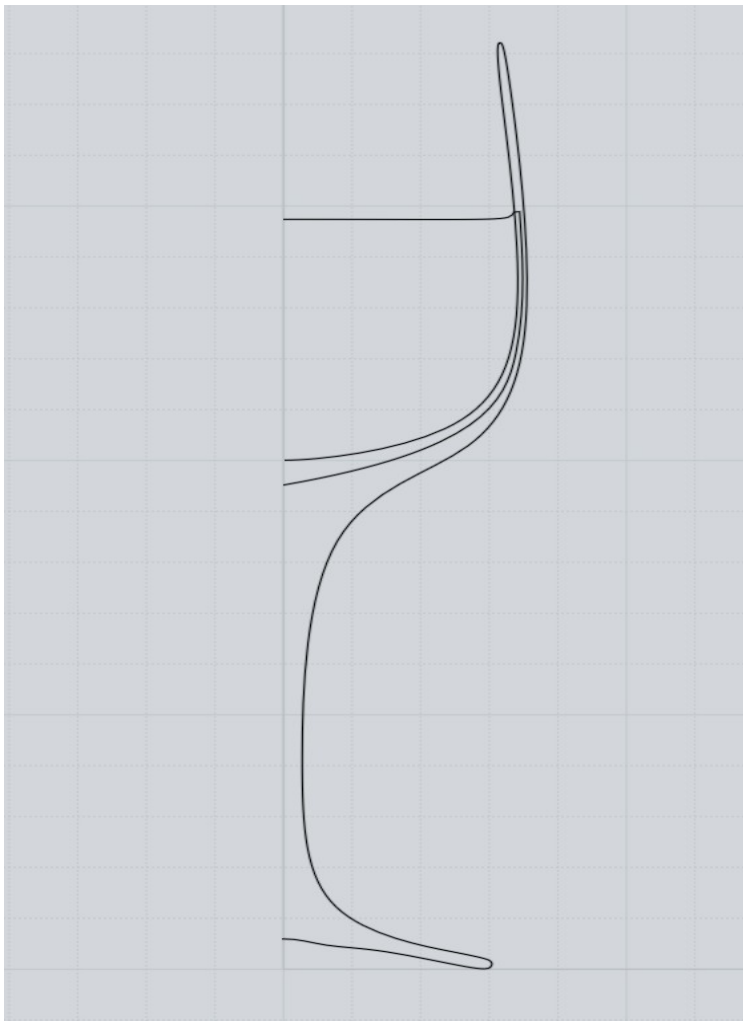
### Additionally:

f) all geometric (given by tri winding order) and smoothed normals should be as labelled.

- g) total number of triangles is expected to be in range 10000-150000
- h) surfaces should be two-manifold and closed (not self-intersecting).
- i) **liquid** and glass should be assigned different materials.
- j) part of the surface of the **liquid** component should lie inside the glass as shown.
- k) The model should be created in units of meters.
- l) The glass medium precedence should be greater than that of the **liquid** medium (because glass displaces water), which in turn should be greater than 1.

The following image shows a wine glass half-profile modelled in MoI, which can easily be turned into a complete 3d model using a lathe/revolve modifier.

Note how the lower and side edge of the wine volume lies inside the wall of the glass.



The wine glass rendered in Indigo:





# Appendix B: Scattering Properties for Liquids

These values are from the paper *Acquiring Scattering Properties of Participating Media by Dilution* by Jensen et al. ([http://graphics.ucsd.edu/~henrik/papers/acquiring\\_scattering\\_properties/](http://graphics.ucsd.edu/~henrik/papers/acquiring_scattering_properties/))

Waat has kindly converted the units from the paper into the SI units that Indigo uses.

Scattering Properties for Liquids			Properties for media diluted in 23-V Liters of Water												Indigo Values																				
			Extinction (x10 <sup>-2</sup> mm <sup>-1</sup> )						Scattering (x10 <sup>-2</sup> mm <sup>-1</sup> )						Average Cosine						Absorption (m <sup>-1</sup> )						Scattering (m <sup>-1</sup> )						G Spectrum		
Material	Volume (V) for Single Scattering (L)	Concentration (c1)	R	G	B	R	G	B	R	G	B	R	G	B	Concentration (c2)	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B					
Milk (lowfat)	0.02	0.07%	0.9126	1.0748	1.2500	0.9124	1.0744	1.2492	0.9320	0.9020	0.8590	100%	2.875	5.750	11.500	13115.750	15444.500	17957.250	0.932	0.902	0.859														
Milk (reduced)	0.02	0.08%	1.0750	1.2213	1.3941	1.0748	1.2203	1.3931	0.8190	0.7970	0.7460	100%	2.556	5.111	12.778	13733.556	15600.388	17800.722	0.819	0.797	0.746														
Milk (regular)	0.02	0.07%	1.1874	1.3296	1.4602	1.1873	1.3293	1.4589	0.7500	0.7140	0.6810	100%	1.533	4.300	18.933	18205.267	20382.600	22369.800	0.750	0.714	0.681														
Coffee (espresso)	0.01	0.03%	0.4376	0.5115	0.6048	0.2707	0.2823	0.2970	0.9070	0.8960	0.8800	100%	4798.375	6575.125	8849.250	7782.625	8130.500	8538.750	0.907	0.896	0.880														
Coffee (mint mocha)	0.01	0.03%	0.1900	0.2600	0.3500	0.0916	0.1081	0.1480	0.9100	0.9070	0.9140	100%	3772.000	5822.833	7820.000	3511.333	4143.833	5596.667	0.910	0.907	0.914														
Soy Milk (lowfat)	0.02	0.07%	0.1419	0.1625	0.2740	0.1418	0.1620	0.2715	0.8500	0.8530	0.8420	100%	1.437	7.188	35.937	2038.375	2328.750	3902.813	0.850	0.853	0.842														
Soy Milk (regular)	0.01	0.05%	0.2434	0.2719	0.4597	0.2433	0.2714	0.4563	0.8730	0.8580	0.8320	100%	1.917	9.583	65.167	4663.250	5201.833	8745.750	0.873	0.858	0.832														
Choc. Milk (lowfat)	0.01	0.04%	0.4282	0.5014	0.5791	0.4277	0.4998	0.5723	0.9340	0.9270	0.9160	100%	11.500	36.800	156.400	9837.100	11495.400	13162.900	0.934	0.927	0.916														
Choc. Milk (regular)	0.02	0.07%	0.7359	0.9172	1.0688	0.7352	0.9142	1.0588	0.8620	0.8380	0.8060	100%	10.062	43.125	143.750	10568.500	13141.625	15220.250	0.862	0.838	0.806														
Soda (coke)	1.6	6.96%	0.7143	1.1688	1.7166	0.0177	0.0208	0.0000	0.9650	0.9720		100%	100.136	165.025	246.804	2.544	2.990	0.000	0.965	0.972	0.000														
Soda (pepsi)	1.6	6.96%	0.6433	0.9990	1.4420	0.0058	0.0141	0.0000	0.9260	0.9790		100%	91.641	141.579	207.288	0.834	2.027	0.000	0.926	0.979	0.000														
Soda (sprite)	1.6	65.22%	0.1299	0.1283	0.1395	0.0069	0.0085	0.0089	0.9430	0.9530	0.9520	100%	1.886	1.831	2.003	0.106	0.136	0.136	0.943	0.953	0.952														
Sports Gatorade	1.6	6.52%	0.4009	0.4185	0.4324	0.2396	0.2927	0.3745	0.9330	0.9330	0.9350	100%	24.794	19.289	8.878	36.677	44.881	57.423	0.933	0.933	0.935														
Wine (chardonnay)	3.3	14.35%	0.1577	0.1748	0.3512	0.0030	0.0047	0.0069	0.9140	0.9580	0.9750	100%	10.782	11.855	23.997	0.209	0.328	0.481	0.914	0.958	0.975														
Wine (white zinfandel)	3.3	14.35%	0.1763	0.2370	0.2913	0.0031	0.0048	0.0066	0.9190	0.9430	0.9720	100%	12.072	16.184	19.842	0.216	0.335	0.460	0.919	0.943	0.972														
Wine (merlot)	1.6	6.52%	0.7639	1.6428	1.9196	0.0053	0.0000	0.0000	0.9740			100%	116.319	251.911	294.338	0.813	0.000	0.000	0.974	0.000	0.000														
Beer (budweiser)	2.0	12.61%	0.1486	0.3210	0.7380	0.0037	0.0063	0.0074	0.9170	0.9560	0.9820	100%	11.492	24.911	57.786	0.293	0.547	0.587	0.917	0.956	0.982														
Beer (coorslight)	1	4.35%	0.0295	0.0663	0.1521	0.0027	0.0055	0.0000	0.9180	0.9660		100%	6.164	13.984	34.983	0.621	1.265	0.000	0.918	0.966	0.000														
Beer (yuengling)	2.0	12.61%	0.1535	0.3322	0.7452	0.0495	0.0521	0.0597	0.9690	0.9690	0.9750	100%	8.248	22.215	54.367	3.926	4.132	4.735	0.969	0.969	0.975														
Detergent (clorox)	1.2	5.22%	0.1600	0.2500	0.3300	0.1428	0.1723	0.1928	0.9120	0.9050	0.8920	100%	3.354	14.893	26.297	27.313	33.024	36.953	0.912	0.905	0.892														
Detergent (era)	2.3	10.00%	0.7987	0.5746	0.2849	0.0553	0.0586	0.0906	0.9490	0.9500	0.9710	100%	74.340	51.600	19.430	5.530	5.860	9.060	0.949	0.950	0.971														
Apple Juice	1.8	7.83%	0.1215	0.2101	0.4407	0.0201	0.0243	0.0323	0.9470	0.9490	0.9450	100%	12.957	23.741	52.184	2.568	3.105	4.127	0.947	0.949	0.945														
Cranberry Juice	1.6	6.52%	0.2700	0.6300	0.8300	0.0128	0.0155	0.0196	0.9470	0.9510	0.9740	100%	39.437	94.223	124.261	1.963	2.377	3.005	0.947	0.951	0.974														
Grape Juice	1.2	5.22%	0.5500	1.2500	1.5300	0.0072	0.0000	0.0000	0.9610			100%	104.037	239.583	293.250	1.380	0.000	0.000	0.961	0.000	0.000														
Ruby Grapefruit Juice	0.24	1.04%	0.2513	0.3517	0.4305	0.1617	0.1606	0.1669	0.9290	0.9290	0.9310	100%	85.867	183.138	252.617	154.963	153.908	159.946	0.929	0.929	0.931														
White Grapefruit Juice	0.16	0.70%	0.3609	0.3800	0.5632	0.3513	0.3665	0.5237	0.5480	0.5450	0.5650	100%	13.800	18.831	56.781	504.994	527.419	752.819	0.548	0.545	0.565														
Shampoo (balancing)	0.3	1.30%	0.0288	0.0710	0.0952	0.0104	0.0114	0.0147	0.9100	0.9050	0.9200	100%	14.107	45.693	61.717	7.973	8.740	11.270	0.910	0.905	0.920														
Shampoo (strawberry)	0.3	1.30%	0.0217	0.0788	0.1022	0.0028	0.0032	0.0033	0.9270	0.9350	0.9940	100%	14.490	57.960	75.823	2.147	2.453	2.530	0.927	0.935	0.994														
Head & Shoulders	0.24	1.04%	0.3674	0.4527	0.5211	0.2791	0.2890	0.3086	0.9110	0.8960	0.8840	100%	84.621	156.879	203.646	267.471	276.958	295.742	0.911	0.896	0.884														
Lemon Tea Powder	5tbsp	0.3400	0.5800	0.8800	0.0795	0.0895	0.1073	0.9460	0.9460	0.9490	5tbsp	2.602	4.902	7.727	0.798	0.898	1.073	0.946	0.946	0.949															
Orange Powder	4tbsp	0.3377	0.5573	1.0122	0.1928	0.2132	0.2259	0.9190	0.9180	0.9220	4tbsp	1.449	3.441	7.863	1.928	2.132	2.259	0.919	0.918	0.922															
Pink Lemonade Powder	5tbsp	0.2400	0.3700	0.4500	0.1235	0.1334	0.1305	0.9020	0.9020	0.9040	5tbsp	1.165	2.366	3.195	1.235	1.334	1.305	0.902	0.902	0.904															
Cappuccino Powder	0.25tbsp	0.2574	0.3536	0.4840	0.0654	0.0882	0.1568	0.8490	0.8430	0.9260	0.25tbsp	1.920	2.654	3.272	0.654	0.882	1.568	0.849	0.843	0.926															
Salt Powder	1.75cup	0.7600	0.8685	0.9363	0.2485	0.2822	0.3216	0.8020	0.7930	0.8210	1.75cup	5.115	5.863	6.147	2.485	2.822	3.216	0.802	0.793	0.821															
Sugar Powder	5cup	0.0795	0.1758	0.2783	0.0145	0.0162	0.0202	0.9210	0.9190	0.9310	5cup	0.650	1.597	2.578	0.145	0.162	0.202	0.921	0.919	0.931															
Suisse Mocha Powder	0.5tbsp	0.5098	0.6476	0.7944	0.3223	0.3583	0.4148	0.9070	0.8940	0.8880	0.5tbsp	1.875	2.893	3.796	3.223	3.583	4.148	0.907	0.894	0.888															
Mission Bay Surface (1-2) hours		100.00%	3.3623	3.2928	3.2193	0.2415	0.2765	0.3256	0.8420	0.8650	0.9120	100%	31.208	30.164	28.937	2.415	2.765	3.256	0.842	0.865	0.912														
Pacific Ocean Surface (1 hour)		100.00%	3.36	3.32	3.24	0.18	0.18	0.23	0.9	0.83	0.91	100%	31.85	31.32	30.15	1.8	1.83	2.28	0.9	0.83	0.91														
Mission Bay 10ft Deep (30 min)		100.00%	3.41	3.34	3.28	0.1	0.13	0.19	0.73	0.82	0.92	100%	33.07	32.14	30.94	0.99	1.27	1.88	0.73	0.82	0.92														
Mission Bay 10ft deep (8 hours)		100.00%	3.4	3.35	3.29	0.1	0.1	0.16	0.93	0.91	0.95	100%	32.98	32.42	31.32	1.02	1.03	1.61	0.93	0.91	0.95														