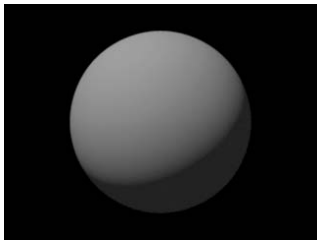
NVIDIA® Gelato®
Shader Library Documentation

Date: 16 December, 2005

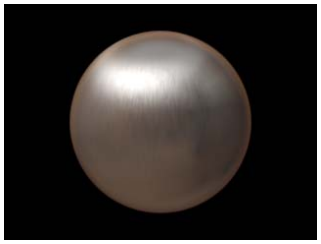
Abstract

Gelato comes with a variety of shaders and shader library routines. These web pages document Gelato's shader library, including detailed descriptions of the use of all shader parameters.

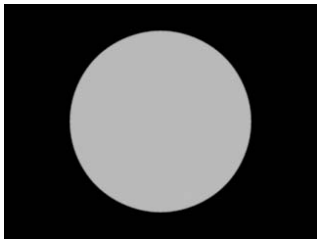
Basic Materials and Surface Shaders



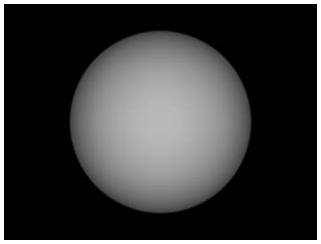
clay.gsl — Clay or other rough diffuse material.



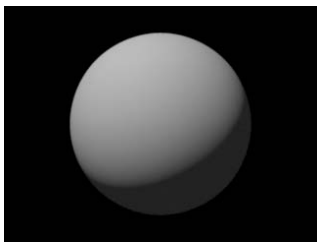
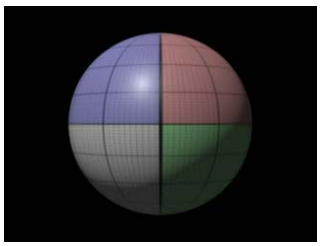
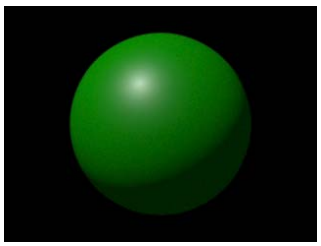
brushedmetal.gsl — Brushed metal



constant.gsl — Constant unlit color



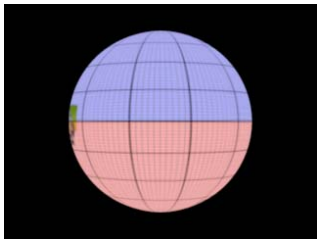
defaultsurface.gsl — Unlit surface that is brighter when facing the camera

	envsurf.gsl	"Backdrop" surface that just looks up into an environment map in the view direction.
	matte.gsl	Lambertian diffuse material
	metal.gsl	Metal with optional reflections
	paintedplastic.gsl	Plastic with a base color determined by texture map
	plastic.gsl	smooth plastic material
	plasticss.gsl	Plastic with subsurface scattering.
	shinyplastic.gsl	Polished plastic with reflections

Surface layers

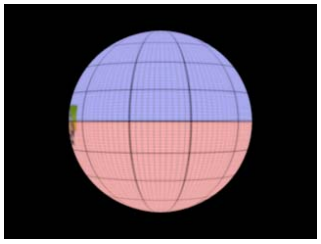
checker.gsl — Basic checker pattern

postgloss.gsl — Layer that adds reflective gloss to an existing shader group



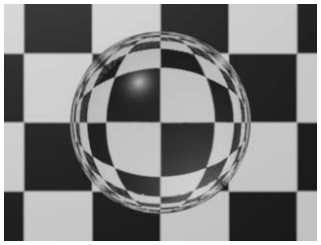
pretexture.gsl — Texture map lookup to set the color of C prior to another layer

subsurflayer.gsl — Layer that adds subsurface scattering to an existing shader group

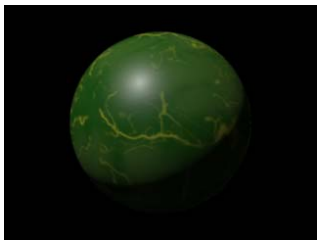


texmap.gsl — Texture map lookup that can be fed into another layer

Complex surfaces



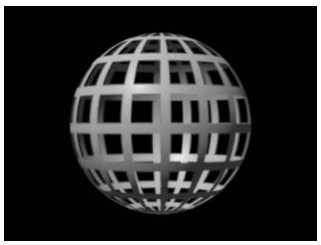
`glass.gsl` Refractive ray-traced glass.



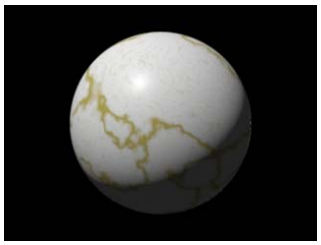
`greenmarble.gsl` Green marble (procedural)

`oak.gsl` Oak wood material.

`oakplank.gsl` Oak planks.

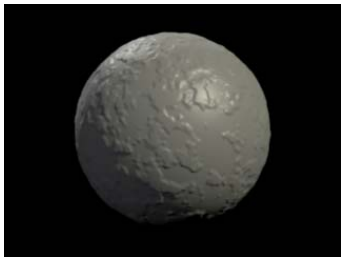


`screen.gsl` Rectangular screen.

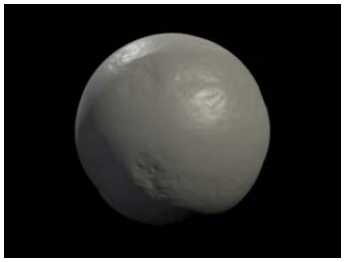


`veinedmarble.gsl` Veined marble (procedural)

Displacement shaders

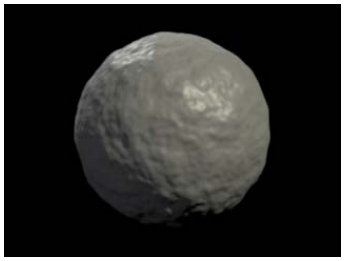


castucco.gsl — Stucco displacement.

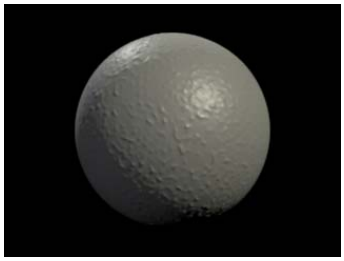


dented.gsl — Dented, hammered, and generally bashed around.

dispmap.gsl — Simple displacement mapping from a texture or using input from an earlier layer.

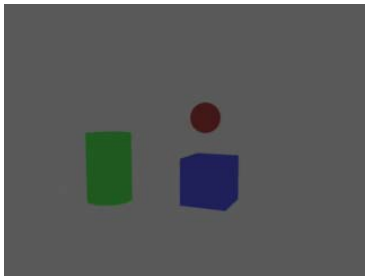


lumpy.gsl — General lumpy noise surface.

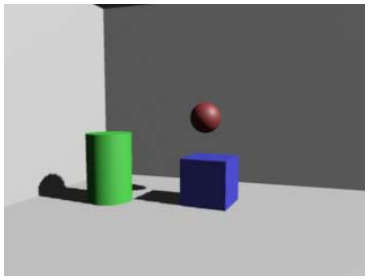


stucco.gsl — Stucco displacement

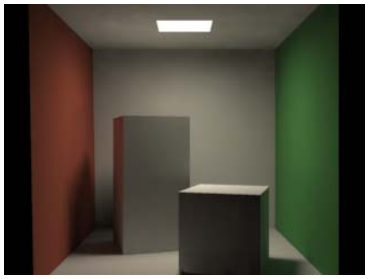
Light sources



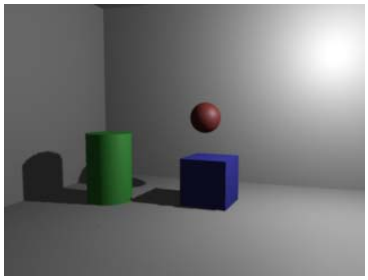
ambientlight.gsl — Ambient light source.



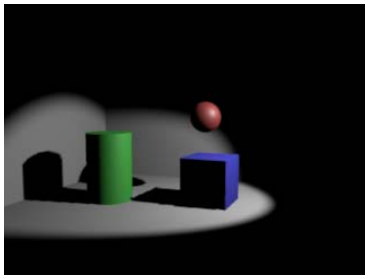
distantlight.gsl — Infinitely-distant light with no falloff (like the sun).



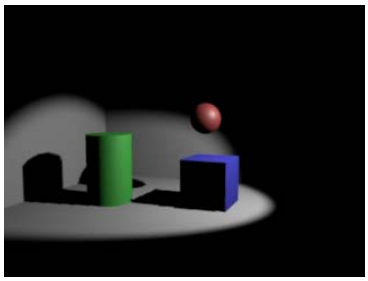
indirectlight.gsl — Light that adds indirect global illumination to the scene.



pointlight.gsl — Simple point light source.



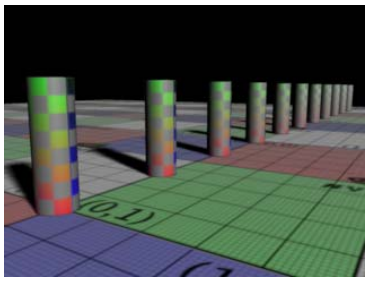
spotlight.gsl — Simple spot light with a circular cross section.



uberlight.gsl

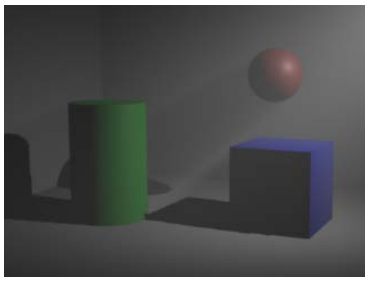
Flexible light with lots of artistic controls.

Volume shaders



desatdist.gsl

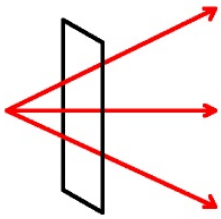
Desaturate the apparent color of distant objects.



smoke.gsl

Smoke volume.

Debugging shaders



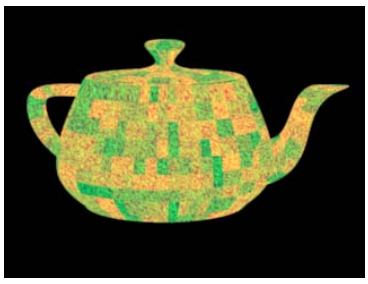
raygoggles.gsl

View behind the surface as the scene is viewed by the ray tracer.

showdudv.gsl

Visualize the derivatives of u and v.

`showfacing.gsl` Identify backfacing surfaces.



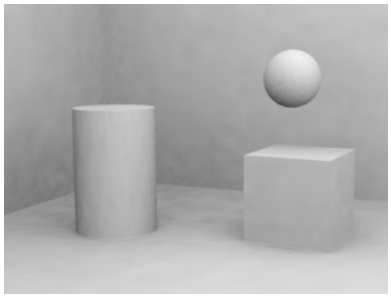
`showgrids.gsl` Visualize the grids and shading points.

`shownormals.gsl` Visualize the surface normal, `N`.

`showst.gsl` Show the *u,v* coordinates of an object.

`showuv.gsl` Show the *u,v* coordinates of an object.

Special-purpose shaders

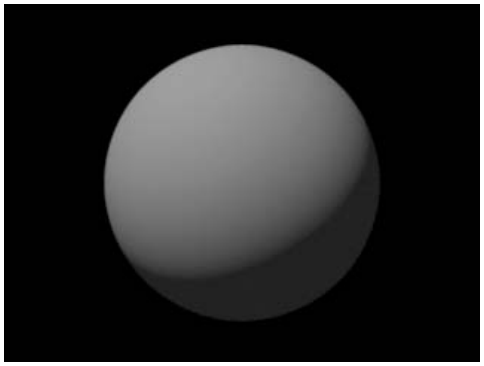


ambocclude.gsl

Ambient occlusion -- encode sky visibility as gray scale.

Basic Materials & Surface Shaders

clay.gsl



Description

The surface shader `clay` makes a realistic material for diffuse but microscopically rough or dusty surfaces (such as clay or the surface of the moon). This material uses the Oren-Nayar reflection model, which is more realistic than the ideal Lambertian because Oren-Nayar tends to act somewhat as a retro-reflector (stronger reflection in the direction the light comes from) as opposed to Lambertian tendency to reflect equally in all directions.

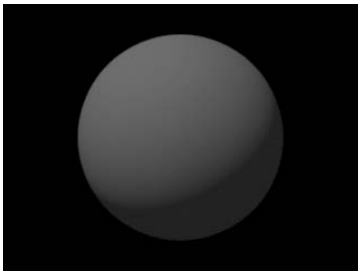
Parameters

float Ka = 1

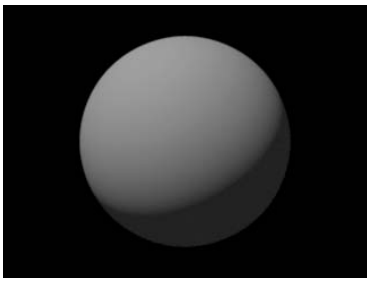
The `Ka` parameter scales the surface's ambient reflectivity (the degree to which it responds to ambient lights).

float Kd = 0.7

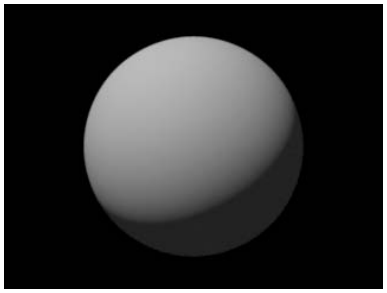
The `Kd` parameter scales the surface's diffuse reflectivity.



Kd = 0.3



Kd = 0.7 (default)

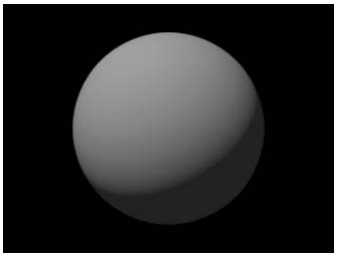


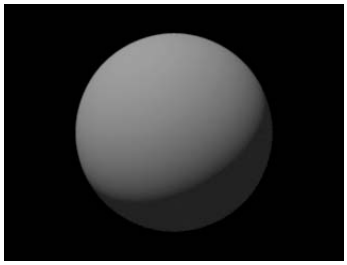
Kd = 1

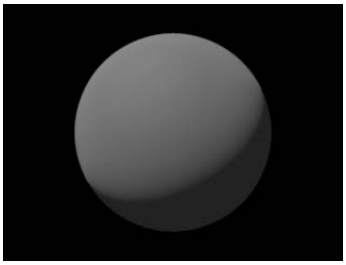
float roughness = 0.1

Controls the roughness of the surface. The more rough the surface is, the more it will behave as a retroreflector, and will resemble a microscopically pitted or dusty material. When roughness = 0, the behavior is Lambertian (i.e., identical to `matte.gal`).

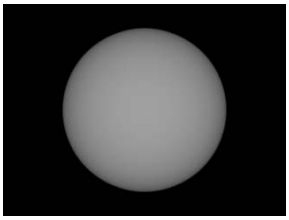
Side view (subtle):

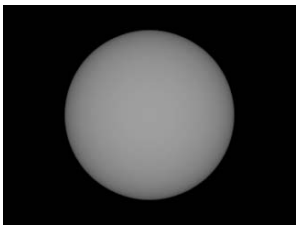

roughness = 0

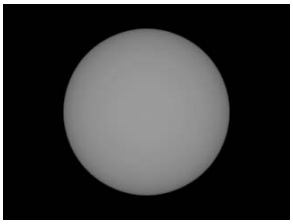

roughness = 0.1 (default)


roughness = 1

Front view (retroreflective tendency more visible):


roughness = 0


roughness = 0.1 (default)


roughness = 0.25

Attributes and Globals

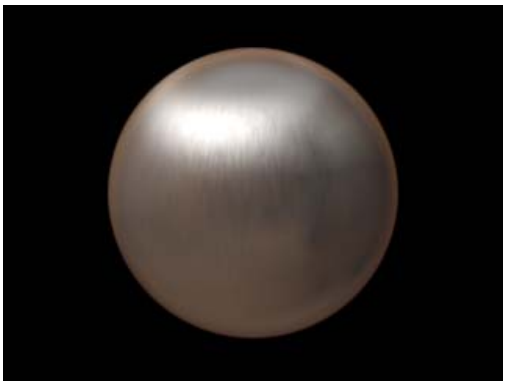
"color C", (1,1,1)

`C` provides the base color of the material.

"color opacity", (1,1,1)

`opacity` provides the opacity of the material.

brushedmetal.gsl



Description

The surface shader `brushedmetal` makes an anisotropic procedural brushed metal.

Anisotropic means that it doesn't reflect evenly in all directions. Due to microscopic aligned grooves (usually the result of rolling the metal into sheets), the metal reflects differently depending on the orientation of the grooves relative to the viewing and reflection directions.

Metals have specular highlights that are tinted the color of the metal (which looks quite different from plastics and other non-metals, which tend to have white highlights regardless of the color of the underlying substrate).

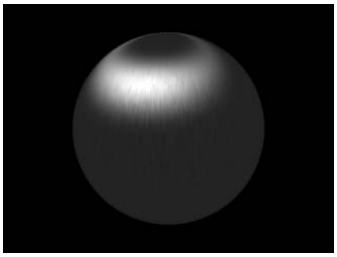
Parameters

float Ka = 1

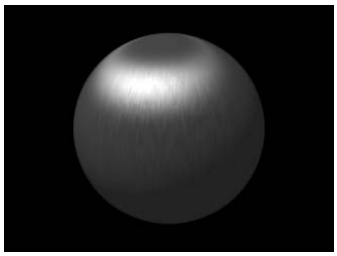
The `Ka` parameter scales the surface's ambient reflectivity (the degree to which it responds to ambient lights).

float Kd = 0.1

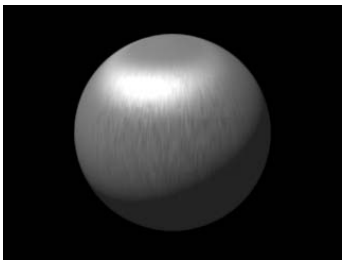
The `Kd` parameter scales the surface's diffuse reflectivity. Metallic objects tend to have very low `Kd` values, which is why the default is only 0.1. The more polished the surface, the lower the `Kd` value should be.



Kd = 0



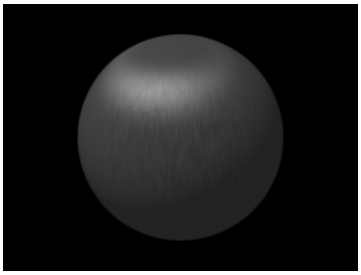
Kd = 0.1 (default)



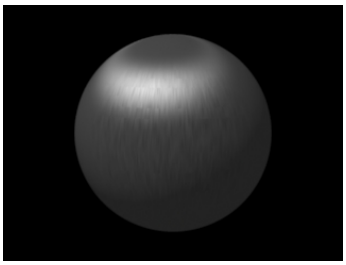
Kd = 0.5

float Ks = 0.8

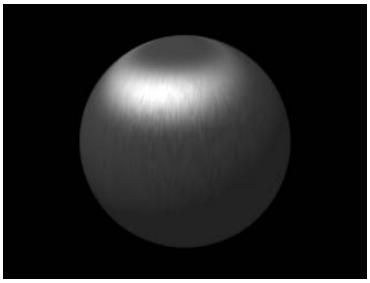
The `Ks` parameter scales the surface's specular highlights. When Ks = 0, there are no specular highlights at all.



Ks = 0.2



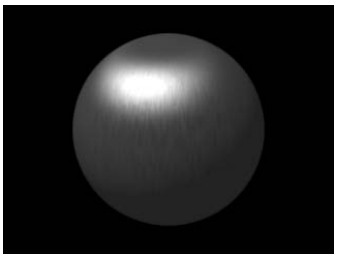
Ks = 0.5

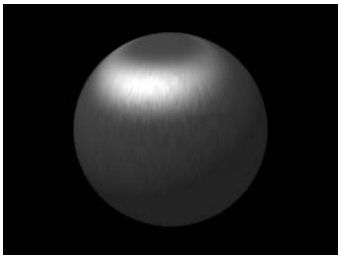


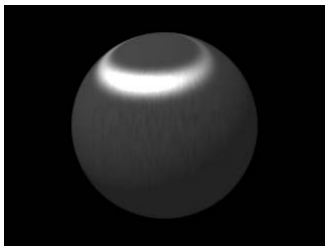
Ks = 0.8 (default)

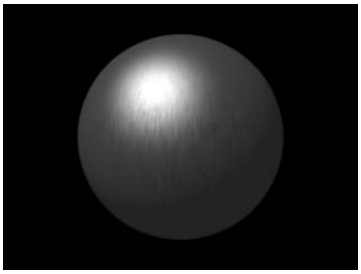
float uroughness = 0.2
float vroughness = 0.5

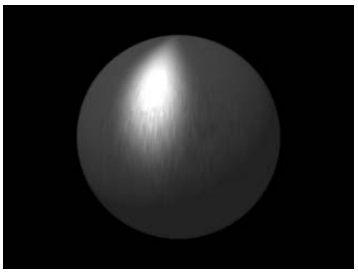
The `uroughness` and `vroughness` parameters control the sharpness of the surface's specular highlights. Unlike ordinary metal, anisotropic metal has potentially different roughness values in each of the two principal directions.

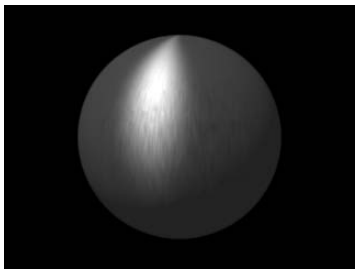

uroughness = 0.2
vroughness = 0.4


uroughness = 0.2
vroughness = 0.5


uroughness = 0.1
vroughness = 0.5


uroughness = 0.3
vroughness = 0.3


uroughness = 0.4
vroughness = 0.2


uroughness = 0.5
vroughness = 0.2

string projection = "st"

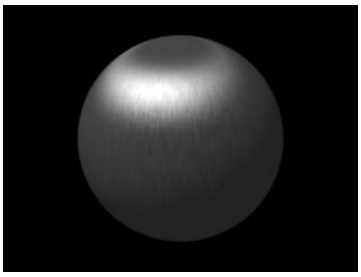
The `projection` determines the orientation of the grain, as well as which tangent axes correspond to the u and v roughness values. It may take one of the following values:

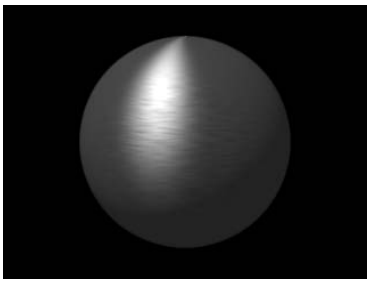
`projection = "st"`
The default, indictes that the grain is oriented using s as the short direction and t as the long direction, and the tangent dPdv indicates the direction of the brush finish.

`projection = "ts"`
This choice reverses the usual orientation of the brush finish: the grain is oriented using t as the short direction and s as the long direction, and the tangent dPdu indicates the direction of the brush finish.

`projection = "Pref"`
This choice doesn't use s and t at all, but rather projects primitive variable `Pref` (giving a reference pose) into the coordinate system given by `"shadingspace"`,

and in this space the grains and brush orientation are aligned to the direction given by vector `"graindir"`.


projection = "st" (default)


projection = "ts"

float s = u
float t = v

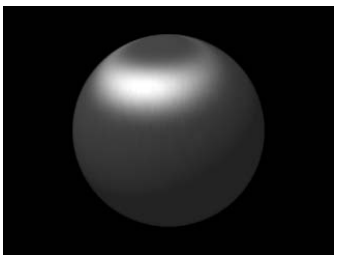
When `projection` is `"st"` or `"ts"`, the `s` and `t` parameters give the 2D parameterization of the surface (and thus the orientation and "shape" of the grain pattern).

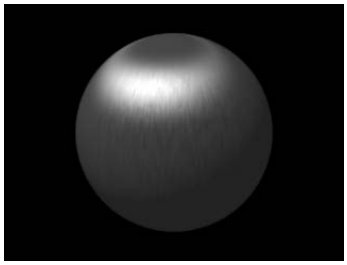
string shadingspace = "shader"
point Pref = P
vector graindir = vector ("shader",0,0,1)

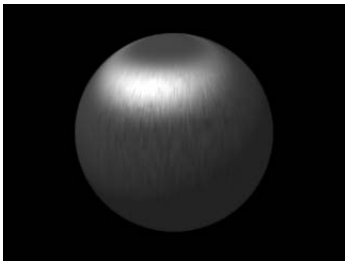
When `projection` is `"Pref"`, the grain and brush orientation are given by `graindir` and aligned to the surface as indicated by transforming the reference mesh `Pref` into the coordinate system named by `shadingspace`.

float grainamp = .75

The amplitude of the "grain", which is really just a stretched-out procedural noise. When `grainamp` is 0, no grain will be visible.

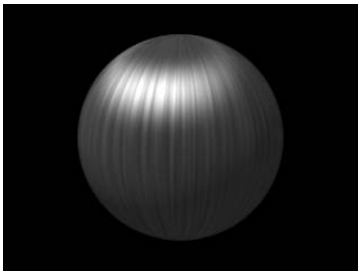

grainamp = 0.25

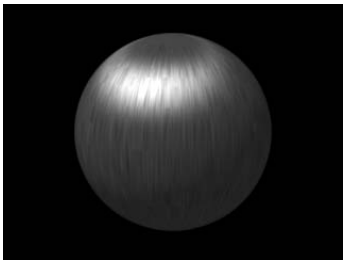

grainamp = 0.75 (default)

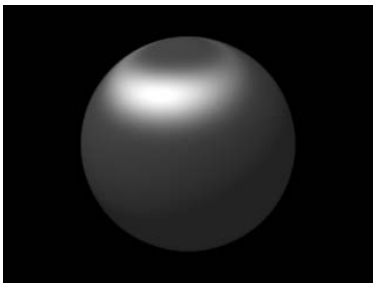

grainamp = 1

float grainfreq = 400

Controls the frequency of the grain. When the frequency is too high, the grain will automatically fade away rather than aliasing.


grainfreq = 25

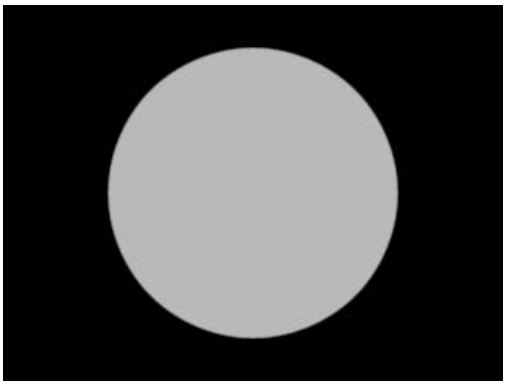

grainfreq = 100


grainfreq = 400

string envname = ""
float Kr = 0.25
float blur = 0
string envspace = "world"
float envrad = 0
float samples = 1
float twosided = 0

These parameters control environmental reflections (including, optionally, ray tracing). They are nearly identical to the their functionality in the `metal` shader.

constant.gsl



Description

The surface shader `constant` makes a material that does not require lights at all, and is simply a constant color.

Parameters

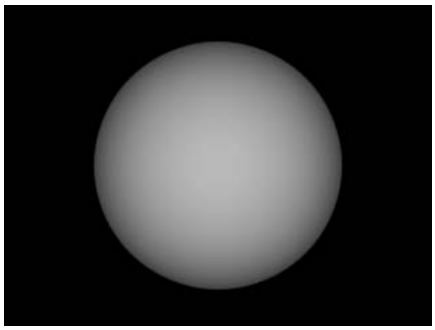
There are no parameters. The surface color will simply be `C`, scaled by the `opacity`.

Attributes and Globals

"color C", (1,1,1)
`C` provides the base color of the material.

"color opacity", (1,1,1)
`opacity` provides the opacity of the material.

defaultsurface.gsl



Description

The surface shader `defaultsurface` makes a material that does not require lights at all, and is brighter when facing the camera.

Parameters

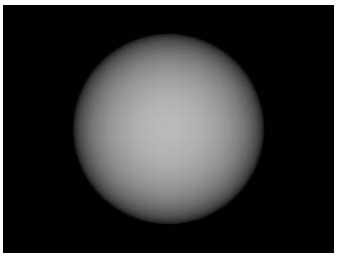
All of the example images below are rendered with the base color `C` green (i.e., `color(0,1,0)`) in order to more clearly distinguish between diffuse and specular color.

float Ka = 1

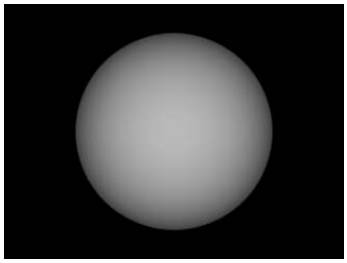
The `Ka` parameter scales the portion of the surface's brightness that is independent of its orientation.

float Kd = 0.9

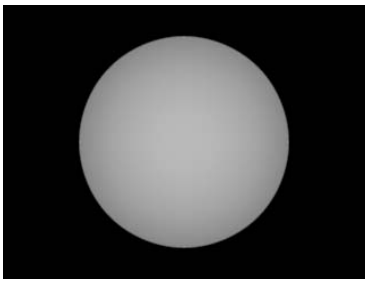
The `Kd` parameter scales the portion of the surface's brightness that varies based on orientation.



Ka = 0, Kd = 1



Ka = 0, Kd = 0.9 (default)



Ka = 0.5, Kd = 0.5

Attributes and Globals

"color C", (1,1,1)

`C` provides the base color of the material.

"color opacity", (1,1,1)
`opacity` provides the opacity of the material.

envsurf.gsl

Description

The surface shader `envsurf` simply looks up from an environment map in the view direction. It is principally used as a "backdrop."

The backdrop geometry itself is presumed to be a sphere surrounding all other objects, or a single Patch that exactly covers the camera view but is behind all other objects in the scene. However, the exact shape of the backdrop does not affect its appearance, which is determined solely by accessing the environment map in the view direction, `I`.

Parameters

string envname = ""

Specifies the name of the environment map file for the background image.

If `envname` is the empty string, no background image will be used.

float blur = 0

Controls the blurriness of the reflections. See the metal.gsl shader for information on blurry environment lookups.

string envspace = "world"

The backdrop image is assumed to have some particular orientation with respect to the scene. This parameter names a coordinate system the describes the orientation of the backdrop. It may be a pre-defined coordinate system such as "world", or it could be the name of a user-named coordinate system (using `SaveAttributes`).

Coordinate systems for `envspace` may be used to orient a captured or generated environment map with the scene. This includes fairly radical re-orientation, such as taking an environment map captured with the assumption that +y is "up" and using it for reflections in a modeled scene in which +z is assumed to be "up".

float envrad = 0

By default the map is assumed to have "infinite" radius -- that is, the backdrop image is looked up from the map based upon only the view direction, without taking camera position into account. This is fine for far-away backdrops (distant mountains), but near backdrops (the walls of a room) may appear unnatural without the usual *parallax* effects.

When `envrad` is non-zero, the backdrop lookup is done with parallax as if the backdrop image was captured the inside of a sphere whose center is at the origin of the `envspace` coordinate system, and whose radius is given by `envrad`.

float Kd = 1

`Kd` simply scales the overall color of the environment map lookup to make it dimmer or brighter.

Attributes and Globals

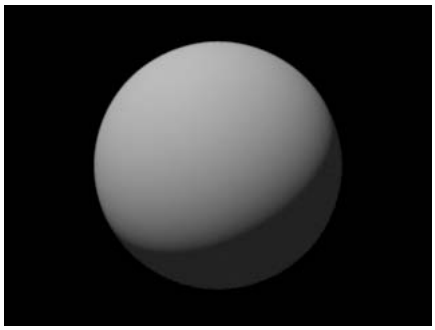
"color C", (1,1,1)

`C` filters the color of the environment lookup, and may be used to "tint" the overall backdrop color.

"color opacity", (1,1,1)

`opacity` provides the opacity of the material, allowing you to see through the backdrop if it is not (1,1,1).

matte.gsl



Description

The surface shader `matte` makes a basic Lambertian material. Lambertian materials scatter light equally in all directions and do not have specular highlights.

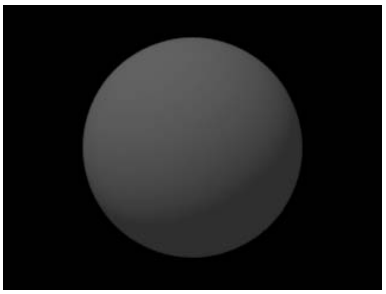
Parameters

float Ka = 1

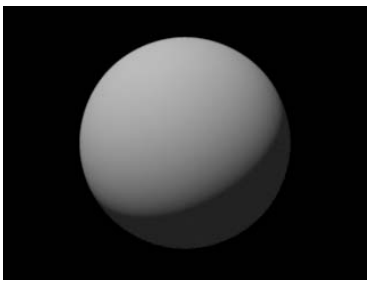
The `Ka` parameter scales the surface's ambient reflectivity (the degree to which it responds to ambient lights).

float Kd = 0.5

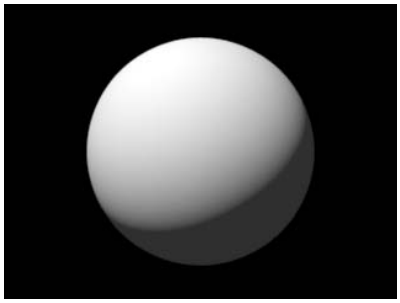
The `Kd` parameter scales the surface's diffuse reflectivity.



Kd = 0.1



Kd = 0.5

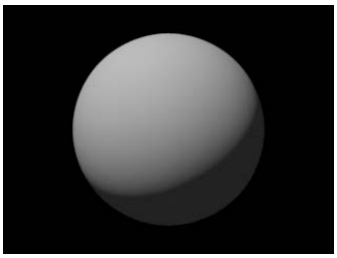


Kd = 1 (default)

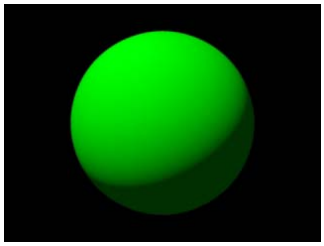
Attributes and Globals

"color C", (1,1,1)

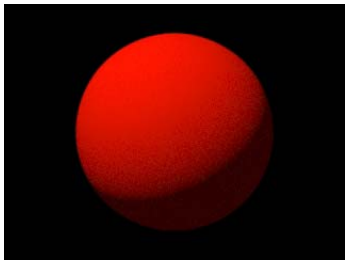
`C` provides the base color of the material.



C = (1,1,1) (default)



C = (0,1,0)



C = (1,0,0)

"color opacity", (1,1,1)

`opacity` provides the opacity of the material.

metal.gsl

Description

The surface shader `metal` makes a basic metal material.

Metals have specular highlights that are tinted the color of the metal (which looks quite different from plastics and other non-metals, which tend to have white highlights regardless of the color of the underlying substrate).

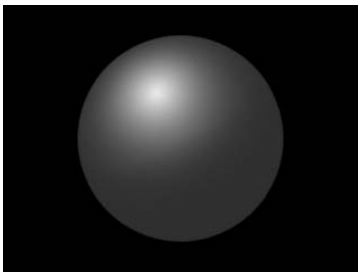
Parameters

float Ka = 1

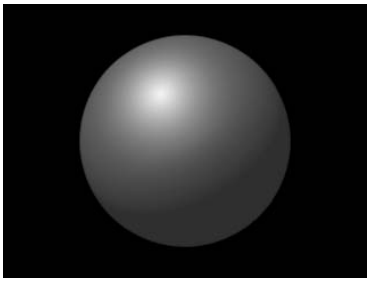
The `Ka` parameter scales the surface's ambient reflectivity (the degree to which it responds to ambient lights).

float Kd = 0.1

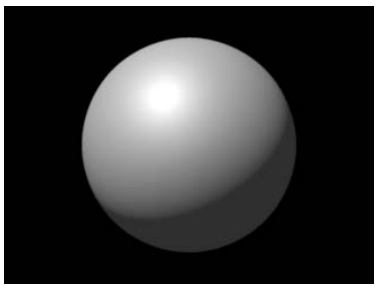
The `Kd` parameter scales the surface's diffuse reflectivity. Metallic objects tend to have very low `Kd` values, which is why the default is only 0.1. The more polished the surface, the lower the `Kd` value should be.



Kd = 0.1



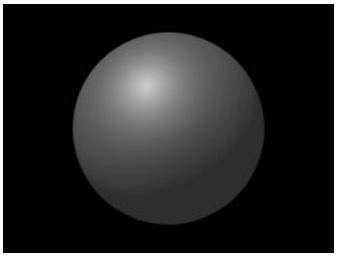
Kd = 0.1 (default)

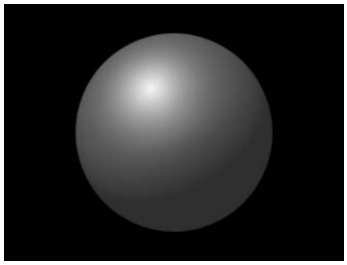


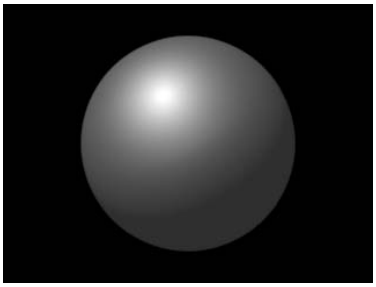
Kd = 0.5

float Ks = 0.8

The `Ks` parameter scales the surface's specular highlights. When Ks = 0, `metal` has no specular highlights at all.

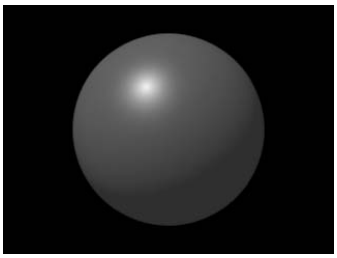

Ks = 0.5

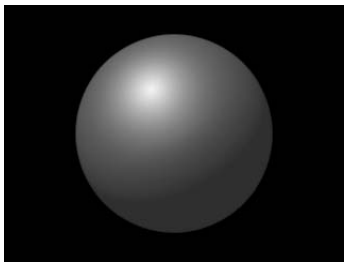

Ks = 0.8 (default)

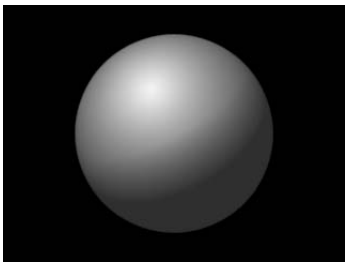

Ks = 1

float roughness = 0.2

The `roughness` parameter controls the sharpness of the surface's specular highlights. It roughly corresponds to microscopic roughness of the surface material. Low-roughness surfaces are smooth and have sharp, small specular highlights. High-roughness surfaces have broad, diffuse highlights.


roughness = 0.1


roughness = 0.2 (default)


roughness = 0.35

string envname = ""

Specifies the source of reflections -- either the name of an environment map, or the name of a geometry set (for ray-traced reflections).

If `envname` is the empty string, no reflections will be present.

float Kr = 0.5

Controls the brightness of the reflections. Larger numbers mean a more mirror-like surface. A `Kr` value of 0 indicates that reflections are not visible.


Kr = 0.1


Kr = 0.5 (default)


Kr = 1

float blur = 0

Controls the blurriness of the reflections.

blur = 0 blur = 0.1 blur = 0.25

string envspace = "world"

When reflections are from an environment map, the map is assumed to have some particular orientation with respect to the scene. This parameter names a coordinate system the describes the orientation of the environmental reflections. It may be a pre-defined coordinate system such as "world", or it could be the name of a user-named coordinate system (using `SaveAttributes`).

Coordinate systems for `envspace` may be used to orient a captured or generated environment map with the scene. This includes fairly radical re-orientation, such as taking an environment map captured with the assumption that +y is "up" and using it for reflections in a modeled scene in which +z is assumed to be "up".

The `envspace` parameter is ignored when using ray-traced reflections.

float envrad = 0

When reflections are from an environment map, by default the map is assumed to have "infinite" radius -- that is, reflections are looked up from the map based upon only the reflected direction, without taking position into account. This tends to make relatively flat objects appear unnatural, and in general does not incorporate a variety of *parallax* effects.

When `envrad` is non-zero, reflections are computed with parallax as if they were on the inside of a sphere whose center is at the origin of the `envspace` coordinate system, and whose radius is given by `envrad`.

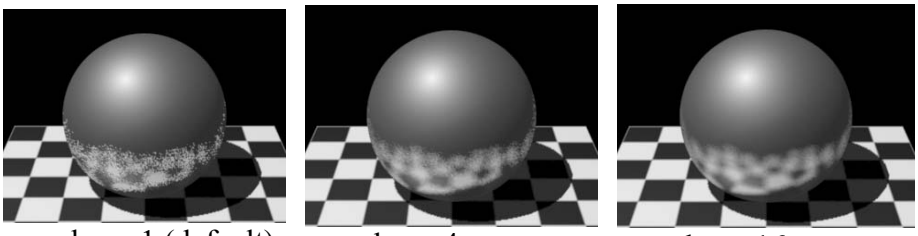
The `envrad` parameter is ignored when using ray-traced reflections.

float samples = 1

When using ray-traced reflections, this controls the number of rays used to simulate blurry reflections. More rays make smoother reflections, but take longer to render. Fewer rays render faster but can look "noisy" for large blurs.

The `samples` parameter is ignored when using environment maps.

The following examples all use ray-traced reflections with `blur = 0.15`.



samples = 1 (default) samples = 4 samples = 16

float twosided = 0

When `twosided = 0` the "backfacing" side does not have reflections. Especially for ray-traced surfaces, this can be a significant speed-up. However, it depends on the surface being "closed" and consistently oriented so that the surface normals always face the "outside." If both sides of the surface are potentially visible and need to appear reflective, you should set `twosided = 1`.

In the example below, a slice has been cut out of the sphere, showing the inside surface:

twosided = 0 (default) twosided = 1

Attributes and Globals

Attribute ("color C", (1,1,1))

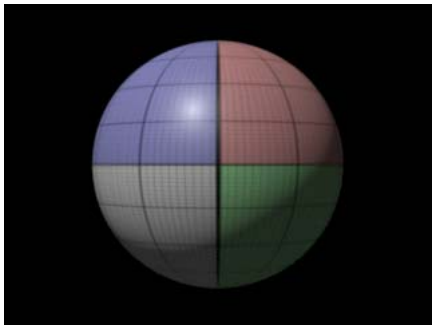
The `C` attribute provides the base color of the metal material. Metals tint their reflections and specular light, as well as diffuse light, by the surface color.

C = (1,1,1) (default) C = (1,1,0) C = (1,0.25,0)

"color opacity", (1,1,1)

`opacity` provides the opacity of the material.

paintedplastic.gsl



Description

The surface shader `paintedplastic` makes a plastic material with a base color determined by texture map. Consult the plastic documentation for the basic non-texture-mapped functionality.

Parameters

float Ka = 1
float Kd = 0.5
float Ks = 0.5
float roughness = 0.1
color specularcolor = color (1,1,1)

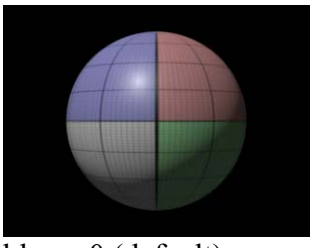
These parameters are identical to their functionality in the plastic shader.

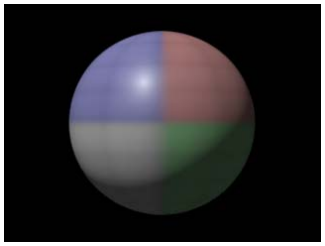
string texturename = ""

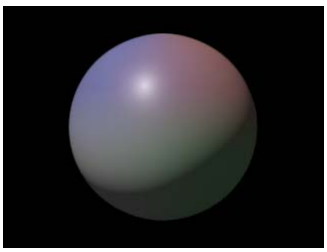
The name of the texture. If the texture name is the empty string (the default), no texture mapping will be used, reducing `paintedplastic.gsl` to the same behavior as ordinary plastic.

float blur = 0

Blurs the texture lookup. A value of 0 is a perfectly sharp texture lookup. A value of 1 indicates that the blur kernel is as wide as the entire texture.


blur = 0 (default)


blur = 0.015

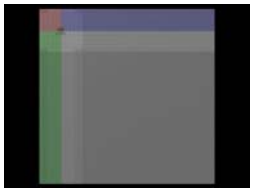

blur = 0.15

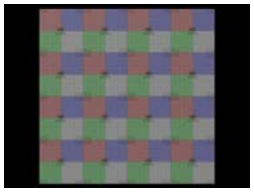
string wrap = "default"

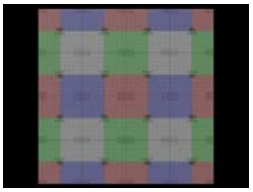
Controls the appearance of the texture when s,t are outside the [0,1] range. Valid values are "black", "clamp", "periodic", and "mirror". If the value is "default", then the wrap mode will be whatever wrap mode is specified in the texture file itself (specified on the command line of `maketx`).

In the example below, the grid texture is placed on a patch that varies from 0-4 in each direction.


wrap = "black"


wrap = "clamp"


wrap = "periodic"


wrap = "mirror"

Separate wrap modes may be used for the two directions, merely by separating two wrap modes by a comma, for example `"black,periodic"` will result in the texture being black when s < 0 or s > 1, but will be periodic for t values outside the [0,1] range.

float s = u
float t = v

The s and t parameters give the 2D texture lookup coordinates. By default, they are equal the the u and v parameters of the surface, respectively. But of course they may be re-mapped as primitive variables (with new "s" and "t" values per vertex of the geometry), or may be computed by another shader layer and joined to this shader using `ConnectShaders`.

Attributes and Globals

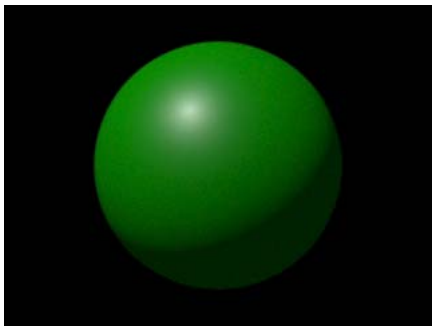
"color C", (1,1,1)

`C` provides the base color of the plastic material, which will act as a filter on the texture value.

"color opacity", (1,1,1)

`opacity` provides the opacity of the material.

plastic.gsl



Description

The surface shader `plastic` makes a basic plastic material.

Actual plastics (and other non-metals) have white specular highlights, regardless of the diffuse color of the material. This is because plastic is usually a clear substrate with embedded pigment particles. Specular reflection takes place at the substrate-air boundary, and thus the pigment doesn't come into play. Diffuse reflection involves light scattered by the pigment itself, and is thus colored.

By giving controls over specularity, surface roughness, and specular highlight color, the `plastic` shader may be used for a wide variety of non-metallic surfaces, not merely manufactured plastic.

Parameters

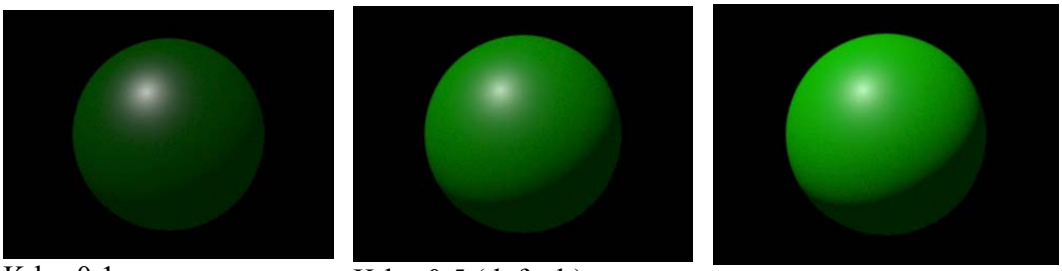
All of the example images below are rendered with the base color `C` green (i.e., `color(0,1,0)`) in order to more clearly distinguish between diffuse and specular color.

float Ka = 1

The `Ka` parameter scales the surface's ambient reflectivity (the degree to which it responds to ambient lights).

float Kd = 0.5

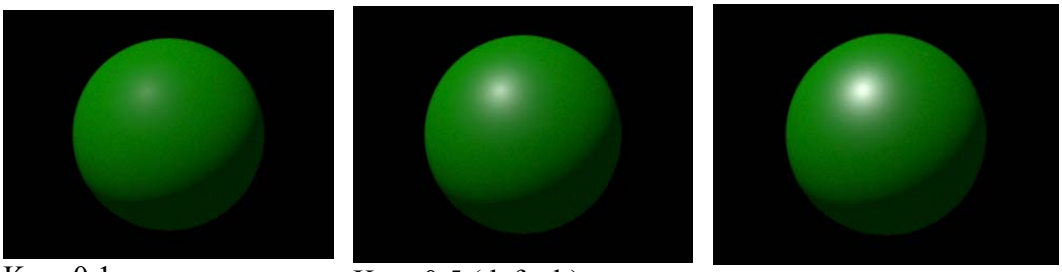
The `Kd` parameter scales the surface's diffuse reflectivity.



Kd = 0.1 | Kd = 0.5 (default) | Kd = 1

float Ks = 0.5

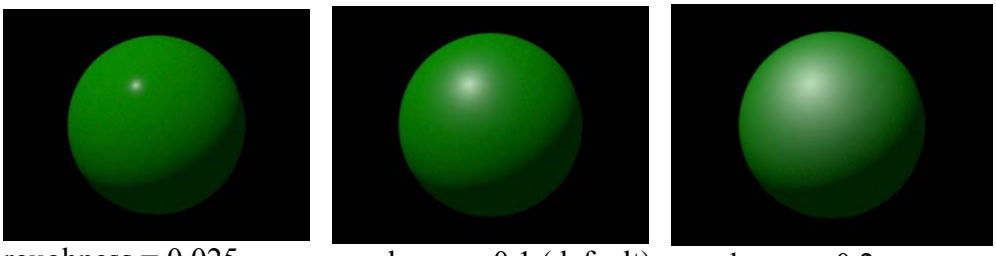
The `Ks` parameter scales the surface's specular highlights. When Ks = 0, `plastic` behaves exactly like `matte`.



Ks = 0.1 | Ks = 0.5 (default) | Ks = 1

float roughness = 0.1

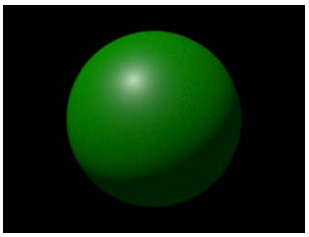
The `roughness` parameter controls the sharpness of the surface's specular highlights. It roughly corresponds to microscopic roughness of the surface material. Low-roughness surfaces are smooth and have sharp, small specular highlights. High-roughness surfaces have broad, diffuse highlights.



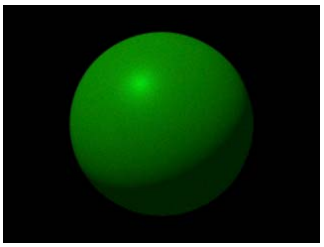
roughness = 0.025 | roughness = 0.1 (default) | roughness = 0.2

color specularcolor = color (1,1,1)

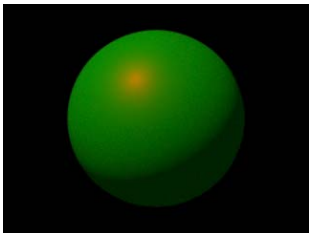
`specularcolor` gives the color of the specular highlight. Actual plastic (and other non-metals) have white specular highlights, regardless of the diffuse color of the material. In contrast, metals tend to have a specular highlight color that is the color of the metal itself.



specularcolor = (1,1,1) (default)



specularcolor = (0,1,0)

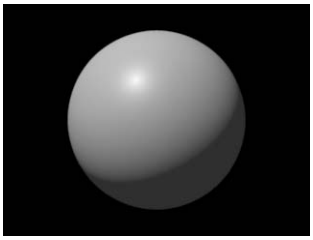


specularcolor = (1,0,0)

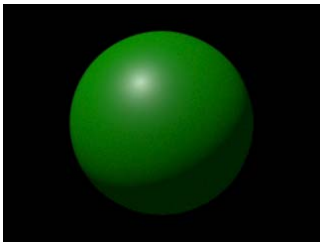
Attributes and Globals

"color C", (1,1,1)

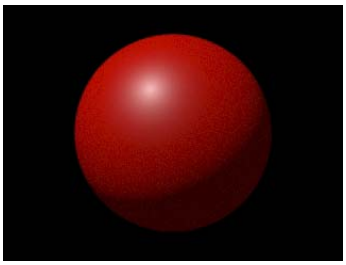
`C` provides the base color of the plastic material.



C = (1,1,1) (default)



C = (0,1,0)



C = (1,0,0)

"color opacity", (1,1,1)

`opacity` provides the opacity of the material.

plasticss.gsl

Description

The surface shader `plasticss` makes a plastic material that incorporates subsurface scattering. Consult the plastic documentation for the basic non-subsurface functionality.

Parameters

float Ka = 1
float Kd = 0.5
float Ks = 0.5
float roughness = 0.1
color specularcolor = color (1,1,1)

These parameters are identical to their functionality in the plastic shader.

string diffusefile = ""

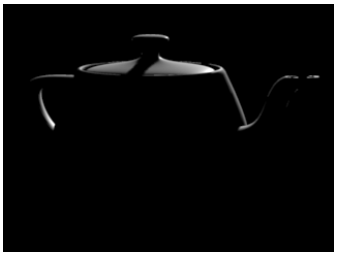
The filename of the spatial database (SDB) containing the baked diffuse illumination from the first pass.

string space = "world"

The name of the coordinate system in which the diffuse data are stored. This should almost never be changed.

float Kss = 1

Controls the overall subsurface contribution. When Kss = 0, there is no subsurface scattering at all.


Kss = 0


Kss = 0.5


Kss = 1 (default)

color subsurfcolor = color(1,1,1)

Filters the overall color of the subsurface glow. Note, however, that this is just a simple color filter, and is not a physically accurate model of how tinted material behaves. In a real tinted material, the light becomes more colored the farther it travels through medium, and this parameter is just a color shift that doesn't take distance into consideration. For more physically-accurate coloring, you should be adjusting the `scattering` and `absorption` parameters.


subsurfcolor = (1,1,1)


subsurfcolor = (1, 0.25, 0.25)

color scattering = color (2.19, 2.62, 3.0)

Reduced scattering coefficients (sigma_s), indicates the rate of scattering (per common space unit). The default value is a physically measured value for marble, expressed in 1/mm.

Here is an example where all components of `scattering` are multiplied or divided by 5:


scattering = (.438, 0.524, 0.6)


scattering = (2.19, 2.62, 3.0) (default)


scattering = (10.95, 13.1, 15)

Here are examples where just the red component of scattering is doubled or halved:


scattering = (1.09, 2.62, 3.0)
(lower red scattering)


scattering = (2.19, 2.62, 3.0)
(default)


scattering = (4.38, 2.62, 3.0)
(higher red scattering)

color absorption = color (.0021, .0041, .0071)

Absorption coefficients of the material (per common space unit). The default value is a physically measured value for marble, expressed in 1/mm.

Here is an example where all components of `absorption` are multiplied or divided by 5:


absorption = (.00042, .00082, .00142)


absorption = (.0021, .0041, .0071)
(default)


absorption = (.0105, .0205, .0355)

Here is an example where just the red component of `absorption` is doubled or halved:

absorption = (.00105, .0041, .0071)
(lower red absorption)

absorption = (.0021, .0041, .0071)
(default)

absorption = (.0042, .0041, .0071)
(higher red absorption)

string ssunits = "mm"

Units in which `scattering` and `absorption` are measured. Most published physical units for these parameters are in 1/mm, so "mm" is generally appropriate. If your scene contains appropriate unit designations (via `Attribute("units:length")`), this will ensure that

float ssscale = 1

Overall scaling of the unit system used for the scattering and absorption parameters. For example, if your scene does not have unit designations and your coefficients are in 1/mm but your scene is modeled in cm, you want scale=10.

Even if your scene has proper unit designations, it may be very useful to use scale simply to adjust scattering and absorption simultaneously, in a somewhat nonphysical but very user-friendly manner to change the overall transparency of the material. In short, raising `scale` makes the material more dense (impervious to subsurface light), while lowering `scale` makes the material more transparent to suburface light.

ssscale = 0.1

ssscale = 0.5

ssscale = 2

float eta = 1.5

Index of refraction of the material. The default value is a physically measured value for marble.

eta = 1.2

eta = 1.5 (default)

eta = 1.8

float twosided = 0

When nonzero, the subsurface effect can be seen from both sides of the material. When zero, it can only be seen from the side in which the normals point.

Setting `twosided = 0` can render faster under many conditions, but is only appropriate if the model is carefully constructed so that normals are always outward-facing. If normals are backwards and twosided is zero, you will not see the subsurface scattering effect.

float maxsolidangle = 1

An error metric for the hierarchical subsurface computation (bigger is faster, but less accurate). You should never need to change this value.

Attributes and Globals

"color C", (1,1,1)

`C` provides the base color of the plastic material. its functionality is identical to how `C` behaves in the plastic shader.

"color opacity", (1,1,1)

`opacity` provides the opacity of the material.

Tips and Sample Settings

Here are a few sample settings for `scattering`, `absorption`, and `eta` that correspond to measured values of real materials. For more sample parameters, see Jensen, et al, "A Practical Model for Subsurface Light Transport," ACM SIGGRAPH 2001.

"marble"
scattering = (2.19, 2.62, 3.0)
absorption = (.00105, .0041, .0071)
eta = 1.5

"skin"
scattering = (0.74, 0.88, 1.01)
absorption = (0.032, 0.17, 0.48)
eta = 1.3

"cream"
scattering = (7.38, 5.47, 3.15)
absorption = (0.0002, 0.0028, 0.0163)
eta = 1.3

"skim milk"
scattering = (0.7, 1.22, 1.9)
absorption = (0.0014, 0.0025, 0.0142)
eta = 1.3

shinyplastic.gsl

Description

The surface shader `shinyplastic` makes a plastic material with reflections. Consult the plastic documentation for the basic non-reflective functionality.

Parameters

float Ka = 1
float Kd = 0.5
float Ks = 0.5
float roughness = 0.1
color specularcolor = color (1,1,1)

These parameters are identical to their functionality in the plastic shader.

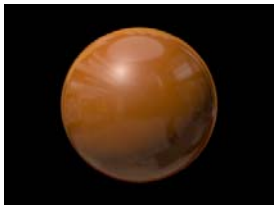
string envname = ""
float Kr = 1
float blur = 0
string envspace = "world"
float envrad = 0
float samples = 1
float twosided = 0

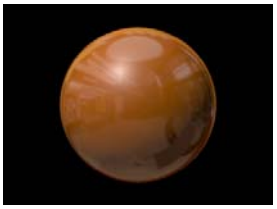
These parameters are identical to their functionality in the metal shader.

float eta = 1.5

Controls the index of refraction of the material. For most dielectrics such as plastic, a value near 1.5 is reasonable. Changing the index of refraction also indirectly changes the apparent reflectivity of the material and the degree to which reflectivity is stronger at grazing angles.


eta = 1.25


eta = 1.75


eta = 2

Attributes and Globals

"color C", (1,1,1)

`C` provides the base color of the plastic material. its functionality is identical to how `C` behaves in the plastic shader.

"color opacity", (1,1,1)

`opacity` provides the opacity of the material.

Surface Layers

checker.gsl

Description

The layer `checker` makes a basic checkerboard pattern. The results are put in outputs `Cout` and `fout`. No lighting is computed in this shader; it is intended to be used to generate checker patterns as input for other layers.

As an example, note the following sequence of layers that makes a plastic material whose color is given the checker pattern, and furthermore has only half of the tiles specularly reflecting:

```
ShaderGroupBegin ()
Shader ("surface", "checker", "checklayer")
Shader ("surface", "plastic", "plasticlayer")
ConnectShaders ("checklayer", "Cout", "plasticlayer", "C")
ConnectShaders ("checklayer", "fout", "plasticlayer", "Ks")
ShaderGroupEnd ()
```

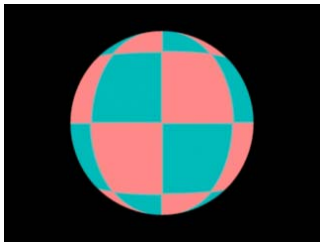
Parameters

color color1 = C
color color2 = 0

Controls the color of the tiles.

The default of `color1`, so if no value is passed for `color1` at all, `color1` will be the previous surface color, `C`.

color1 = (1,1,1)
color2 = (0,0,0) (default)

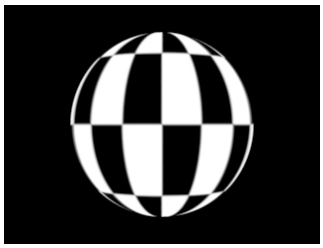


color1 = (1,.25,.25)
color2 = (0,.5,.5)

float stilesize = 1
float ttilesize = 1

Controls the size of the tiles, in units of `s` and `t`, respectively.

stilesize = 1
ttilesize = 1 (default)



stilesize = 0.5
ttilesize = 1

stilesize = 1
ttilesize = 0.5

float s = u
float t = v

The s and t parameters give the 2D texture lookup coordinates. By default, they are equal the the u and v parameters of the surface, respectively. But of course they may be re-mapped as primitive variables (with new "s" and "t" values per vertex of the geometry), or may be computed by another shader layer and joined to this shader using `ConnectShaders`.

Outputs

output color Cout

The color of the checker pattern -- either `color1` or `color2`. It can also take on intermediate values to antialias the boundaries between the checker tiles.

output float fout

A float containing 0 when inside the tiles with `color1`, 1 when inside the tiles with `color2`, and intermediate values to antialias the edges.

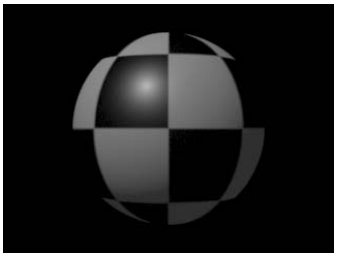
postgloss.gsl

Description

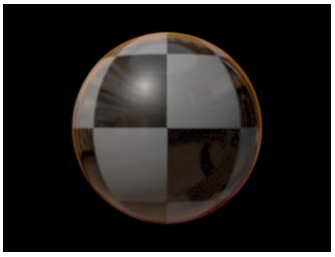
The surface layer `postgloss` *adds* reflective gloss to the surface color `C` of previous layers. The effect is similar to isolating just the reflections of shinyplastic.

The purpose of `postgloss` is to add reflections to another shader or group of shaders, without needing to alter the other shaders (or even have access to their source code). For example, the following uses `plastic` and `postgloss` to make a material that looks identical to shinyplastic:

```
    ShaderGroupBegin ()
    Shader ("surface", "plastic", "plasticlayer")
    Shader ("surface", "postgloss", "glosslayer", "string envname",
"office.env")
    ShaderGroupEnd ()
```



checkered plastic



postgloss over checkered plastic

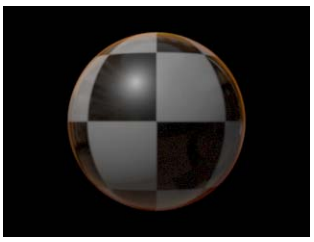
Parameters

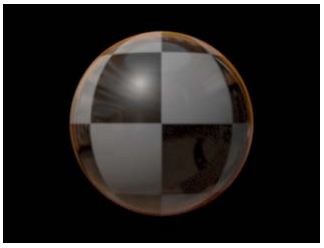
string envname = ""
float Kr = 1
float blur = 0
string envspace = "world"
float envrad = 0
float samples = 1
float twosided = 0

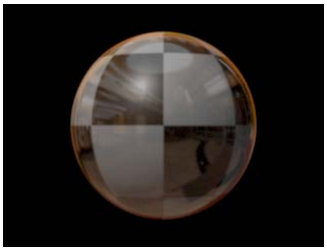
These parameters are similar to their functionality in the metal shader.

float eta = 1.5

Controls the index of refraction of the material. For most dielectrics such as plastic, a value near 1.5 is reasonable. Changing the index of refraction also indirectly changes the apparent reflectivity of the material and the degree to which reflectivity is stronger at grazing angles.


eta = 1.25


eta = 1.5 (default)


eta = 2

float use_fresnel = 1

When nonzero, uses Fresnel attenuation, which causes reflectivity to be higher at grazing angles, such as is the case for plastics and other dielectrics. When zero, reflectivity will be independent of angle, more like metals behave.


use_fresnel = 1 (default), Kr = 1

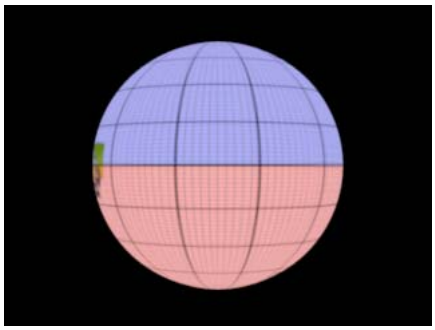

use_fresnel = 0, Kr = 0.25

Attributes and Globals

"color C", (1,1,1)

`C` provides the *lit* base color of the material. The `postgloss` will attenuate `C` according to transmission and add reflections to `C`.

pretexture.gsl



Description

The surface layer `pretexture` filters `C` by a color lookup from a texture map. It does not perform any lighting calculations, and is thus designed specifically to be used as a layer prior to another layer that performs lighting calculations.

As an example, note the following sequence of layers that colors `C` based on a texture lookup and then uses a plastic illumination model. It should give a result identical to using `paintedplastic`:

```
ShaderGroupBegin ()
Shader ("surface", "pretexture", "string filename", "grid.tx")
Shader ("surface", "plastic")
ShaderGroupEnd ()
```

Note that there is no need to connect the layers via `ConnectShaders` since `pretexture` modifies `C` and `plastic` uses `C` as the base color of the material.

Parameters

string texturename = ""
float blur = 0
string wrap = "default"

These parameters are identical to their functionality in the paintedplastic shader.

float s = u
float t = v

The s and t parameters give the 2D texture lookup coordinates. By default, they are equal the the u and v parameters of the surface, respectively. But of course they may be re-mapped as primitive variables (with new "s" and "t" values per vertex of the geometry), or may be computed by another shader layer and joined to this shader using `ConnectShaders`.

string projection = "st"

This parameter is currently unused and is reserved for future functionality, where it would be used to determine a texture projection other than simply using `s` and `t`.

Attributes and Globals

"color C", (1,1,1)

`C` provides the base color of the material, The `pretexture` layer *multiples* the prior layer's `C` value by the texture lookup.

subsurflayer.gsl

Description

The surface layer `subsurflayer` *adds* subsurface scattering to an existing material (after ordinary lighting has been applied in an earlier layer).

Consult the plasticss documentation for the basic subsurface scattering functionality.

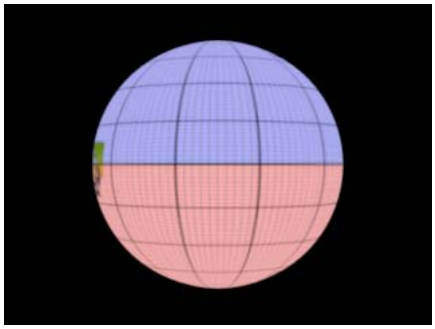
Note that layering `plastic` followed by `subsurflayer` is equivalent to simply using `plasticss`. But `subsurflayer` can be very useful to add subsurface scattering to a shader (or shader group) that doesn't already incorporte subsurface scattering effects, and for which you may not have access to the shader source code.

Parameters

string diffusefile = ""
string space = "world"
float Kss = 1
color subsurfcolor = color(1,1,1)
color scattering = color (2.19, 2.62, 3.0)
color absorption = color (.0021, .0041, .0071)
string ssunits = "mm"
float ssscale = 1
float eta = 1.5
float twosided = 0
float maxsolidangle = 1

These parameters are identical to their functionality in the plasticss.gsl shader.

texmap.gsl



Description

The surface layer `texmap` performs a lookup from a texture map and stores the results in its outputs `Cout`, `fout`, and `aout`. It does not perform any lighting calculations, and is thus designed specifically to be used for its outputs to be connected to the input(s) of a subsequent shader layer.

As an example, note the following sequence of layers that produces a plastic material where the base surface color `C` is based on the texture map lookup.It should give a result identical to using `paintedplastic`:

```
    ShaderGroupBegin ()
    Shader ("surface", "texmap", "texlayer", "string filename",
"grid.tx")
    Shader ("surface", "plastic", "plasticlayer")
    ConnectShaders ("texlayer", "Cout", "plasticlayer", "C")
    ShaderGroupEnd ()
```

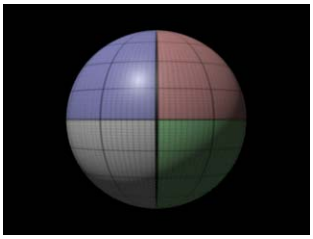
Parameters

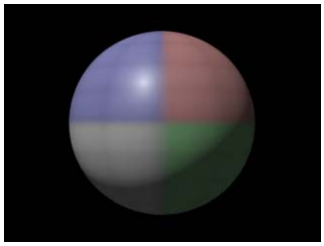
string texturename = ""

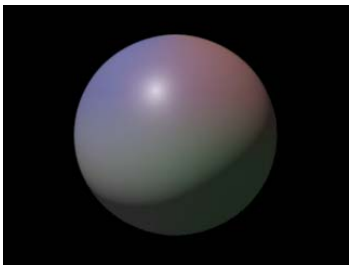
The name of the texture. If the texture name is the empty string (the default), no texture mapping will be used and all outputs will be 0.

float blur = 0

Blurs the texture lookup. A value of 0 is a perfectly sharp texture lookup. A value of 1 indicates that the blur kernel is as wide as the entire texture.


blur = 0 (default)


blur = 0.015

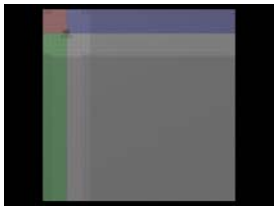

blur = 0.15

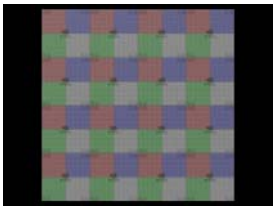
string wrap = "default"

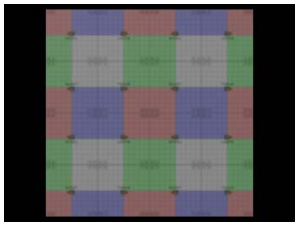
Controls the appearance of the texture when s,t are outside the [0,1] range. Valid values are "black", "clamp", "periodic", and "mirror". If the value is "default", then the wrap mode will be whatever wrap mode is specified in the texture file itself (specified on the command line of `maketx`).

In the example below, the grid texture is placed on a patch that varies from 0-4 in each direction.


wrap = "black"


wrap = "clamp"


wrap = "periodic"


wrap = "mirror"

Separate wrap modes may be used for the two directions, merely by separating two wrap modes by a comma, for example `"black,periodic"` will result in the texture being black when s < 0 or s > 1, but will be periodic for t values outside the [0,1] range.

float firstchannel = 0

By default, the texture lookup starts with the first channel of the map. This can be adjusted using the `firstchannel` parameter.

float alphachannel = 3

The channel of the texture that will get stored in output `aout`. When `alphachannel` is < 0, output `aout` will get the same value as `fout`.

float s = u
float t = v

The s and t parameters give the 2D texture lookup coordinates. By default, they are equal the the u and v parameters of the surface, respectively. But of course they may be re-mapped as primitive variables (with new "s" and "t" values per vertex of the geometry), or may be computed by another shader layer and joined to this shader using `ConnectShaders`.

Outputs

output color Cout

`Cout` will get the first three channels of the texture lookup, starting with channel `firstchannel`.

output float fout

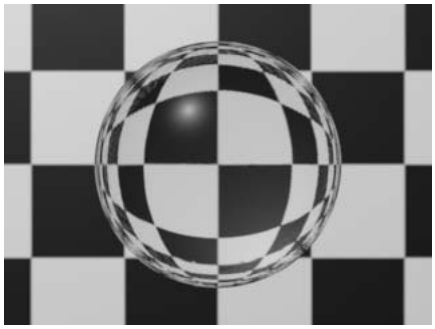
`fout` will get the first channel of the texture lookup, starting with channel `firstchannel`. Note that this is identical to `Cout[0]`.

output color aout

If `alphachannel` is nonnegative, `aout` will get the channel of the texture lookup named by `alphachannel`. If `alphachannel` is negative (the default), `aout` will get the same value as `fout`.

Complex Surfaces

glass.gsl



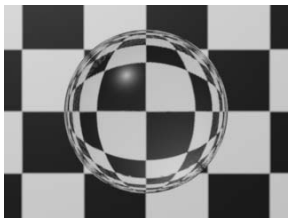
Description

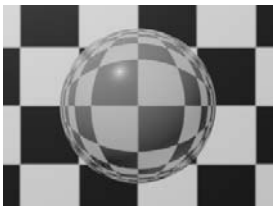
The surface shader `glass` makes a glass material with reflections and refractions.

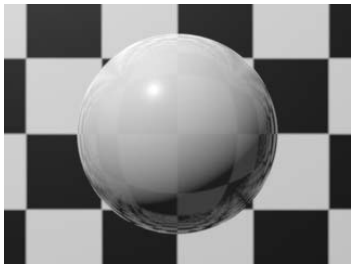
Parameters

float Ka = 0.2
Scaling for ambient lighting.

float Kd = 0
Ordinarily, glass should have no, or very little, diffuse reflectivity. However, for colored or "milky" glass, you may want some diffuse reflection. You probably also will want to reduce `Kt` so that there is less transmission when there is more diffuse reflectivity, making sure that they do not sum to more than 1.

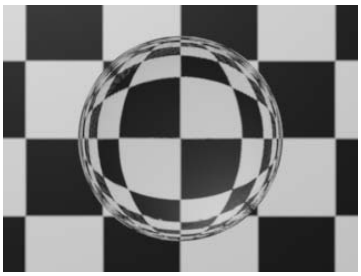

Kd = 0, Kt = 1 (default)

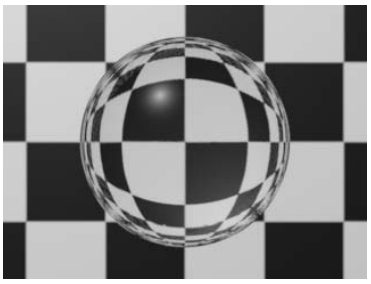

Kd = 0.25, Kd = 0.75

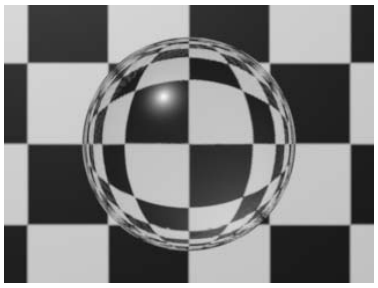

Kd = 0.75, Kd = 0.25

float Ks = 0.5

Controls the brightness of the specular highlight.


Ks = 0


Ks = 0.5 (default)


Ks = 1

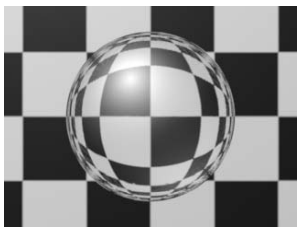
float roughness = 0.05

Controls the size/sharpness of the specular highlight. Small values make small, sharp specular highlights; large values make broad, soft specular highlights.


roughness = 0.025


roughness = 0.05 (default)


roughness = 0.1


roughness = 0.2

string envname = ""

Specifies the source of reflections and refractions -- either the name of an environment map, or the name of a geometry set (for ray-traced reflections/refractions).

If `envname` is the empty string, no reflections or refractions will be present.

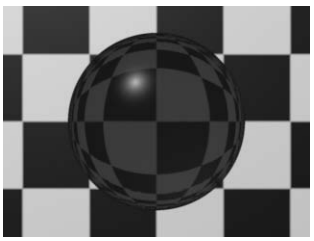
string envspace = "world"
float envrad = 0

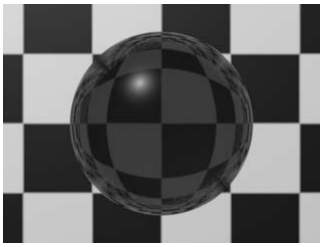
When reflections and refractions are from an environment map, these parameters give the position/orientation and apparent radius of the environment map, as described in `metal`.

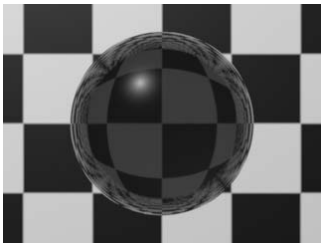
These parameters are not used if ray tracing is used for reflections and refractions.

float Kr = 1

Scales the reflectivity of the glass. Note that this isn't a simple scale of reflected rays, but rather this scales the physically-based Fresnel reflection term. So reflection is much dimmer than, for example, metal with a similar Kr value, and exhibits the correct Fresnel effect of having stronger reflections at grazing angles. For these examples, Kt = 0.25 so the reflections will be more clear.


Kr = 0

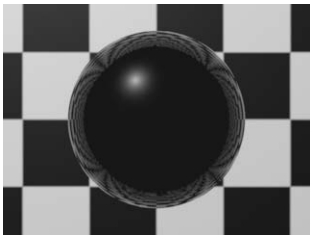

Kr = 0.5

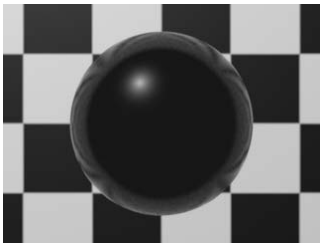

Kr = 1 (default)

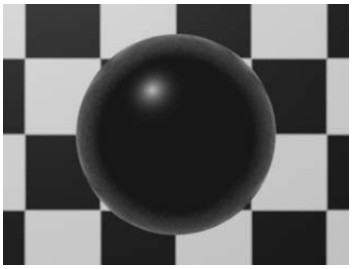
float reflblur = 0

The amount of blur for reflections.

In the examples below, Kt = 0 to make the reflections more easily visible, and `samples` = 16:


reflblur = 0 (default)

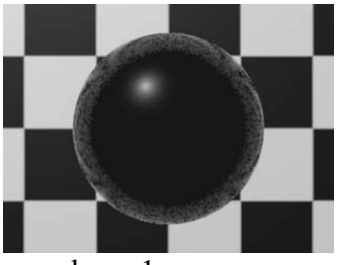

reflblur = 0.05

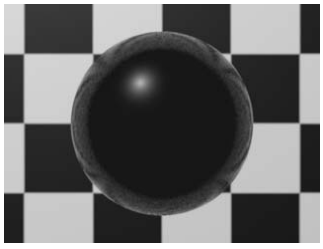

reflblur = 0.15

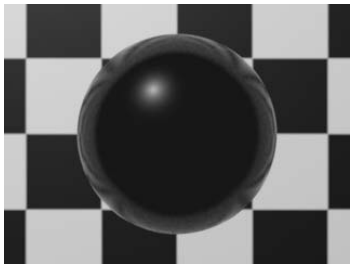
float samples = 1

The number of rays to cast for reflections. More rays are more expensive, but improve image quality by decreasing noise visible in blurry reflections.

In the examples below, Kt = 0 to make the reflections more easily visible, and `reflblur` = 0.05:

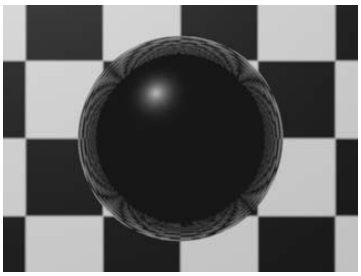

samples = 1

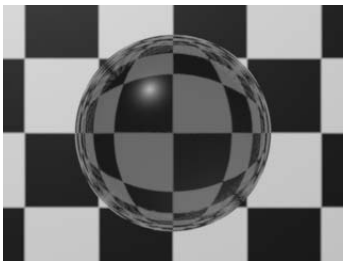

samples = 4

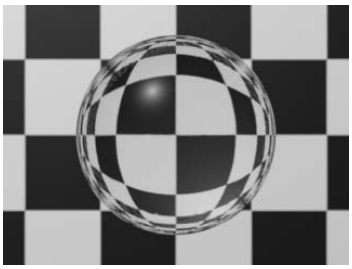

samples = 16

float Kt = 1

The amount of transmission through the glass.


Kt = 0


Kt = 0.5

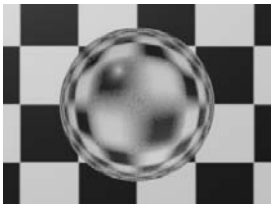

Kt = 1 (default)

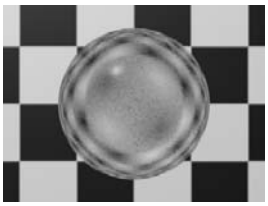
float refrblur = 0

Controls the blurriness of the refraction.


refrblur = 0
(default)


refrblur = 0.02

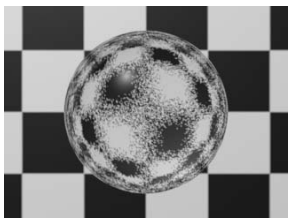

refrblur = 0.1

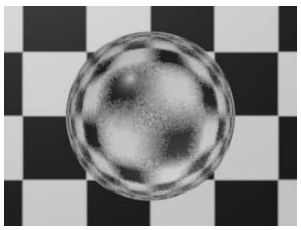

refrblur = 0.2

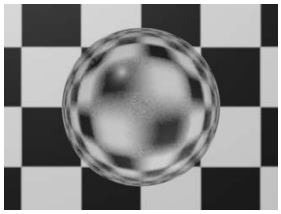
(In the examples above, refrsamples = 16.)

float refrsamples = 1

The number of reflection ray samples. More samples takes longer to render but improves the image especially for large values of `refrblur`.


refrsamples = 1 (default)


refrsamples = 4

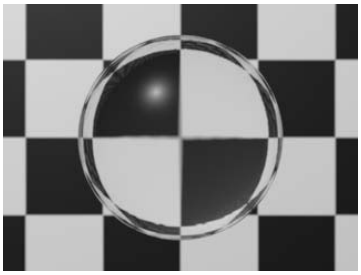

refrsamples = 16

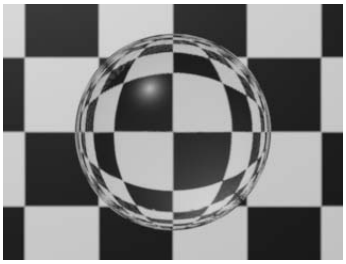
(In the examples above, refrblur = 0.1.)

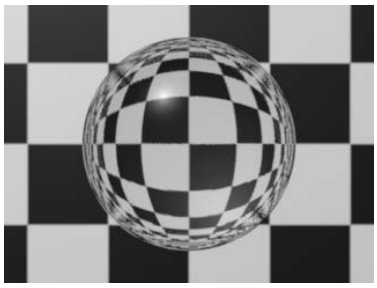
float eta = 1.5

Controls the index of refraction of the material -- that is, the degree to which it bends refracted light. A value of 1 would tend not to bend the light at all. Larger values bend the light more. For ordinary window glass, a value near 1.5 is physically accurate. Water has a value of about 1.33. Various gems have indices of refraction higher than glass (for example, diamonds have an index of refraction

of 2.4175). Changing the index of refraction also indirectly changes the apparent reflectivity of the material and the degree to which reflectivity is stronger at grazing angles.

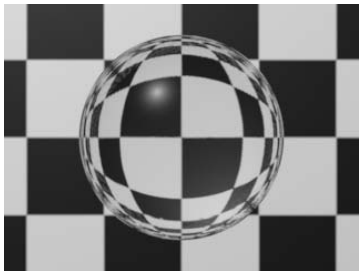

eta = 1.25

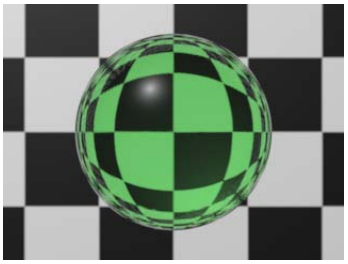

eta = 1.5 (default)


eta = 1.75

color transmitcolor = (1,1,1)

Filtering of the color as it is refracted through the material. Note that this filter is applied identically to all refracted rays, and is not dependent upon angle or the amount of the glass material that the refracted ray travels through (for an example of the latter, see the `extinction` parameter).


transmitcolor = (1,1,1) (default)

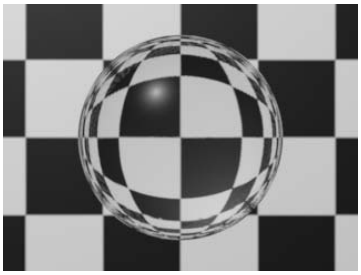

transmitcolor = (.5, 1, .5)

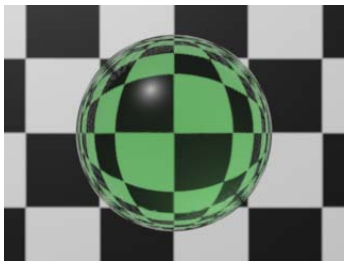
color extinction = (0,0,0)

If `extinction` is 0 (the default), refracted rays will not be attenuated by the specific distance they pass through the material (that is, how far the ray travels after refracting before it comes out "on the other side").

If `extinction` is nonzero, this parameter describes how rapidly each channel (red, green, blue) diminishes as it travels through the material. A value of 0 means it never diminishes at all. Larger values indicate that the light diminishes rapidly as it moves through thicker slabs of the material.

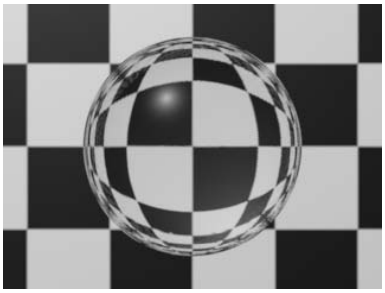
Note an important difference between `extinction` and `transmitcolor` -- the former will diminish light more when it travels through more glass, whereas the latter scales all refracted light identically. The difference is very apparent for objects that have both thick and thin sections, so it is visible that the thick sections diminish light more than thin sections (and for colored glass, thick sections tint the light more than thin sections).

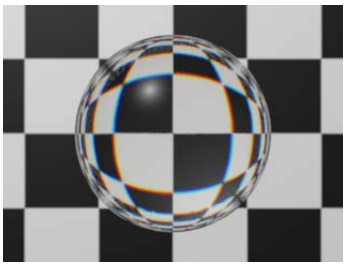

extinction = (0,0,0) (default)


extinction = (.75, .125, .75)

float dispersive = 0

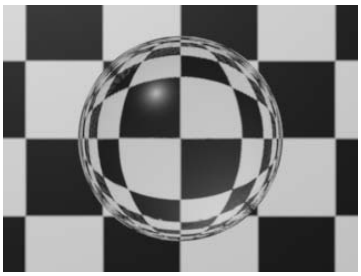
When nonzero, accounts for *dispersive* refraction, that is, the tendency of real glass to bend red light light less than blue light. This has the effect of adding rainbow colored fringes to edges in the refracted image. Larger values for `dispersive` make wider fringes. Note that using dispersive refraction is much more expensive than forgoing dispersion. The default is not to use dispersive refraction.

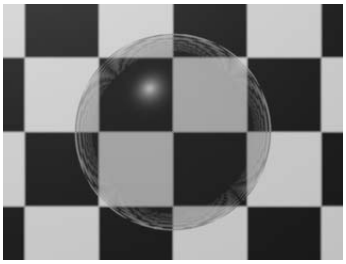

dispersive = 0 (default)


dispersive = 5

float fakerefract = 0

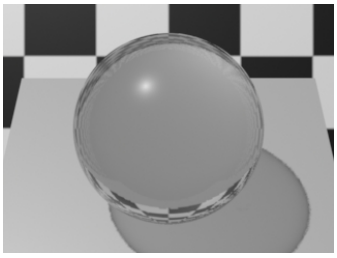
When zero, ray tracing (or environment mapping, depending on the name passed as the `"envname"` parameter) will be used for refraction. When nonzero, no ray tracing environment lookup will performed, but rather refraction will be "faked" by using the `opacity` to simply allow object in the background to show through.


fakerefract = 0 (default)

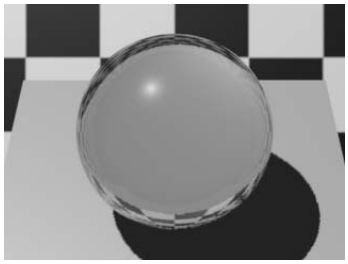

fakerefract = 1

color shadowcolor = (1,1,1)

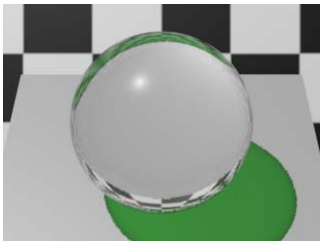
This parameter scales the color of the shadow cast by the material. respect to ray-traced shadows. When white (the default), the apparent opacity to the shadow ray will be computed using a simple Fresnel effect (and also properly scaled by `Kt` and `transmitcolor`. When (0,0,0), the object will be opaque to shadow rays, casting a completely black shadows. Values between zero and one will interpolate these two strategies. Colored shadows can also be achieved by adjusting this color parameter.



shadowcolor = (1,1,1) (default)



shadowcolor = (0,0,0)



shadowcolor = (.25, .75, .25)

string bgenvname = ""
string bgenvspace = "world"
float bgenvrad = 0

These parameters allow you to specify an environment map (including the usual position/orientation in `bgenvspace` and radius in `bgenvrad`) that will be used as a *background* for reflection or refraction rays that do not hit any objects (including rays deeper than the `"ray:maxdepth"` limit).

The reason this is helpful is to reduce the amount of geometry actually ray traced (and therefore reducing computational time and memory). "Distant" objects may be rendered into an environment map, while near objects are ray traced.

Alternately, this can be used to speed up ray tracing of a glass object of extremely complex shape (in which refraction might be expected to to through dozens of layers before emerging on the other side) by purposely setting `"ray:maxdepth"` fairly low (say, 4) and relying on the background environment map for the appearance of deeper refractions. This much less expensive approach can look just as convincing as a "correct" ray tracing of an arbitrarily number of refractions through the complex object.

float twosided = 0

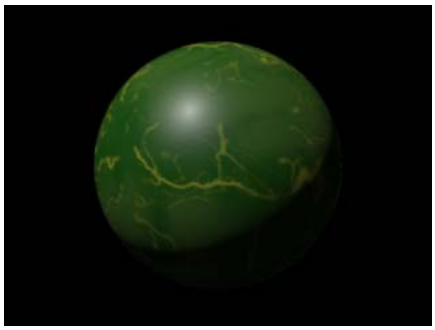
If nonzero, this parameter causes the back side of objects (as visible to the camera) not to trace any rays. For objects that are *closed* and known to have their normals consistently facing the "outside," this can speed up rendering without changing the appearance at all. But it is only safe for those objects.

Attributes and Globals

"color C", (1,1,1)

`C` provides the base color of the diffuse illumination (but does not affect specular, reflection, or refraction).

greenmarble.gsl



Description

The surface shader `greenmarble` makes a solid green veined marble material.

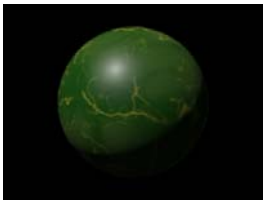
Parameters

float Ka = 0.1
float Kd = 0.6
float Ks = 0.4
float roughness = 0.1
color specularcolor = color (1,1,1)

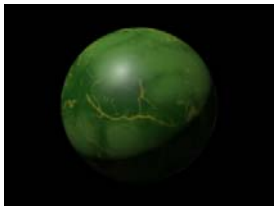
These parameters are similar to their functionality in the plastic shader.

color darkcolor = (0.01, 0.12, 0.004)
color lightcolor = (0.06, 0.18, 0.02)

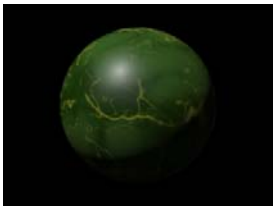
Defines two colors from which the base color of the marble substrate (not the veins) will be derived. The base color actually varies over the surface, using a low-frequency noise pattern to vary among colors between `darkcolor` and `lightcolor`.



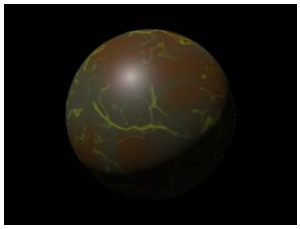
color darkcolor = (0.01, 0.12, 0.004), color lightcolor = (0.06, 0.18, 0.02) (default)



color darkcolor = (0.01, 0.12, 0.004), color lightcolor = (0.12, 0.36, 0.04)



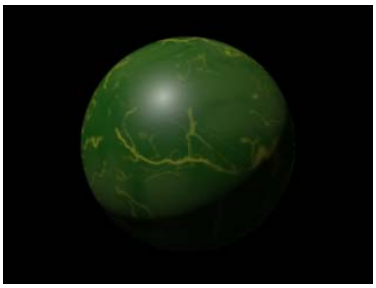
color darkcolor = (0, 0.06, 0), color lightcolor = (0.06, 0.18, 0.02)



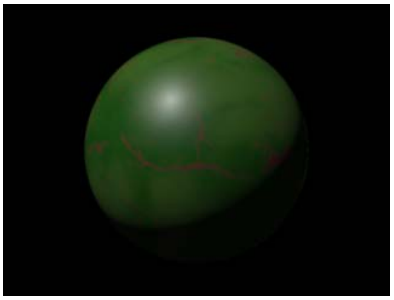
color darkcolor = (0.16, 0.1, 0.05), color lightcolor = (0.16, 0.06, 0.01)

color veincolor = color(0.47, 0.57, 0.03)

The color of the veins.



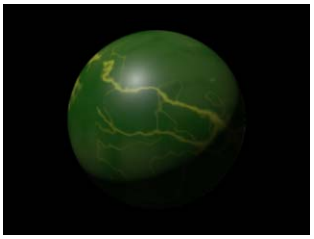
veincolor = (0.47, 0.57, 0.03) (default)



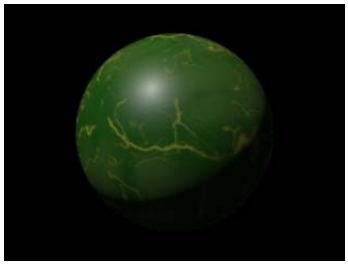
veincolor = (.25, .02, .05)

float veinfreq = 1

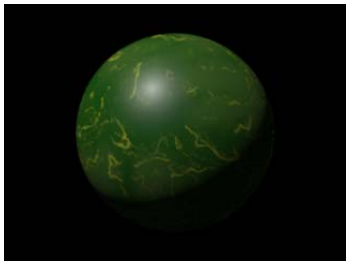
Alters the basic frequency of the veining. Higher values make smaller, closer veins; lower values make broader, more spaced-out veins.



veinfreq = 0.5



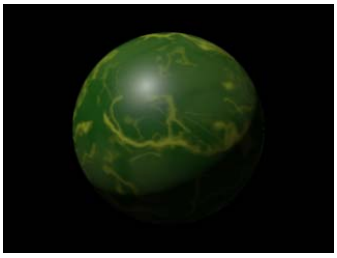
veinfreq = 1 (default)



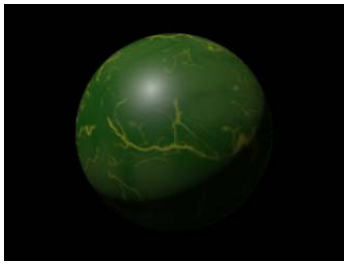
veinfreq = 2

float sharpness = 25

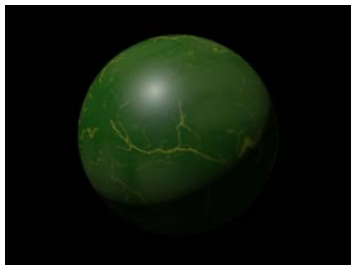
Controls the sharpness of the boundary between the veins and the background substrate. Low values make a very soft transition, high values make a sharp transition between vein and background.



sharpness = 5



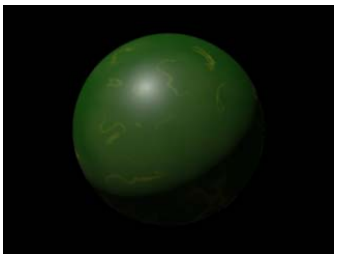
sharpness = 25 (default)

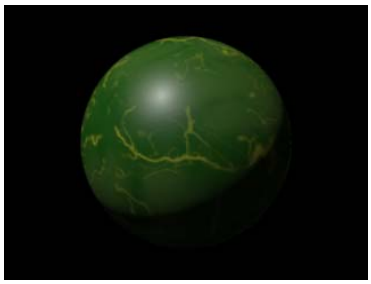


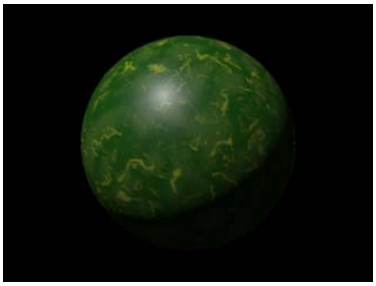
sharpness = 75

float shadingfreq = 1

Scales the overall texture (not just the vein spacing, as `veinfreq`).


shadingfreq = 0.5


shadingfreq = 1 (default)


shadingfreq = 3

point Pref = P
string shadingspace = "shader"

The pattern may be attached to a "reference mesh" given by the primitive variable `"Pref"`. If no `Pref` is attached to the geometry, the actual geometric position `P` will be the basis for texturing. In either case, the point will be transformed into the coordinate system named by `shadingspace`, which can be rigidly attached to the object. The space named by `shadingspace` may be any named coordinate system (e.g., in Mango, specified by a locator node that causes the transformation to be saved), and the coordinate system may be oriented separately from the object or shader, thus moving the solid texture pattern relative to the object.

Attributes and Globals

"color opacity", (1,1,1)

`opacity` provides the opacity of the material.

oak.gsl

Description

The surface shader `oak` makes a procedural solid texture material that looks like oak wood.

Parameters

float Ka = 1
float Kd = 1
float Ks = 0.25
float roughness = 0.25

These parameters are similar to their functionality in the plastic shader.

float divotdepth = 0.5

Scales the overall amplitude of the various pits and grooves appear to make the surface appear bumpy.


divotdepth = 0.0


divotdepth = 0.02


divotdepth = 0.05

color Clightwood = color (0.5, 0.2, .067)
color Cdarkwood = color (0.15, 0.077, 0.028)

Controls the light and dark wood colors. Changing these can an have important on the type of wood it looks like and the contrast of the grain and rings.

Clightwood = (.5, .2, .067)
Cdarkwood = (0.15, 0.077, 0.028)

Clightwood = (.6, .28, .0938)
Cdarkwood = (0.375, 0.1925, 0.07)

Clightwood = (.1, .04, .0134)
Cdarkwood = (0.0188, 0.00962, 0.007)

float ringy = 1

Controls the visibility of the tree rings.

ringy = 0.5

ringy = 1 (default)

ringy = 1.5

float ringfreq = 8

Controls the frequency of the tree rings. Lower values will make thicker, widely-spaced rings. Higher values will make thinner, closer rings.

ringfreq = 4

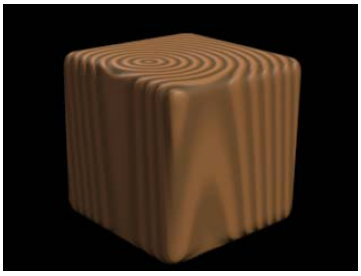
ringfreq = 8 (default)

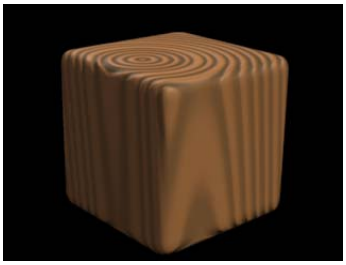
ringfreq = 12

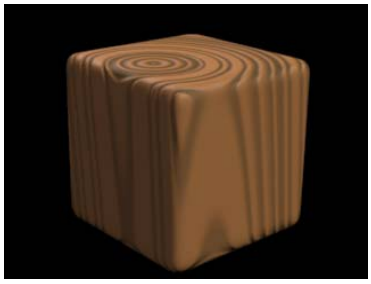
float ringunevenness = 0.3

Controls the unevenness of the tree rings. Lower values will make rings that all have the same thickness. Higher values will make some rings thick and others thin.

For clarity, the examples below are rendered with ringfreq = 4, and with ringnoise, trunkwobble, angularwobble, and grainy all set to 0.


ringunevenness = 0

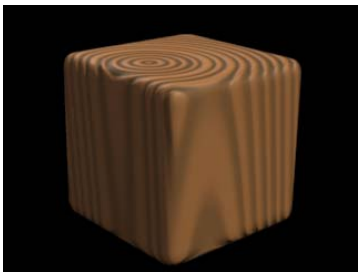

ringunevenness = 0.3 (default)

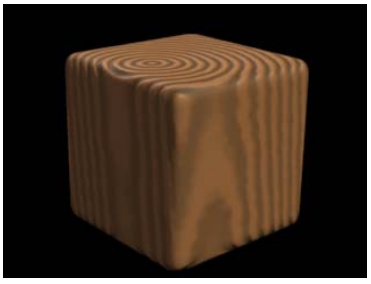

ringunevenness = 0.75

float ringnoise = 0.02

Controls the amplitude of noise of the tree ring.

For clarity, the examples below are rendered with ringfreq = 4, and with ringunevenness, trunkwobble, angularwobble, and grainy all set to 0.


ringnoise = 0

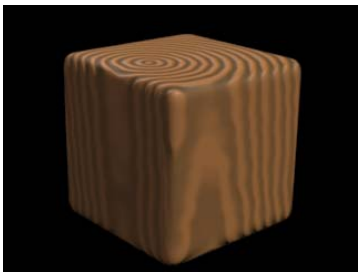

ringnoise = 0.05


ringnoise = 0.1

float ringnoisefreq = 1

Controls the frequency of the noise of the tree rings.

For clarity, the examples below are rendered with ringfreq = 4, ringnoise = 0.05, and with ringunevenness, trunkwobble, angularwobble, and grainy all set to 0.


ringnoisefreq = 0.5

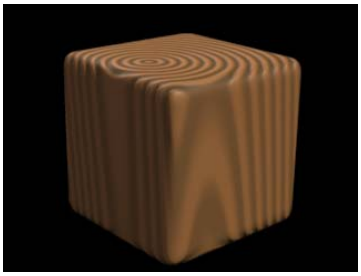

ringnoisefreq = 1

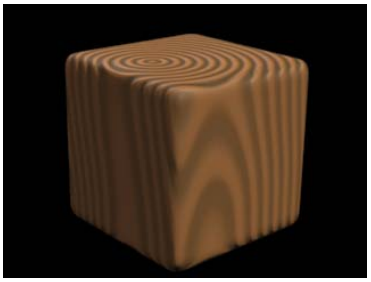

ringnoisefreq = 2

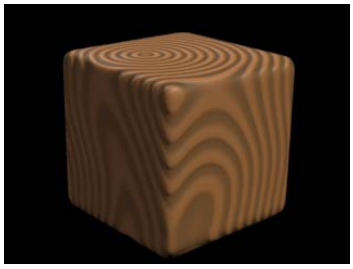
float trunkwobble = 0.15

Controls amount of noise along trunk (that is, in the *z* direction).

For clarity, the examples below are rendered with ringfreq = 4, trunkwobblefreq = 0.25, and with ringnoise, ringunevenness, angularwobble, and grainy all set to 0.


trunkwobble = 0

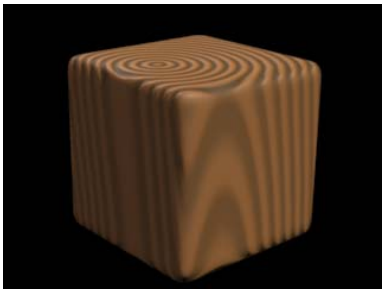

trunkwobble = 0.25 (default)

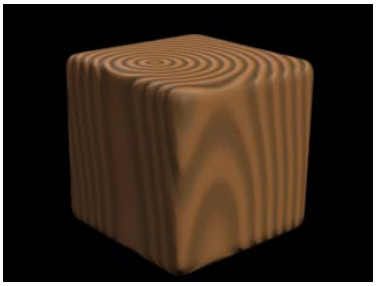

trunkwobble = 1

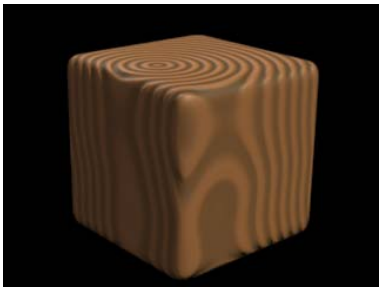
float trunkwobblefreq = 0.025

Controls amount of frequency of the trunk wobble noise.

For clarity, the examples below are rendered with ringfreq = 4, trunkwobble = 0.25, and with ringnoise, ringunevenness, angularwobblefreq, and grainy all set to 0.


trunkwobblefreq = 0

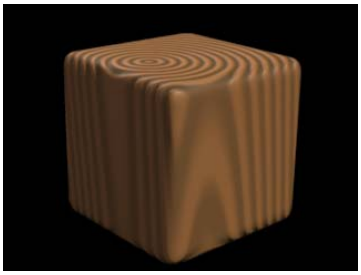

trunkwobblefreq = 0.25 (default)

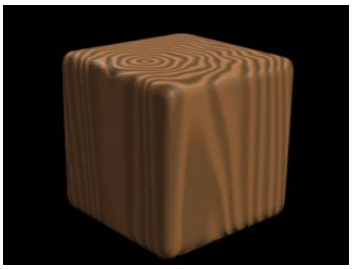

trunkwobblefreq = 1

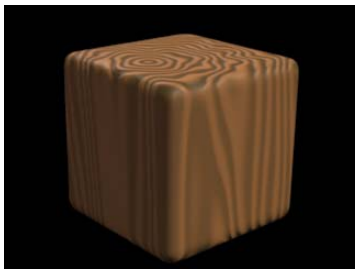
float angularwobble = 0.15

Controls amount of noise around angles of the center line.

For clarity, the examples below are rendered with ringfreq = 4, angularwobblefreq = 1.5, and with ringnoise, ringunevenness, trunkwobble, and grainy all set to 0.


angularwobble = 0

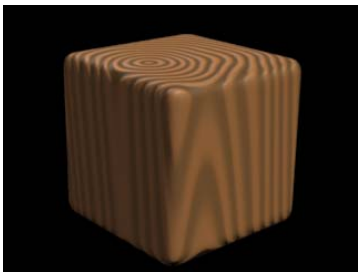

angularwobble = 1

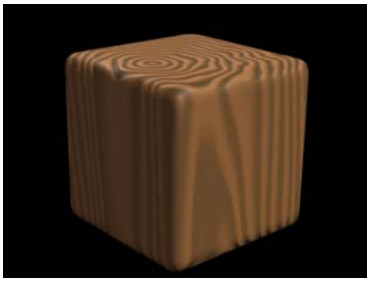

angularwobble = 1.5

float angularwobblefreq = 0.025

Controls frequency of the angular wobble noise.

For clarity, the examples below are rendered with ringfreq = 4, angularwobble = 1, and with ringnoise, ringunevenness, trunkwobble, and grainy all set to 0.

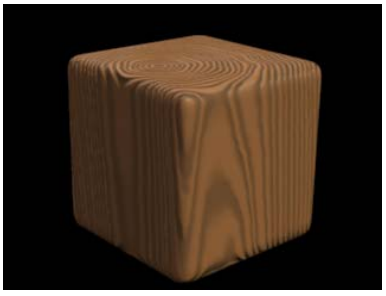

angularwobblefreq = 0.75


angularwobblefreq = 1.5


angularwobblefreq = 3

float grainy = 1

Controls the visibility of the wood grain.


grainy = 0


grainy = 0.5


grainy = 1 (default)

float grainfreq = 8

Controls the frequency of the wood grain.


grainfreq = 15


grainfreq = 25 (default)


grainfreq = 40

float shadingfreq = 1

Scales the frequency of the overall texture (rings, noise, grain, everything).


shadingfreq = 0.5


shadingfreq = 1 (default)


shadingfreq = 1.5

point Pref = P
string shadingspace = "shader"

The pattern may be attached to a "reference mesh" given by the primitive variable `"Pref"`. If no `Pref` is attached to the geometry, the actual geometric position `P` will be the basis for texturing. In either case, the point will be transformed into the coordinate system named by `shadingspace`, which can be rigidly attached to the object. The space named by `shadingspace` may be any named coordinate system (e.g., in Mango, specified by a locator node that causes the transformation to be saved), and the coordinate system may be oriented separately from the object or shader, thus moving the solid texture pattern relative to the object.

Attributes and Globals

"color C", (1,1,1)

`C` provides the base color of the marble material, if `basecolor` does not override it.

"color opacity", (1,1,1)

`opacity` provides the opacity of the material.

oakplank.gsl

Description

The surface shader `oakplank` makes a procedural texture material that looks like oak wood planks. The oak pattern within each plank is the same as the solid `oak.gsl` shader. The planks are staggered and somewhat randomized, giving an irregular look.

The plank pattern is projected onto the *x*-*y* plane of `shadingspace` (which is `"shader"` space by default). It is important to properly align the shading space to the object, or the pattern will not look like wood planks.

Parameters

float Ka = 1
float Kd = 1
float Ks = 0.75
float roughness = 0.1

These parameters are similar to their functionality in the plastic shader.

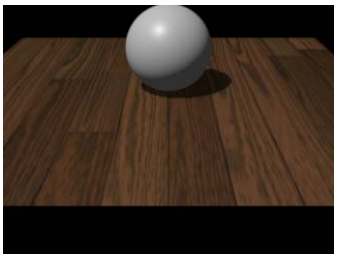
float divotdepth = 0.012
color Clightwood = (.5, .2, .067)
color Cdarkwood = (.15, .077, .028)
float ringy = 1
float ringfreq = 8
float ringunevenness = 0.5
float ringnoise = 0.02
float ringnoisefreq = 1
float trunkwobble = 0.15
float trunkwobblefreq = 0.025
float angularwobble = 1
float angularwobblefreq = 1.5
float grainy = 1
float grainfreq = 25

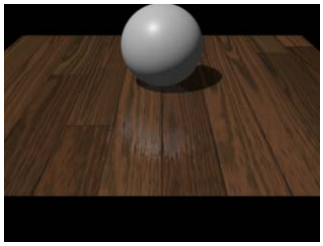
These parameters are similar to their functionality in the `oak` shader. The wood grain pattern within each plank is identical to `oak`.

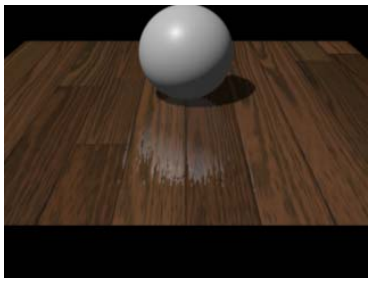
float Kr = 1
float eta = 1.5
float blur = 0
string envname = ""
string envspace = "world"
float envrad = 0
float samples = 1

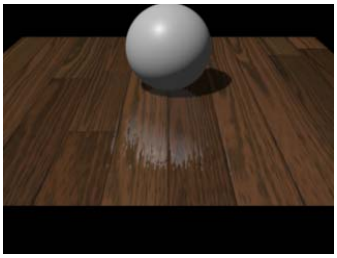
These parameters control reflection, much as they do in `metal.gsl`.

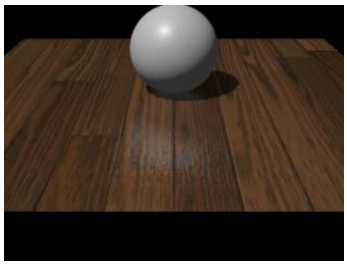
For clarity, in the following examples, `Kd` = 0.25:

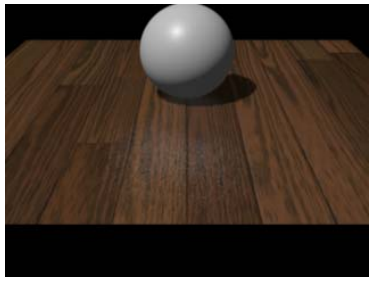

Kr = 0


Kr = 1 (default)


Kr = 2

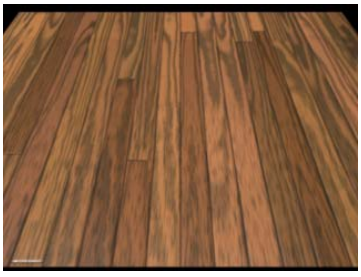

blur = 0


blur = 0.1


blur = 0.25

float plankwidth = 2

The width of each plank.


plankwidth = 1


plankwidth = 2 (default)


plankwidth = 4

float planklength = 30

The length of each plank.


planklength = 15

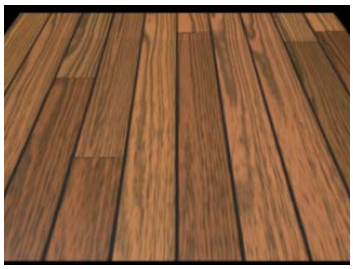

planklength = 30 (default)

float groovewidth = 0.05

The width of the grooves between planks (in the "short" direction).


groovewidth = 0


groovewidth = 0.05


groovewidth = 0.15

float grooveheight = 0.05

The width of the grooves between planks (in the "long" direction).


grooveheight = 0


grooveheight = 0.05


grooveheight = 0.25

float groovedepth = 0.03

The depth that the grooves between planks receed into the wood.

In the examples below, for clarity, `planklength` = 15.


groovedepth = 0


groovedepth = 0.05


groovedepth = 0.15

float edgewidth = 0.1

Controls how close to the groove the plank starts to round downward into the groove itself.

In the examples below, for clarity, `planklength` = 15 and `groovedepth` = 0.15.


edgewidth = 0.05

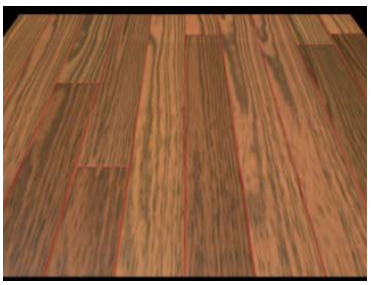

edgewidth = 0.15


edgewidth = 0.35

color Cgroove = (0.02, 0.02, 0.02)

The color of the grooves between the planks.


Cgroove = (.02, .02, .02)


Cgroove = (.5, .02, .02)

float varylum = 0.5

Controls the amount that the luminance of the wood randomly varies from plank to plank.


varylum = 0


varylum = 0.5


varylum = 1

float varysat = 0.1

Controls the amount that the saturation of the wood randomly varies from plank to plank.


varysat = 0

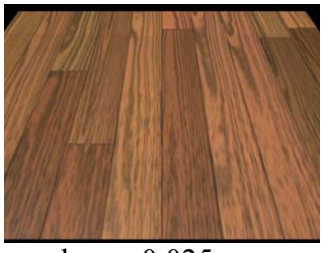

varysat = 0.2

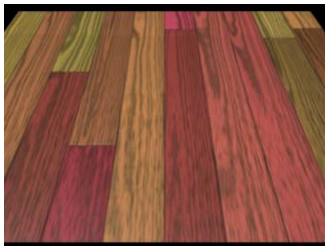

varysat = 0.5

float varyhue = 0.015

Controls the amount that the hue of the wood randomly varies from plank to plank.


varyhue = 0


varyhue = 0.025

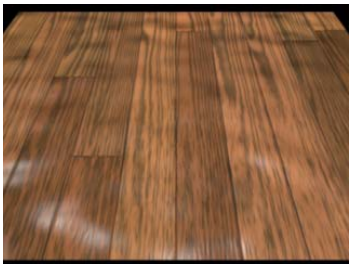

varyhue = 0.15

float varnishlump = 0.01

The amplitude of lumps in the "varnish" (addition bumps over the surface of the wood).


varnishlump = 0.01


varnishlump = 0.1


varnishlump = 0.4

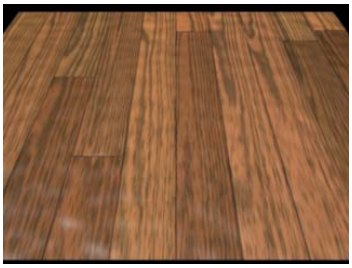
float varnishlumpfreq = 0.5

The frequency of lumps in the "varnish" (addition bumps over the surface of the wood).

For clarity, the examples below use `varnishlump` = 0.1.


varnishlumpfreq = 0.25

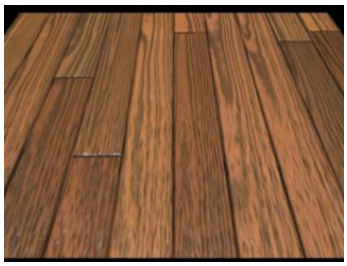

varnishlumpfreq = 0.5

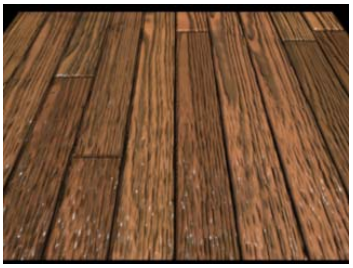

varnishlumpfreq = 1

float Km = 0.5

Scales the overall amount of bump (for everything -- grooves, grain, etc.).


Km = 1


Km = 5


Km = 15

float shadingfreq = 1

Scales the frequency of the overall texture (planks, rings, noise, grain, everything).

shadingfreq = 0.5

shadingfreq = 1 (default)

shadingfreq = 2

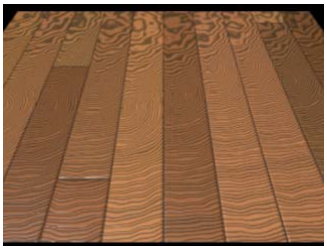
point Pref = P
string shadingspace = "shader"

The pattern may be attached to a "reference mesh" given by the primitive variable `"Pref"`. If no `Pref` is attached to the geometry, the actual geometric position `P` will be the basis for texturing. In either case, the point will be transformed into the coordinate system named by `shadingspace`, which can be rigidly attached to the object. The space named by `shadingspace` may be any named coordinate system (e.g., in Mango, specified by a locator node that causes the transformation to be saved), and the coordinate system may be oriented separately from the object or shader, thus moving the solid texture pattern relative to the object.

Note, however, that while adjusting the orientation of the shading space relative to the object can re-align the planks (such as in the middle example below), the orientation can also be *mis-aligned*, as in the example on the right below, in which the pattern no longer looks like correct wood planks because the object is not oriented properly with respect to the solid texture:

Properly aligned

Rotated 45 deg in z



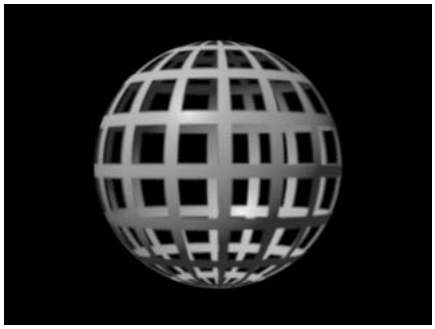
Incorrectly rotate 45 deg in x

Attributes and Globals

"color opacity", (1,1,1)

`opacity` provides the opacity of the material.

screen.gsl



Description

The surface shader `screen` makes a plastic or metallic material in rectangular strips, with holes between the strips.

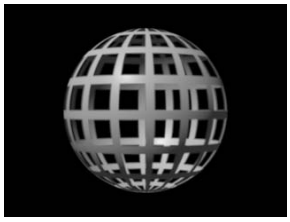
Parameters

float Ka = 1
float Kd = 0.75
float Ks = 0.4
float roughness = 0.1
color specularcolor = color (1,1,1)

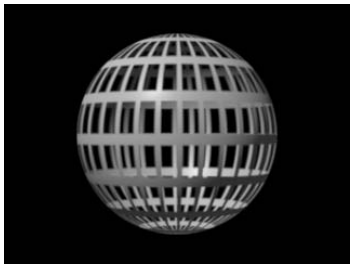
These parameters are identical to their functionality in the plastic shader.

float sfreq = 10
float tfreq = 10

Control the frequency of the screen strips in s and t. Higher frequencies lead to smaller, closer strips; lower frequencies lead to wider strips spaced farther apart.


sfreq = 10, tfreq = 10 (default)

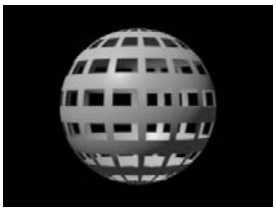

sfreq = 5, tfreq = 10


sfreq = 10, tfreq = 20

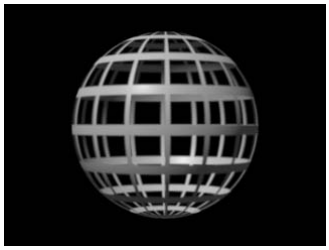
float sdensity = 0.25
float tdensity = 0.25

Control the portion of each cycle that is solid material versus hole, for the s and t directions, respectively.

sdensity = 0.25, tdensity = 0.25 (default)



sdensity = 0.5, tdensity = 0.25



sdensity = 0.25, tdensity = 0.125

float s = u
float t = v

The s and t parameters give the 2D coordinates for the pattern. By default, they are equal the the u and v parameters of the surface, respectively. But of course they may be re-mapped as primitive variables (with new "s" and "t" values per vertex of the geometry), or may be computed by another shader layer and joined to this shader using `ConnectShaders`.

Attributes and Globals

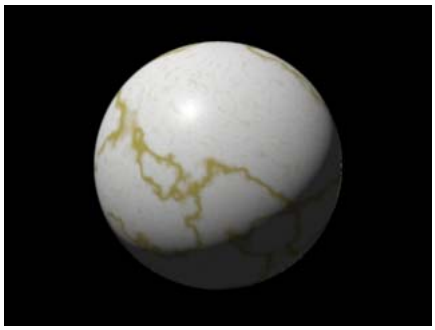
"color C", (1,1,1)

`C` provides the base color of the screen material.

"color opacity", (1,1,1)

`opacity` provides the base opacity of the strip material (of course the opacity goes to zero in the "hole" parts of the surface).

veinedmarble.gsl



Description

The surface shader `veinedmarble` makes a basic veined marble material.

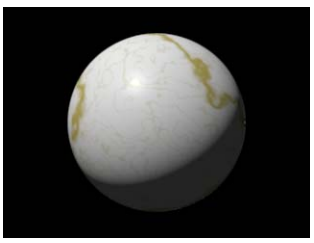
Parameters

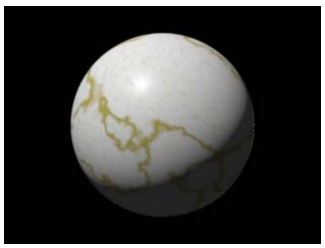
float Ka = 0.5
float Kd = 0.8
float Ks = 0.4
float roughness = 0.075
color specularcolor = color (1,1,1)

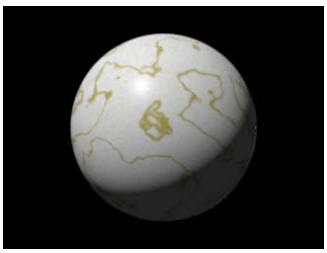
These parameters are similar to their functionality in the plastic shader.

float veinfreq = 1

Alters the basic frequency of the veining. Higher values make smaller, closer veins; lower values make broader, more spaced-out veins.

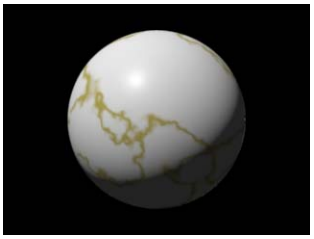

veinfreq = 0.5


veinfreq = 1 (default)

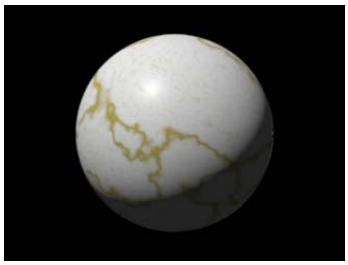

veinfreq = 2

float veinlevels = 2

The number of levels of veins. Higher vales make smaller, thinner, veins in addition to the big thick ones. Only integral values (1, 2, 3, ...) make sense.



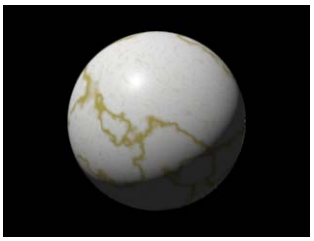
veinlevels = 1



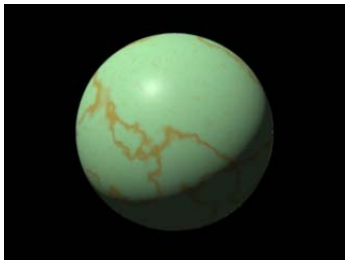
veinlevels = 2 (default)

color basecolor = C

By default, the base color (the marble substrate, not the veins) is the default surface color `C`. But it can be set to a specific color with this parameter.



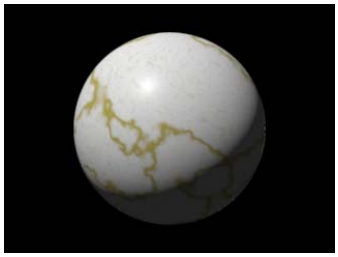
basecolor = (1,1,1)



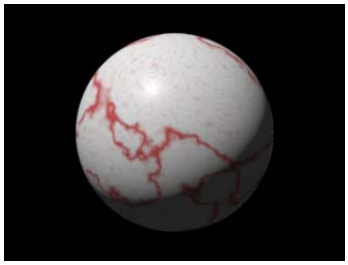
basecolor = (0.5,1,0.5)

color veincolor = color(0.6, 0.5, 0.1)

The color of the veins.



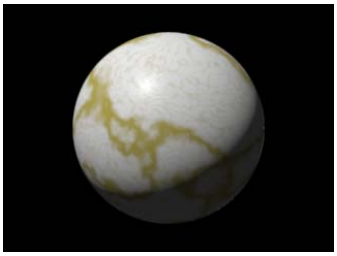
veincolor = (0.6, 0.5, 0.1) (default)

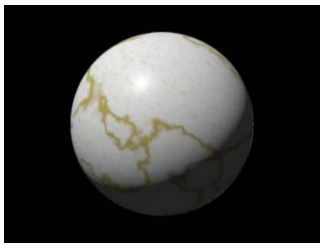


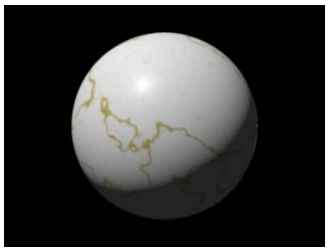
veincolor = (0.75, .1, .1)

float sharpness = 8

Controls the sharpness of the boundary between the veins and the background substrate. Low values make a very soft transition, high values make a sharp transition between vein and background.

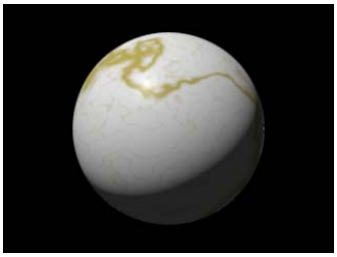

sharpness = 2

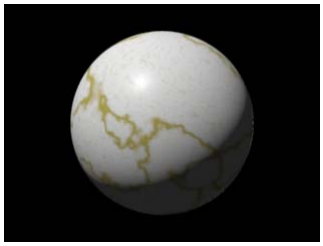

sharpness = 8 (default)

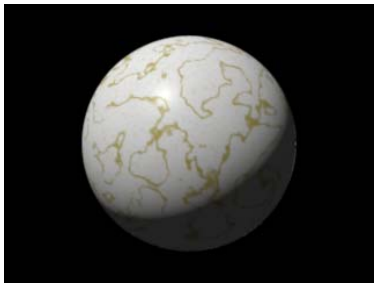

sharpness = 30

float shadingfreq = 1

Scales the overall texture (not just the vein spacing, as `veinfreq`).


shadingfreq = 0.5


shadingfreq = 1 (default)


shadingfreq = 3

point Pref = P
string shadingspace = "shader"

The pattern may be attached to a "reference mesh" given by the primitive variable `"Pref"`. If no `Pref` is attached to the geometry, the actual geometric position `P` will be the basis for texturing. In either case, the point will be transformed into the coordinate system named by `shadingspace`, which can be rigidly attached to the object. The space named by `shadingspace` may be any named coordinate system (e.g., in Mango, specified by a locator node that causes the transformation to be saved), and the coordinate system may be oriented separately from the object or shader, thus moving the solid texture pattern relative to the object.

Attributes and Globals

"color C", (1,1,1)

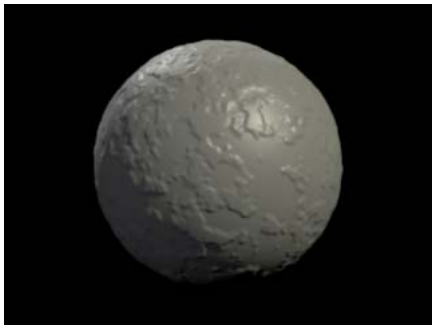
`C` provides the base color of the marble material, if `basecolor` does not override it.

"color opacity", (1,1,1)

`opacity` provides the opacity of the material.

Displacement Shaders

castucco.gsl



Description

The displacement shader makes a stucco-like material.

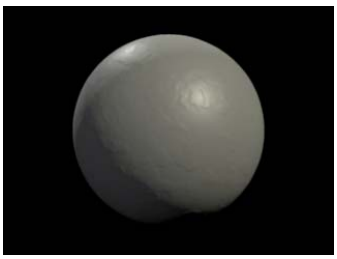
Basically, the displacement is a noise pattern, sharpened, and thresholded at both the top and bottom (making "floors" and "mesas" to the bumps). It's called "castucco" because it's the texture used on practically every wall in the state of California ("CA").

Parameters

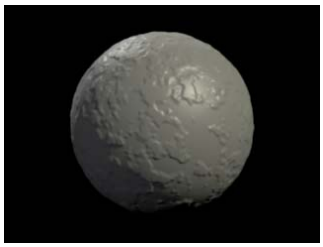
All of the example images below are rendered with `plastic.gsl` as the surface shader.

float Km = 0.05

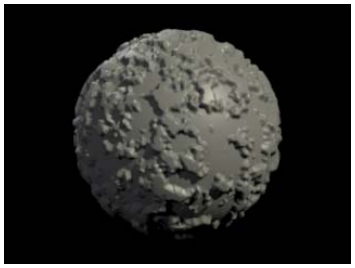
The overall height of the bumps, measured in the coordinate system specified by the `shadingspace` parameter (usually "shader" space).



Km = 0.01



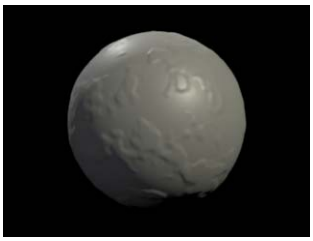
Km = 0.05 (default)

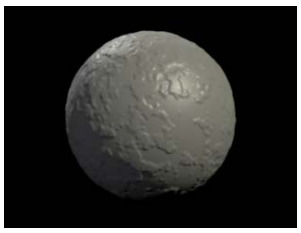


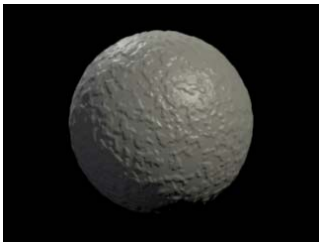
Km = 0.25

float freq = 1

The frequency of the bump pattern.

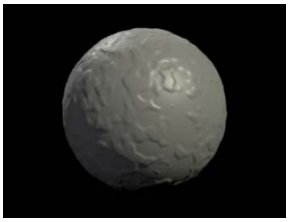

freq = 0.5

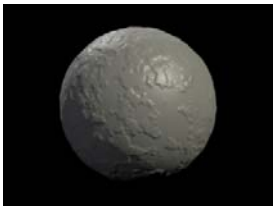

freq = 1 (default)

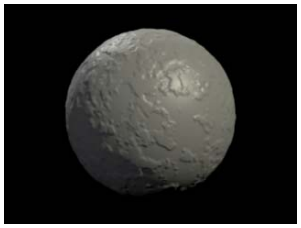

freq = 2

float octaves = 3

The number of octaves used in the noise pattern. More octaves make for more wiggly borders of the mesas. Fewer octaves make smoother borders of the mesas.

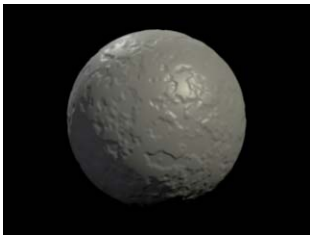

octaves = 2

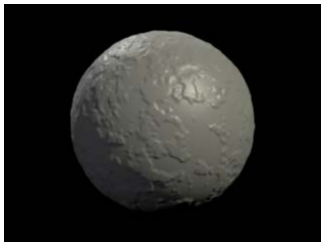

octaves = 3 (default)

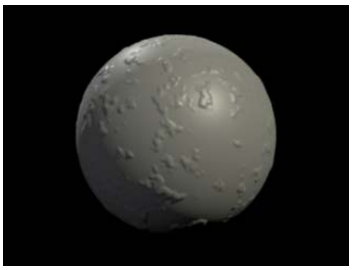

octaves = 4

float sharp = 1

The sharpness of the bumps. Low values make rolling hills; high values make mesas with sheer faces.


sharp = 0.75


sharp = 1 (default)


sharp = 1.5

float trough = 0.43
float peak = 0.67

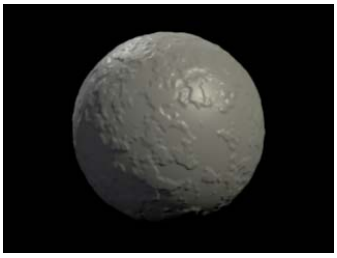
Control the low and high values at which the noise pattern is thresholded (and then rescaled to the 0-1 range).

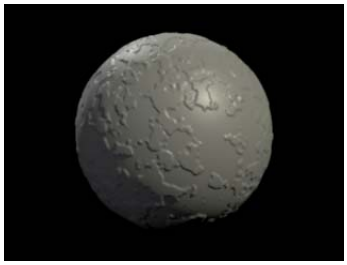
Low values for `trough` makes the valleys hilly and indistinct. Higher values for `trough` make the valleys wide and very flat. High values for `peak` makes the tops

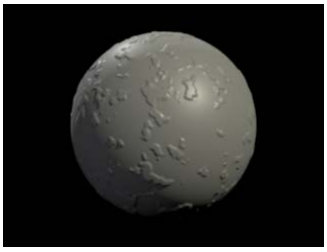
of the mesas hilly and indistinct. Lower values for `peak` make the tops of the mesas wide and very flat.

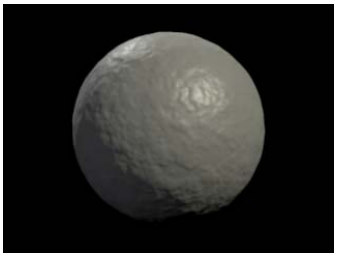
When `trough` and `peak` are far apart, the overall pattern is more like hills. When `trough` and `peak` are close together in value, you will have wide flat valleys and wide flat mesas with very sharp walls separating them.

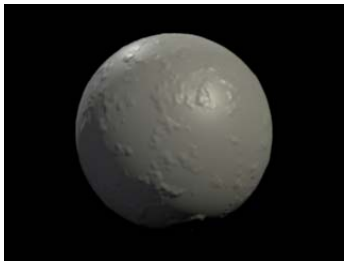
When the "midpoint" of `trough` and `peak` is close to 0.5, the pattern will be evenly mixed between valleys and mesas. As the midpoint deviates from 0.5, more of the pattern overall will be "valley floor" versus "mesa peak," depending on the direction of the shift.


trough = 0.43, peak = 0.67 (default)


trough = 0.5, peak = 0.6

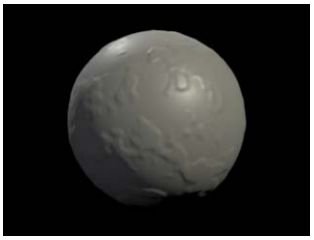

trough = 0.6, peak = 0.7

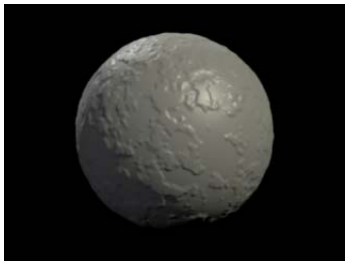

trough = 0.2, peak = 0.8

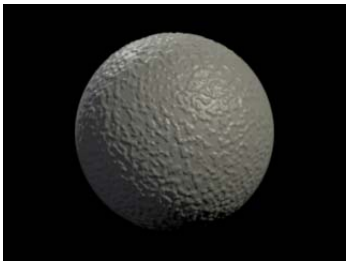

trough = 0.5, peak = 0.9

float shadingfreq = 1

Scales the overall texture.


shadingfreq = 0.5


shadingfreq = 1 (default)


shadingfreq = 4

point Pref = P
string shadingspace = "shader"

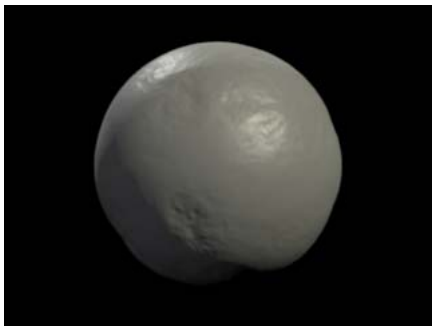
The pattern may be attached to a "reference mesh" given by the primitive variable `"Pref"`. If no `Pref` is attached to the geometry, the actual geometric position `P` will be the basis for texturing. In either case, the point will be transformed into the coordinate system named by `shadingspace`, which can be rigidly attached to the object. The space named by `shadingspace` may be any named coordinate system (e.g., in Mango, specified by a locator node that causes the transformation to be saved), and the coordinate system may be oriented separately from the object or shader, thus moving the solid texture pattern relative to the object.

Attributes and Globals

Attribute ("float displace:maxradius", 0)

Don't forget that this is a true displacement, and in order to avoid artifacts you will need to set the `displace:maxradius` appropriately to the largest amount that the surface may displace (which is typically `Km` units, in `shadingspace`).

dented.gsl



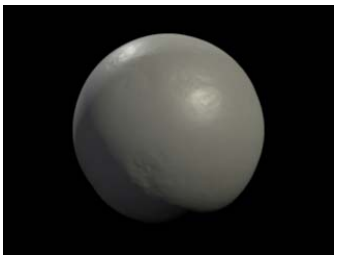
Description

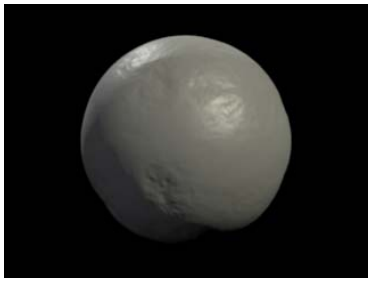
The displacement shader makes a dented material, like something that has been beaten with a blunt instrument.

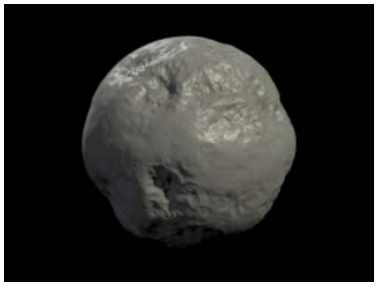
Parameters

All of the example images below are rendered with `plastic.gsl` as the surface shader.

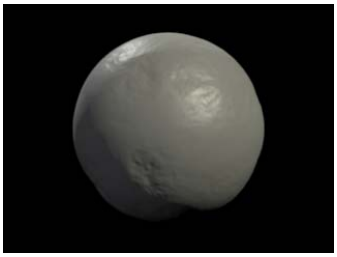
float Km = 1
The overall depth of the bumps, measured in the coordinate system specified by the `shadingspace` parameter (usually "shader" space).

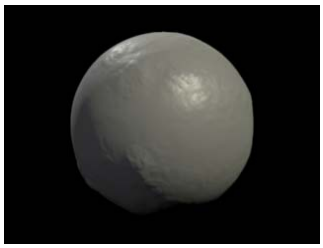

Km = 0.025


Km = 0.05


Km = 0.25

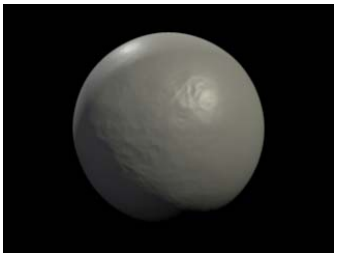
The dents go opposite the direction of the surface normal `N` (which generally means *inward* toward the center of the object). Choosing a negative value for Km will make the dents go in the opposite direction. This can be used simply to correct for normals facing the wrong direction, or simply to make the dents bump out rather than in:

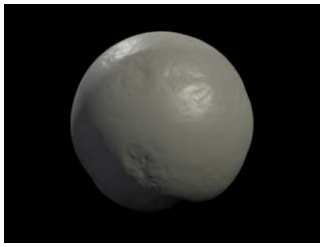

Km = 0.05

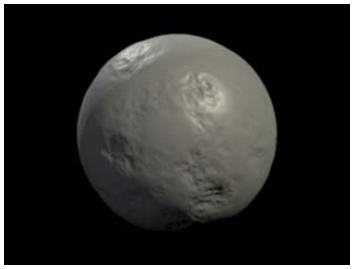

Km = -0.05

float freq = 1

The frequency of the bump pattern.

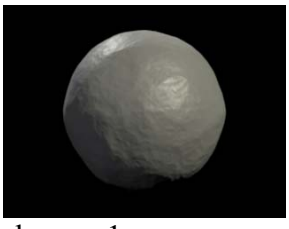

freq = 0.5

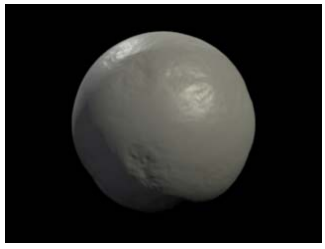

freq = 1 (default)

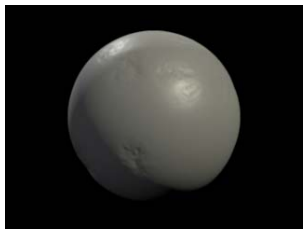

freq = 2

float sharp = 3

The sharpness of the bumps. Low values are more like uniformly worn, crumpled paper; high values are more like a mostly-smooth surface that has occasionally been hit several times with a hammer.


sharp = 1

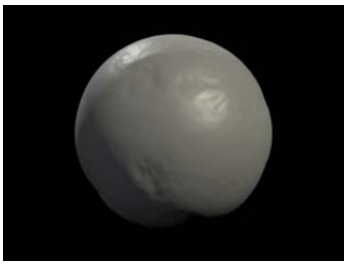

sharp = 3 (default)

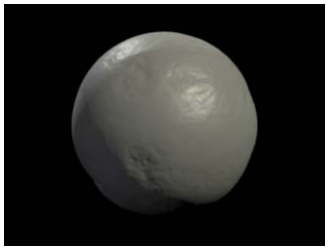

sharp = 5

float octaves = 6

The number of octaves in the bumps. More octaves make for "rougher" bumps, whereas fewer octaves make for smooth bumps.


octaves = 2


octaves = 4


octaves = 8

float shadingfreq = 1

Scales the overall texture, both frequency and amplitude.

point Pref = P
string shadingspace = "shader"

The pattern may be attached to a "reference mesh" given by the primitive variable `"Pref"`. If no `Pref` is attached to the geometry, the actual geometric position `P` will be the basis for texturing. In either case, the point will be transformed into the coordinate system named by `shadingspace`, which can be rigidly attached to the object. The space named by `shadingspace` may be any named coordinate system (e.g., in Mango, specified by a locator node that causes the transformation to be saved), and the coordinate system may be oriented separately from the object or shader, thus moving the solid texture pattern relative to the object.

Attributes and Globals

Attribute ("float displace:maxradius", 0)

Don't forget that this is a true displacement, and in order to avoid artifacts you will need to set the `displace:maxradius` appropriately to the largest amount that the surface may displace (which is typically `Km` units, in `shadingspace`).

dispmap.gsl

Description

The displacement shader displaces based on either an input connected from an earlier layer, or a simple s-t texture lookup, or both.

There are two main ways to use this shader. First, it may be used "stand-alone" by performing a simple s-t texture lookup and using it for displacement. For example (Pyg):

```
Shader ("displacement", "dispmap", "string texturename",
  "nvlogo.tx", "string wrap", "periodic", "float blur", 0)
```

Alternately, this shader can be the last layer in a shader group, where earlier layers compute the amount of displacement per point, which is then connected to this shader's `disp` parameter. For example, the following Pyg code creates a shader group that uses the texmap.gsl shader to read texture and then connect the results to the `dispmap` shader:

```
ShaderGroupBegin()
Parameter ("string texturename", "nvlogo.tx")
Parameter ("string wrap", "periodic")
Shader ("displacement", "texmap", "texlayer")
Parameter ("float Km", 0.5)
Parameter ("float offset", 0.5)
Shader ("displacement", "dispmap", "displayer")
ConnectShaders ("texlayer", "fout", "displayer", "disp")
ShaderGroupEnd()
```

Parameters

All of the example images below are rendered with `plastic.gsl` as the surface shader. For the texture being used, the "background" is white (1) and the "logo" part is black (0).

float disp = 0

Input displacement amount, presumably connected from an earlier shader layer via `ConnectShaders`.

string texturename = ""
float blur = 0
string wrap = "default"
float s = u
float t = v

If a texture name is supplied, a simple s-t texture lookup will be performed and added to `disp`.

The s and t parameters give the 2D texture lookup coordinates, if texture mapping is used in this shader. By default, they are equal the the u and v parameters of the surface, respectively. But of course they may be re-mapped as primitive variables (with new "s" and "t" values per vertex of the geometry), or may be computed by another shader layer and joined to this shader using `ConnectShaders`.

float Km = 1

The overall amplitude scaling of the bumps, measured in the coordinate system specified by the `shadingspace` parameter (usually "shader" space).


Km = 0.125


Km = 0.5


Km = 1

Choosing a negative value for Km will make the dents go in the opposite direction as the surface normal. This can be used simply to correct for normals facing the wrong direction, or simply to swap the valleys and mountains:


Km = 0.5


Km = -0.5

string shadingspace = "shader"

Controls the coordinate system in which the displacement amounts are measured.

float offset = 0

This parameter establishes the convention for which displacement value (`disp` + the texture lookup) values move the surface up or down. Displacement amounts greater than `offset` will displace in the direction of the surface normal (usually outwards -- mountains), while less than `offset` will displace opposite the surface normal (usually inwards -- dents). Displacement values that are equal to `offset` will not move the surface at all.

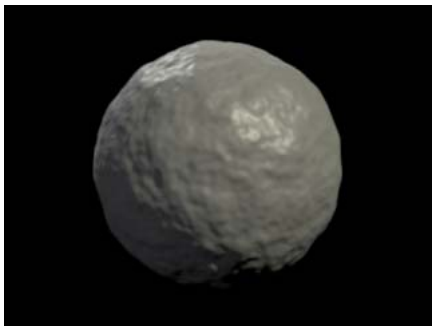
As an example, suppose that you were performing a texture lookup on a texture whose values range from 0 to 1. If **`offset = 0`**, black texels will not displace, and white texels displace fully outward (toward the surface normal). If **`offset = 1`**, black texels will displace fully inwards (away from the surface normal), and white texels will not displace at all. If **`offset = 0.5`**, texels with values < 0.5 will bump inwards while texels wth values > 0.5 will bump outwards, thus allowing you to paint a texture that controls both dents and bumps.

Attributes and Globals

Attribute ("float displace:maxradius", 0)

Don't forget that this is a true displacement, and in order to avoid artifacts you will need to set the `displace:maxradius` appropriately to the largest amount that the surface may displace (which is typically `Km` units, in `shadingspace`).

lumpy.gsl



Description

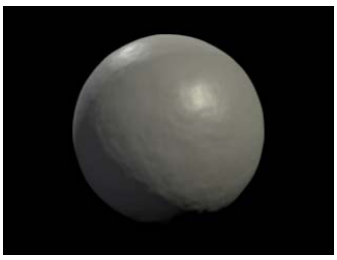
The displacement shader makes a lumpy material, like something that has been beaten with a blunt instrument.

Parameters

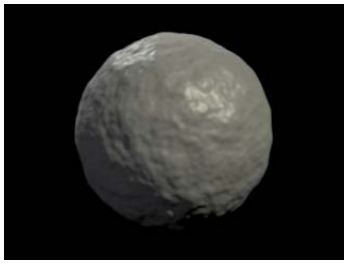
All of the example images below are rendered with `plastic.gsl` as the surface shader.

float Km = 1

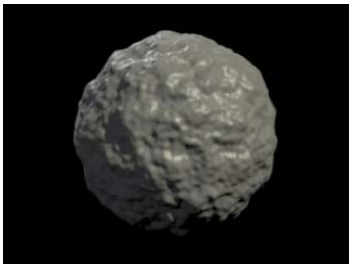
The overall depth of the bumps, measured in the coordinate system specified by the `shadingspace` parameter (usually "shader" space).



Km = 0.025



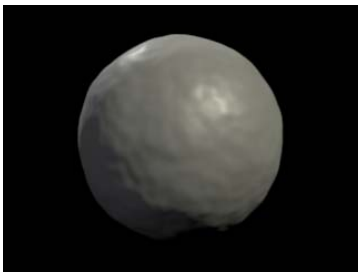
Km = 0.1

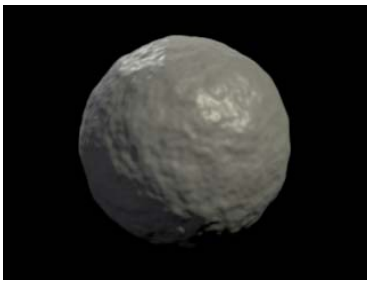


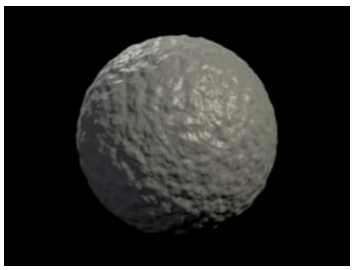
Km = 0.25

float freq = 1

The frequency of the bump pattern.

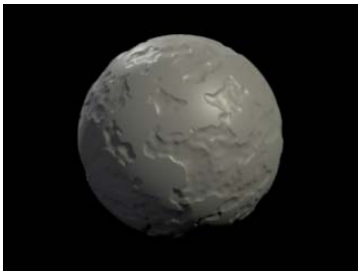

freq = 0.5

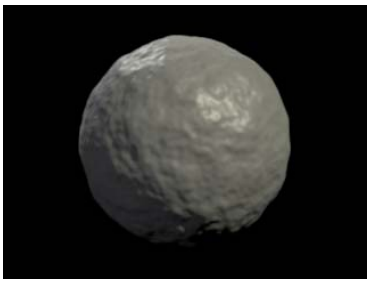

freq = 1 (default)

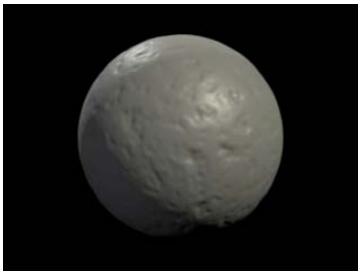

freq = 2

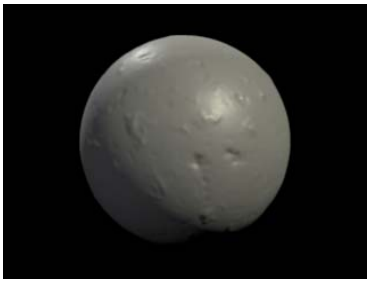
float sharp = 3

The sharpness of the bumps. Low values are more like uniformly worn, crumpled paper; high values are more like a mostly-smooth surface that has occasionally been hit several times with a hammer.


sharp = 0.5

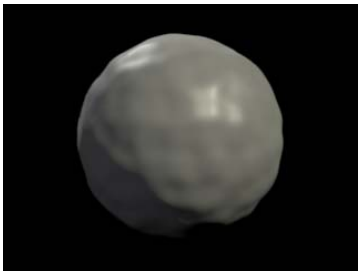

sharp = 1 (default)

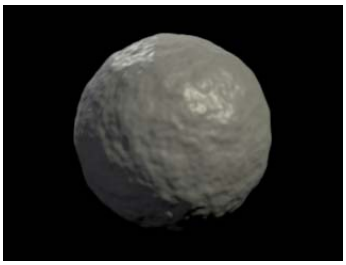

sharp = 2

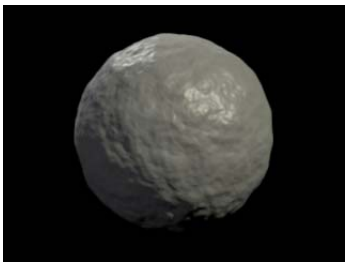

sharp = 3

float octaves = 3

The number of octaves in the bumps. More octaves make for "rougher" bumps, whereas fewer octaves make for smooth bumps.


octaves = 2


octaves = 3 (default)


octaves = 5

float shadingfreq = 1

Scales the overall texture, both frequency and amplitude.

point Pref = P
string shadingspace = "shader"

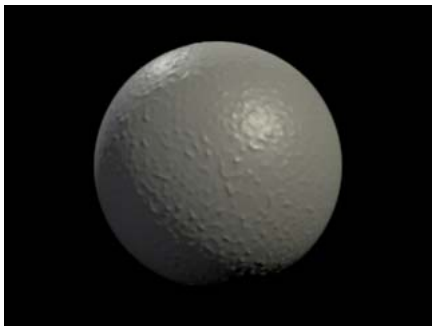
The pattern may be attached to a "reference mesh" given by the primitive variable `"Pref"`. If no `Pref` is attached to the geometry, the actual geometric position `P` will be the basis for texturing. In either case, the point will be transformed into the coordinate system named by `shadingspace`, which can be rigidly attached to the object. The space named by `shadingspace` may be any named coordinate system (e.g., in Mango, specified by a locator node that causes the transformation to be saved), and the coordinate system may be oriented separately from the object or shader, thus moving the solid texture pattern relative to the object.

Attributes and Globals

Attribute ("float displace:maxradius", 0)

Don't forget that this is a true displacement, and in order to avoid artifacts you will need to set the `displace:maxradius` appropriately to the largest amount that the surface may displace (which is typically `Km` units, in `shadingspace`).

stucco.gsl



Description

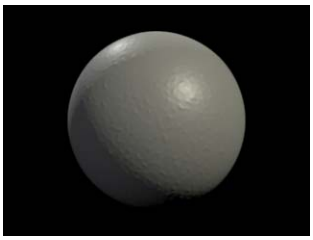
The displacement shader makes a stucco-like material with little nibs.

Parameters

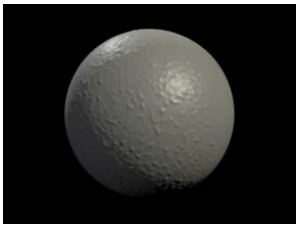
All of the example images below are rendered with `plastic.gsl` as the surface shader.

float Km = 0.05

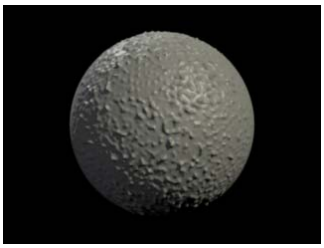
The overall height of the bumps, measured in the coordinate system specified by the `shadingspace` parameter (usually "shader" space).



Km = 0.02



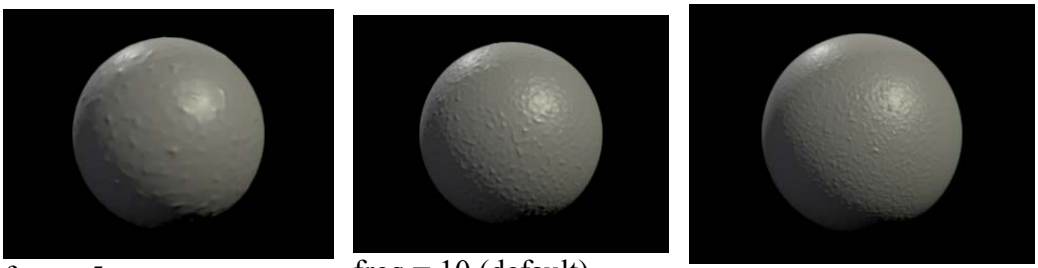
Km = 0.05 (default)



Km = 0.2

float freq = 10

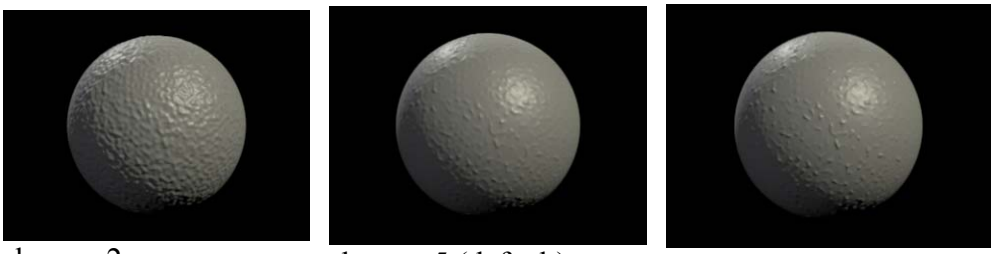
The frequency of the bump pattern.



freq = 5 | freq = 10 (default) | freq = 15

float sharp = 5

The sharpness of the bumps. Low values make rolling hills; high values make sharp little spikes.



sharp = 2 | sharp = 5 (default) | sharp = 7

float shadingfreq = 1

Scales the overall texture, both frequency and amplitude.

point Pref = P
string shadingspace = "shader"

The pattern may be attached to a "reference mesh" given by the primitive variable `"Pref"`. If no `Pref` is attached to the geometry, the actual geometric position `P` will be the basis for texturing. In either case, the point will be transformed into the coordinate system named by `shadingspace`, which can be rigidly attached to the object. The space named by `shadingspace` may be any named coordinate system (e.g., in Mango, specified by a locator node that causes the transformation to be saved), and the coordinate system may be oriented separately from the object or shader, thus moving the solid texture pattern relative to the object.

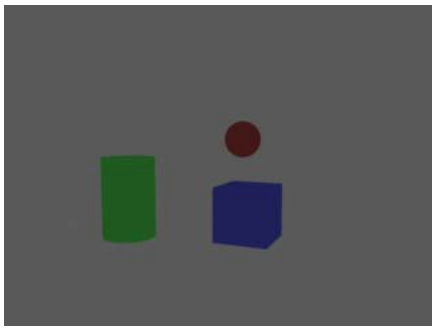
Attributes and Globals

Attribute ("float displace:maxradius", 0)

Don't forget that this is a true displacement, and in order to avoid artifacts you will need to set the `displace:maxradius` appropriately to the largest amount that the surface may displace (which is typically `Km` units, in `shadingspace`).

Light Sources

ambientlight.gsl



Description

A simple ambient light that flatly adds light to all surfaces in the scene (generally scaled by the surface's Ka parameter and base surface color).

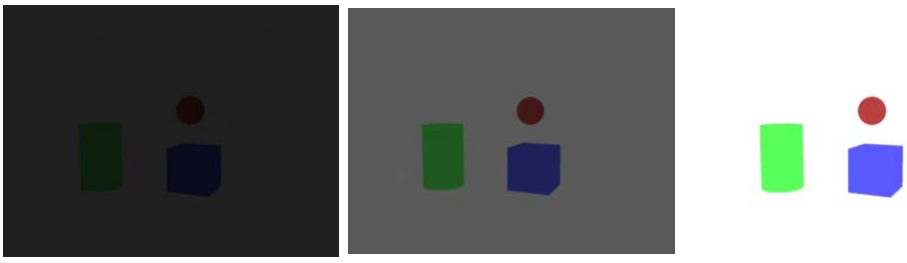
The purpose of an ambient light is to add just a bit of light to even the darkest corners, to ensure that large parts of the image do not go black.

The danger is that too much ambient light can make large parts of the image be bright and flat, which may be worse than dark and flat. Use ambient light sparingly.

Parameters

float intensity = 1

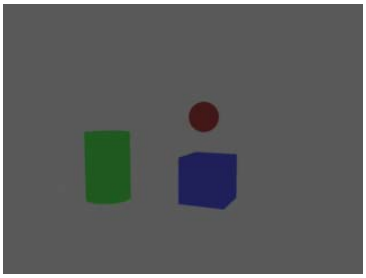
Controls the intensity (brightness) of the ambient light.



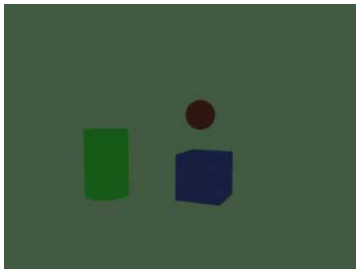
intensity = 0.01 intensity = 0.1 intensity = 1

color lightcolor = (1,1,1)

Controls the color of the ambient light.

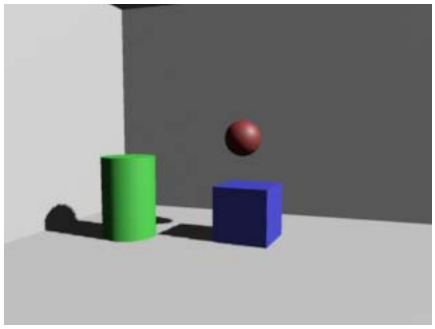


lightcolor = (1,1,1)
intensity = 0.1



lightcolor = (0.5,1,0.5)
intensity = 0.1

distantlight.gsl

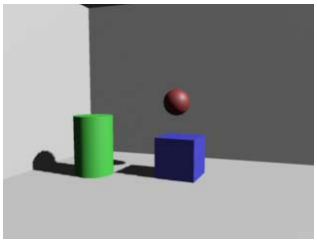
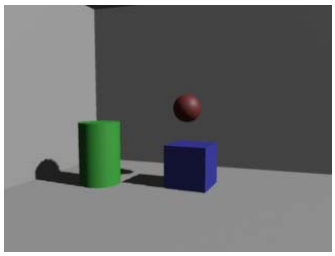
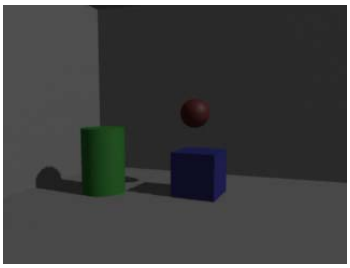


Description

A simple distant light with parallel rays, coming from infinitely far away. Shadows (both depth maps and ray tracing) are supported.

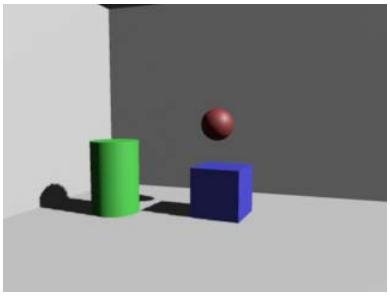
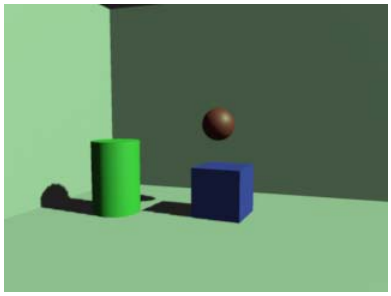
Parameters

float intensity = 1
Controls the intensity (brightness) of the light.

intensity = 1 (default) intensity = 0.5 intensity = 0.1

color lightcolor = (1,1,1)
Controls the color of the light.

lightcolor = (1,1,1) lightcolor = (0.5,1,0.5)

point from = point ("shader", 0, 0, 0)
point to = point ("shader", 0, 0, 1)

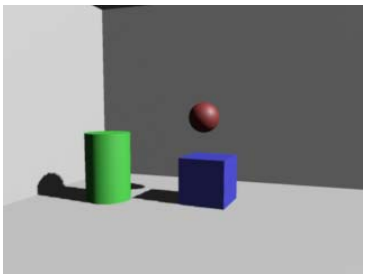
The direction that the light shines (and the shadows will cast) is in the direction implied by the vector joining the `from` and `to` positions. The default values of these parameters imply that the light shines is in the direction of the *z* axis of the light's local coordinate system. It is occasionally helpful to have a different behavior, and overriding the `from` and `to` parameters will allow you to make the light point in arbitrary directions relative to its local coordinate frame.

These parameters are somewhat archaic; their existance is largely for back-compatibility with older renderers and older shaders. We recommend leaving these parameters alone and orienting the light by merely rotating the light's local coordinate frame (this is how an interactive modeling system will usually do it).

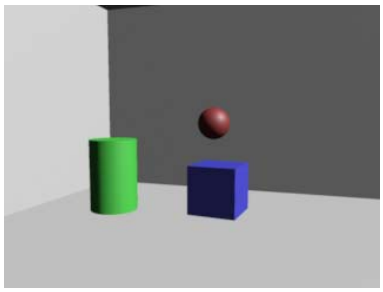
string shadowname = ""

The `shadowname` parameter specifies how the light should cast shadows. If no shadow name is supplied (the default), this light will not cast shadows. If `shadowname` is the name of a ray-traced geometry set, ray-traced shadows will be used. If `shadowname` is the name of a shadow map, the map will provide shadows (this is true for any of normal shadow maps, Woo shadows, volume shadows, or dynamic shadows).

If shadow maps are used, it is strongly recommended that the shadow map be created using an orthographic projection, since a distant light is supposed to be casting parallel light rays.



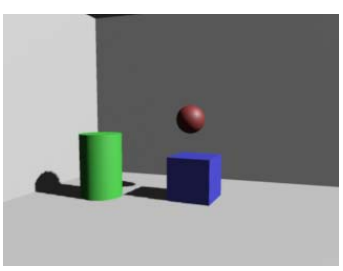
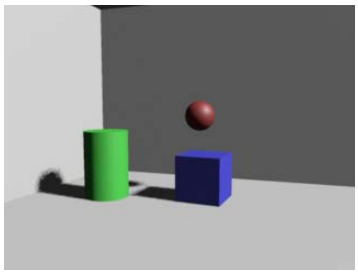
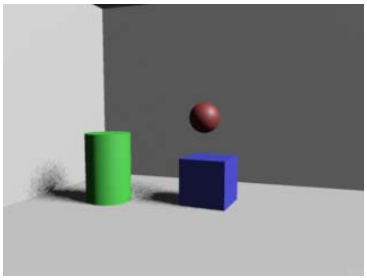
ray-traced shadows



no shadows

float shadowblur = 0

Controls how blurry the shadows are. A value of 0 will make perfectly sharp shadows. Larger values will make the shadows blurrier. This is true of both ray-traced and mapped shadows, although a particular blur value will not make an identical image with the two techniques.

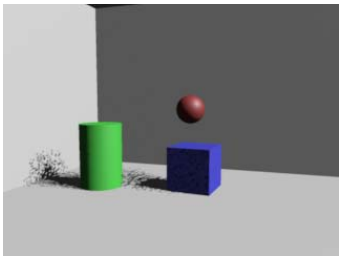
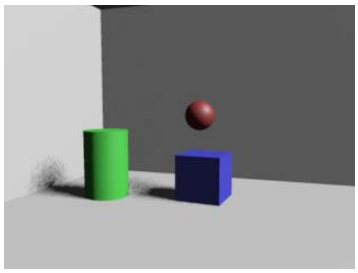
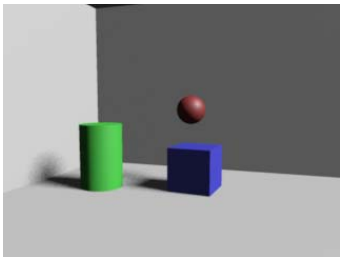




shadowblur = 0 (default) shadowblur = 0.05 shadowblur = 0.2

float shadowsamples = 1

Controls the number of samples used to compute the blurry shadows. As `shadowblur` increases, you may find that you also need to increase `shadowsamples`.

When using shadow maps, Gelato clamps `shadowsamples` at 16; that is, you will always get a minimum of 16 shadow samples for map lookups, even if you ask for less. But for ray-traced shadows, no such minimum exists (since ray tracing is so expensive).

shadowsamples = 1 shadowsamples = 4 shadowsamples = 16

float shadowbias = -1

Particularly when using shadow maps, there is a tendency for surfaces to incorrectly shadow themselves due to numerical precision issues. It is customary to add a "bias" to shadow lookups, which simply means a minimum distance that objects will appear to cast shadows. By making this bias larger than the numerical precision limits, you can usually avoid the self-shadowing artifacts.

The default of -1 indicates that the global shadow bias should be used (set by `Attribute("shadow:bias")`). Values larger than zero will set the shadow bias for this light only, independently of the global bias value.

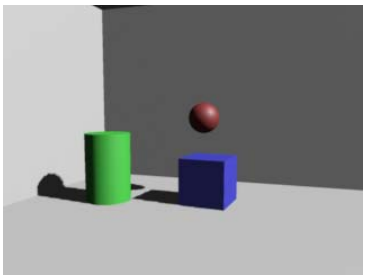
Shadow bias applies to ray tracing as well as shadow map lookups. When using ray-traced shadows, the bias is simply the minimum distance that objects will appear to cast shadows.

float __nonspecular = 0

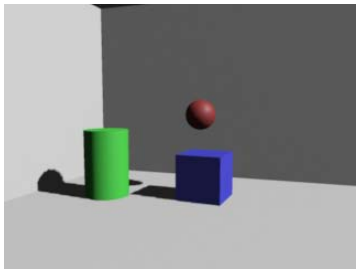
By default, the light will supply both diffuse illumination and specular highlights. By adjusting `__nonspecular` (yes, there are two underscores -- historical compatibility), you can coax the light into not contributing to specular highlights.

By default, `__nonspecular` is zero, so the light will contribute to specular highlights. If `__nonspecular` = 1, the light will not contribute to specular highlights at all. Intermediate values will give partial specularity as expected.

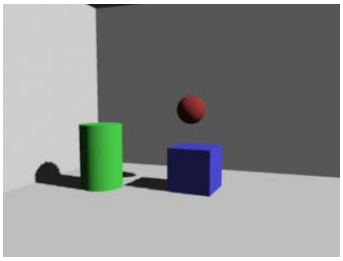
Non-specular lights are good for "fill lights" -- those lights that contribute to overall illumination but that you specifically wish to not add specular highlights to the scene.



__nonspecular = 0

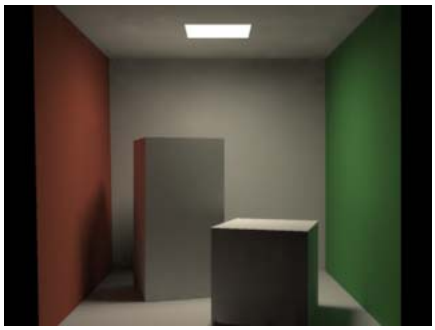


__nonspecular = 0.5



__nonspecular = 1

indirectlight.gsl



Description

This light accounts for *indirect illumination* -- the light bouncing around between objects in the scene (sometimes also known as "global illumination," though there are also many other unrelated global illumination effects). To render a scene with global illumination, all that is required is to add one light to the scene using this shader. Shine the light (via `LightSwitch`) on all objects that should receive indirect illumination. All objects that may interact (that is, have the light bounce indirectly off of, or block indirect light) need to be in a geometry set, whose name is passed to the `indirectname` parameter.

The indirect illumination is computed by tracing rays distributed around the hemisphere above each shaded point. In order to reduce the expense of ray tracing, fully-sampled hemispheres are computed *sparsely*, that is, only every once in a while, and the results are smoothly interpolated. When full hemispheres are computed, their results are stored in a spatial database (SDB), which is then used for the sparse interpolation for points in which the full sampling is not performed.

There are several parameters that control the work per hemisphere and how frequently the hemispheres are computed. Many rays per hemisphere is more expensive than few rays per hemisphere, and more frequent full sampling of hemispheres is more expensive than very sparsely computed hemispheres.

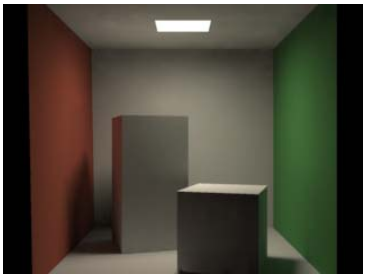
Parameters

string indirectname = "indirect"

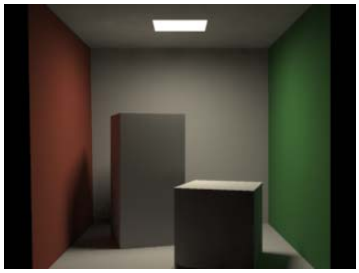
The name of the geometry set containing the objects that provide occlusion.

float intensity = 1

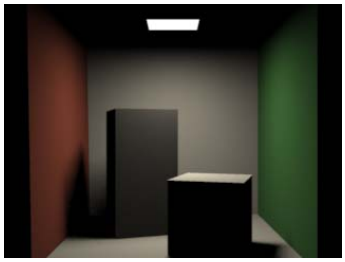
Scales the indirect contribution. The default value of 1 is a fairly physically-based simulation, but you can increase or decrease the amount of indirect illumination (in a non-physical way).



intensity = 1 (default)



intensity = 0.5



no indirectlight (equivalent to `intensity = 0`)

string sdbname = ""

The file name of the spatial database, in the case that the SDB is to be read from or written to disk.

string sdbmode = ""

The file mode for the spatial database. Valid choices are:

`""`

The SDB is kept in memory only. New samples will be computed and added to the in-memory SDB as required, but will not be saved to disk.

`"r"`

Before the first sample, the SDB will be read from disk (if a disk SDB file exists with that name). New samples will be computed and added to the in-memory SDB as required.

`"ro"`

Before the first sample, the SDB will be read from disk (if a disk SDB file exists with that name). No new samples will ever be computed -- existing samples will always be interpolated, even if it needs to ignore the `maxhitdist` value.

`"w"`

New samples will be computed and added to the in-memory SDB as required. After rendering is complete, the SDB will be written to disk.

`"rw"`

Both "r" and "w" options apply -- the SDB will be read from disk before rendering begins (if it exists), new samples will be added if necessary

during rendering, and the final SDB will be written to disk after rendering is complete.

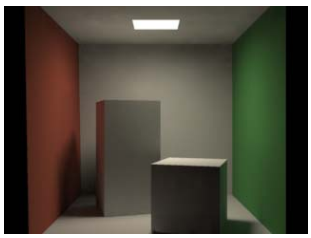
`"wo"`

Computed hemisphere samples will be saved to disk, but never interpolated (thus forcing full hemisphere computation at every shading point).

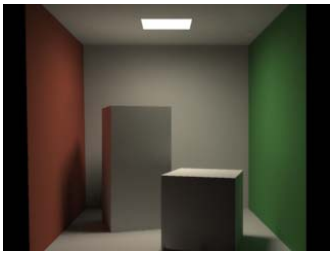
float samples = 64

The maximum number of rays used to sample the hemisphere. Total rendering time increases as the number of samples increases, but more samples make for more accurate (less noisy) estimates of the ambient occlusion. The default value of 64 is fine for rapid previews, but for full production frames a value of 256 or higher is probably necessary to reduce the sampling noise.

samples = 64
(time: 0:17)



samples = 256
(time: 0:51)



samples = 1024
(time: 3:09)

float adaptive = 0

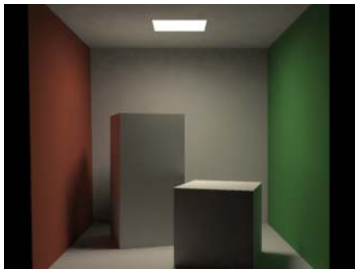
When zero (the default), the maximum number of rays will always be used when sampling a hemisphere. When nonzero, fewer than the full number of samples will be used where possible. Larger values will result in more adaptive sampling (closer to zero will result in the full sampling being used most of the time, values larger than one will be more aggressive in its attempt to use fewer samples).

float maxerror = -1

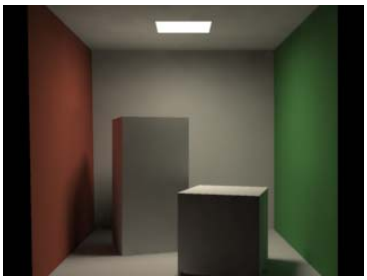
Gives the allowable error metric that in part determines when values already in the occlusion SDB can be interpolated, and when new hemispheres need to be sampled. Larger values indicate a higher allowable error -- less frequent full hemisphere recomputation. Smaller values indicate a lower tolerance for error, and therefore more frequent full recomputation. A value of 0 will force a full hemisphere computation at every shading point, never reusing previous hemispheres already in the SDB (thus giving a maximal quality image, though at maximal expense).

A value less than zero will use the value set by `Attribute("indirect:maxerror")`, which defaults to 0.25. The default parameter value is -1, which means that unless this value is changed, the attribute will be used to supply the value.

maxerror = 0.25
(time: 0:52)



maxerror = 0.1
(time: 2:33)



maxerror = 0.5
(time: 0:24)

float maxpixeldist = -1

In addition to the `maxerror` metric, full hemisphere computation will be performed at any point that is farther away from the next nearest full hemisphere by this distance, expressed roughly in units of final image pixels. Thus, a value of `maxpixeldist` = 10 will compute fully sampled hemispheres no less frequently than about every 10x10 pixel block (although they are not aligned to pixels, and it may sample more frequently, depending on the error metrics).

A value of `maxpixeldist` = 0 will force full recomputation of occlusion on every shading point (just as with `maxerror` = 0), achieving maximal quality at maximal expense.

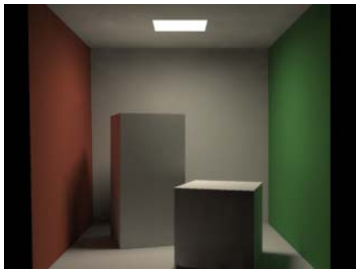
A value less than zero will use the value set by `Attribute("indirect:maxpixeldist")`, which defaults to 20. The default parameter value is -1, which means that unless this value is changed, the attribute will be used to supply the value.

float bias = 0.1

As with shadows and ray tracing, objects closer than the bias distance are ignored when computing indirect illumination. This helps to eliminate self-intersection artifacts. But be careful not to use a bias too large, or you will see a different kind of artifact, particularly visible near corners and edges.

A value less than zero (the default is -1) will use the global `Attribute("shadow:bias")` as the bias amount.

bias = 0.00001
(self-intersection artifacts)



bias = 0.01
(just right)

float maxhitdist = 1.0e6

Objects farther than this distance will be ignored when computing indirect illumination.

float falloff = 0
float falloffmode = 0

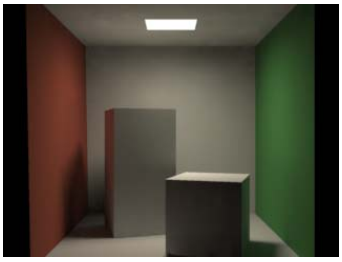
If `falloff` is 0 (the default), all indirect rays hits are weighted in a physically-based manner. But if `falloff` is nonzero, the indirect influence of objects falls off with distance according to `falloffmode` (in both cases, `dist` is the distance to the closest object along the ray):

`falloffmode = 0`
Exponential falloff where the amount of indirect light is weighted by `exp(-falloff/dist)`

`falloffmode = 1`
Polynomial falloff that drops to 0 as distance approaches `maxhitdist`, where the amount of indirect illumination is weighted by `pow (1-dist/maxhitdist, falloff)`

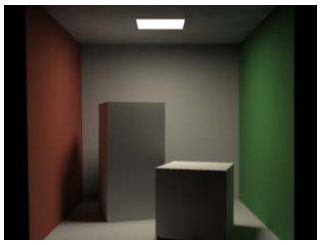
The big difference between the two modes is that with exponential falloff, even very distant objects still contribute a little to the occlusion amount (though the occlusion diminishes rapidly with distance), though occlusion drops immediately to zero once distance is >= `maxhitdist`. In contrast, with polynomial falloff, occlusion smoothly drops to zero as distance approaches `maxhitdist`, with no

abrupt cutoff (and also with a linear or polynomial, rather than exponential diminishing of occlusion).



falloff = 0 (no falloff)
falloffmode = 0

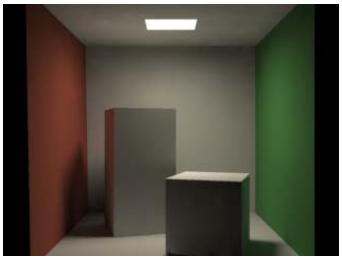
falloff = 0.1
falloffmode = 0



falloff = 1
falloffmode = 1
maxhitdist = 500

float maxlevel = 1

Determines the number of recursive levels for which indirect illumination is accounted. A maxlevel of 0 means that no indirect illumination is used. A `maxlevel` of 1 means that light paths will be included that bounce from light, off one surface, and to the shading point. A `maxlevel` of 2 means that in addition to light->object->shading point paths, it will also account for paths that are light->object1->object2->shading point. And so on. More levels will account for additional light and be more physically accurate, but will also be much more expensive. Each successive level, while increasing expense, will have a smaller noticeable change to the final image. For almost all scenes, `maxlevel` = 1 is sufficient to get a pleasing image.



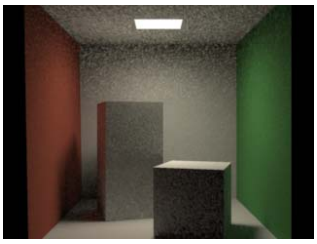
maxlevel = 1
time: 0:52

maxlevel = 2
time: 2:45

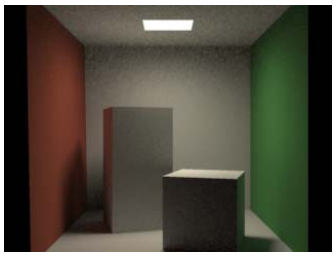
Hints and Tips

The sparse sampling and interpolation results in "blotchy" artifacts, as you can see in some of the examples above. The blotches are less noticeable as `maxerror` and/or `maxpixeldist` is reduced, and as `samples` is increased. Some people prefer sampling noise to blotches, and so leave `maxerror` = 0 but reduce samples in order to

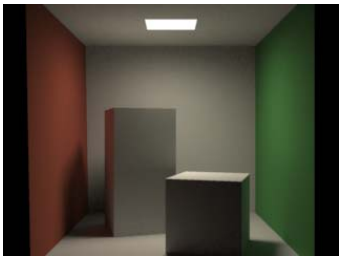
achieve good rendering times, accepting the noise. Below are some examples showing how appearance changes with `samples` when `maxerror` = 0:



maxerror = 0
samples = 16
(0:48)

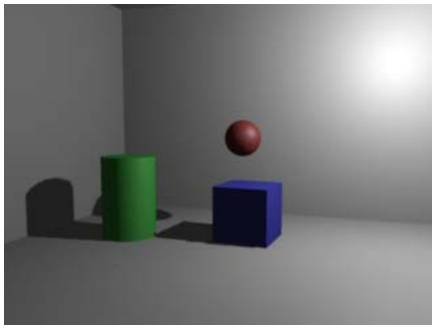


maxerror = 0
samples = 64
(3:42)



maxerror = 0
samples = 256
(12:12)

pointlight.gsl



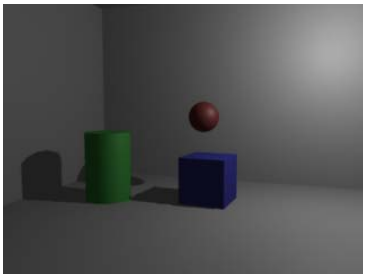
Description

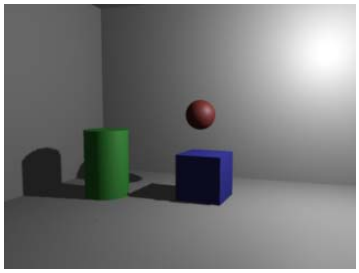
A simple point light with a specific position that casts light in all directions. Shadows (both depth maps and ray tracing) are supported.

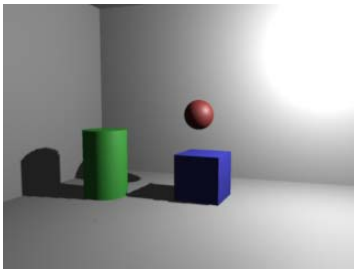
Parameters

float intensity = 1

Controls the intensity (brightness) of the light.

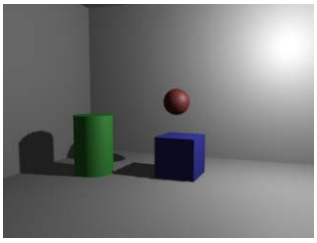

intensity = 10

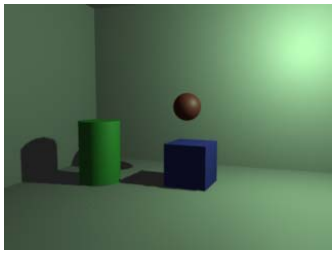

intensity = 25


intensity = 50

color lightcolor = (1,1,1)

Controls the color of the light.


lightcolor = (1,1,1)


lightcolor = (0.5,1,0.5)

point from = point("shader", 0, 0, 0)

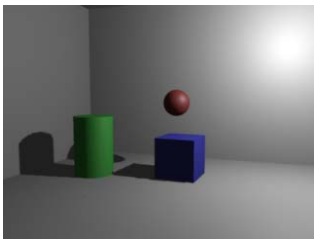
The position from which the light shines (and the shadows will cast) is given by the `from` parameter. The default value makes the light shine from the origin of the light's local coordinate system. It is occasionally helpful to have a different behavior, and overriding the `from` parameter will allow you to make the light originate from arbitrary positions relative to its local coordinate frame.

This parameter is somewhat archaic; its existance is largely for back-compatibility with older renderers and older shaders. We recommend leaving this parameter alone and positioning the light by merely moving the light's local coordinate frame (this is how an interactive modeling system will usually do it).

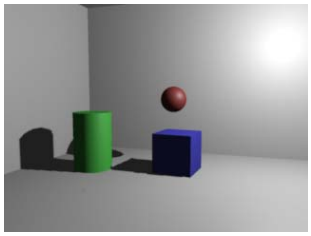
float falloff = 2

Controls the rate at which illumination diminishes with distance from the light source. It is actually the *exponent* to which the distance (in "common" space) is raised. A falloff value of 2 indicates "$1/r^2$" falloff, which is physically realistic but sometimes hard to control. A falloff value of 1 indicates linear (1/r) falloff. A falloff value of 0 indicates no falloff, that is, the light will not appear more dim to distant objects -- a distinctly non-physical effect that nonetheless is sometimes very helpful.

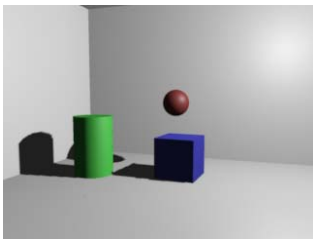
Notice that when using `falloff`, and especially depending on the scale of your scene, you may need to adjust the `intensity` of the light in order to adequately illuminate distant objects (see the examples below).



falloff = 2 (default)
intensity = 25



falloff = 1
intensity = 5

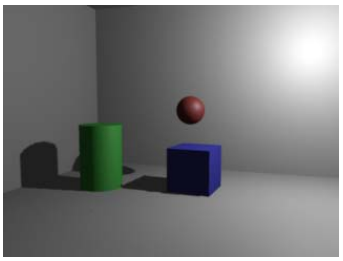
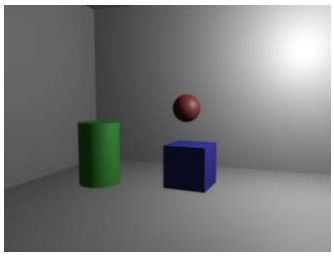


falloff = 0
intensity = 0.75

string shadowname = ""

The `shadowname` parameter specifies how the light should cast shadows. If no shadow name is supplied (the default), this light will not cast shadows. If `shadowname` is the name of a ray-traced geometry set, ray-traced shadows will be used. If `shadowname` is the name of a shadow map, the map will provide shadows (this is true for any of normal shadow maps, Woo shadows, volume shadows, or dynamic shadows).

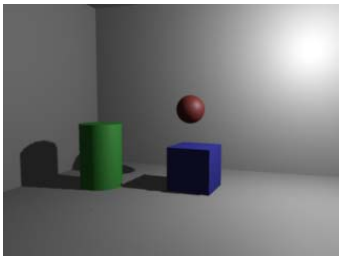
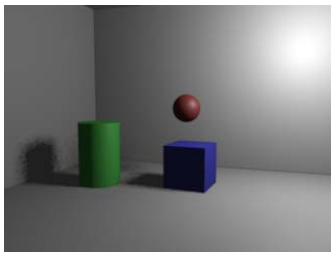
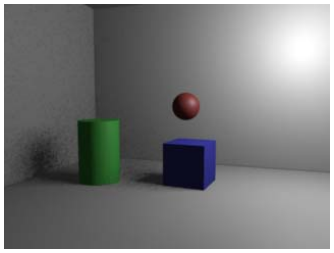
If shadow maps are used, it is strongly recommended that the shadow map be created using a perspective projection, rendered from the position of the light. If shadow-casting objects may be in all directions, you may wish to use a cube-face shadow map.

ray-traced shadows no shadows

float shadowblur = 0

Controls how blurry the shadows are. A value of 0 will make perfectly sharp shadows. Larger values will make the shadows blurrier. This is true of both ray-traced and mapped shadows, although a particular blur value will not make an identical image with the two techniques.

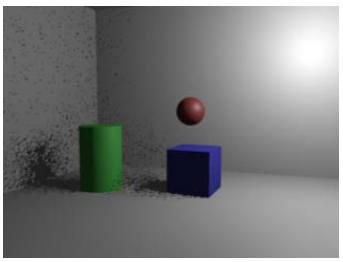
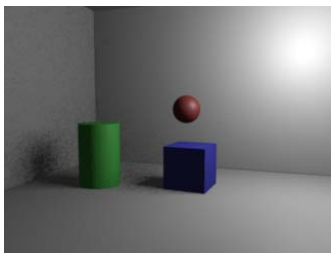
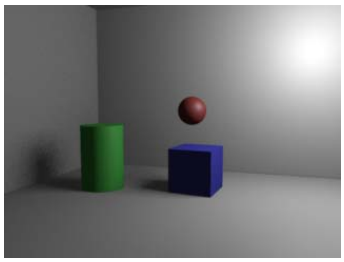




shadowblur = 0 (default) shadowblur = 0.1 shadowblur = 0.25

float shadowsamples = 1

Controls the number of samples used to compute the blurry shadows. As `shadowblur` increases, you may find that you also need to increase `shadowsamples`.

When using shadow maps, Gelato clamps `shadowsamples` at 16; that is, you will always get a minimum of 16 shadow samples for map lookups, even if you ask for less. But for ray-traced shadows, no such minimum exists (since ray tracing is so expensive).

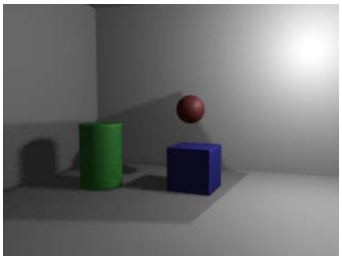
shadowsamples = 1 shadowsamples = 4 shadowsamples = 16

float shadowbias = -1

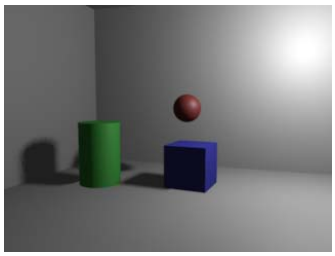
Particularly when using shadow maps, there is a tendency for surfaces to incorrectly shadow themselves due to numerical precision issues. It is customary to add a "bias" to shadow lookups, which simply means a minimum distance that objects will appear to cast shadows. By making this bias larger than the numerical precision limits, you can usually avoid the self-shadowing artifacts.

The default of -1 indicates that the global shadow bias should be used (set by `Attribute("shadow:bias")`). Values larger than zero will set the shadow bias for this light only, independently of the global bias value.

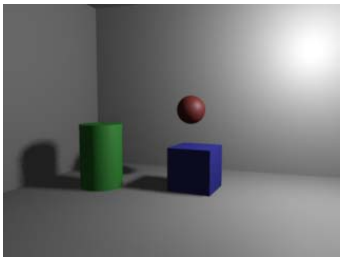
Shadow bias applies to ray tracing as well as shadow map lookups. When using ray-traced shadows, the bias is simply the minimum distance that objects will appear to cast shadows.



shadow bias = 0.025
(too little -- self-shadowing apparent)



shadowbias = 2.5
(too big -- detatched shadows)



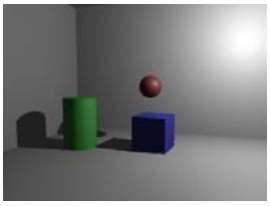
shadowbias = 0.25
(just right)

float __nonspecular = 0

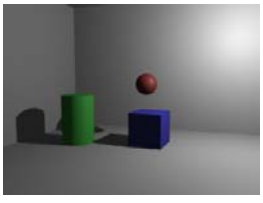
By default, the light will supply both diffuse illumination and specular highlights. By adjusting `__nonspecular` (yes, there are two underscores -- historical compatibility), you can coax the light into not contributing to specular highlights.

By default, `__nonspecular` is zero, so the light will contribute to specular highlights. If `__nonspecular` = 1, the light will not contribute to specular highlights at all. Intermediate values will give partial specularity as expected.

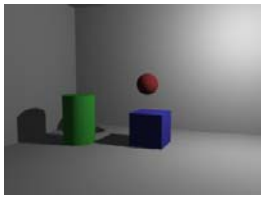
Non-specular lights are good for "fill lights" -- those lights that contribute to overall illumination but that you specifically wish to not add specular highlights to the scene.



__nonspecular = 0

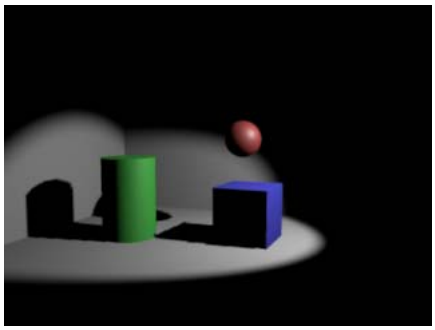


__nonspecular = 0.5



__nonspecular = 1

spotlight.gsl

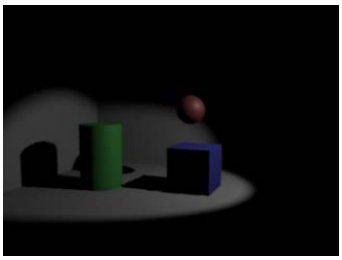


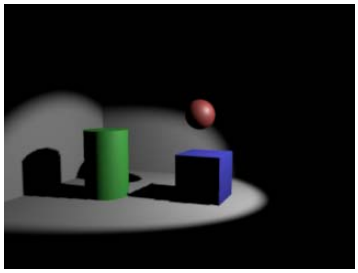
Description

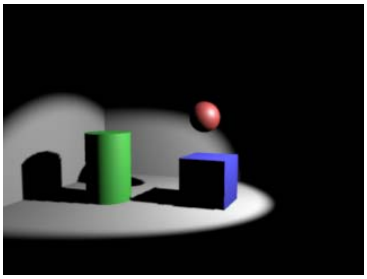
A simple spot light with a circular cross-section. Shadows (both depth maps and ray tracing) are supported.

Parameters

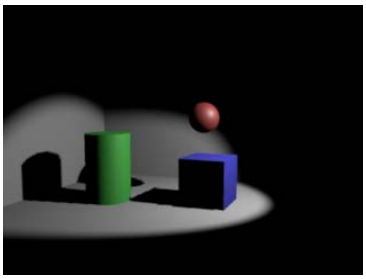
float intensity = 1
Controls the brightness of the light.

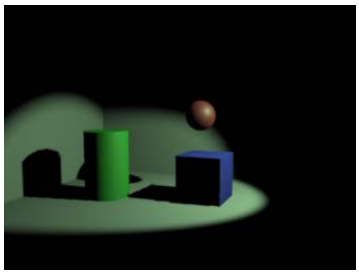

intensity = 10


intensity = 50


intensity = 100

color lightcolor = (1,1,1)
Controls the color of the light.


lightcolor = (1,1,1)


lightcolor = (0.5,1,0.5)

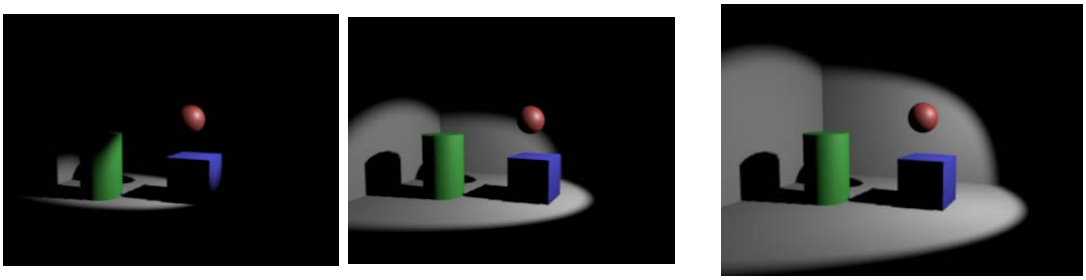
point from = point ("shader", 0, 0, 0)
point to = point ("shader", 0, 0, 1)

The position from which the light shines (and the shadows will cast) is implied by the `from` position and the orientation of the spot light is impied by the vector joining the `from` and `to` positions. The default values of these parameters imply that the light shines from the origin of the light's local coordinate system, and that the spot light is pointed in the direction of the $+z$ axis of the light's local coordinate system. It is occasionally helpful to have a different behavior, and overriding the `from` and `to` parameters will allow you to make the light originate and point in arbitrary directions relative to its local coordinate frame.

These parameters are somewhat archaic; their existance is largely for back-compatibility with older renderers and older shaders. We recommend leaving these parameters alone and pointing the light by merely positioning and rotating the light's local coordinate frame (this is how an interactive modeling system will usually do it).

float coneangle = 0.5236 // = radians(30)

The half-angle of the spot cone, i.e., the angular deviation from the spot light axis direction at which light is completely cut off. This angle is expressed in radians (PI/2 radians are 90 degrees).



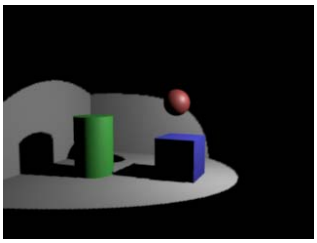
coneangle = 0.3

coneangle = 0.5236 (default)

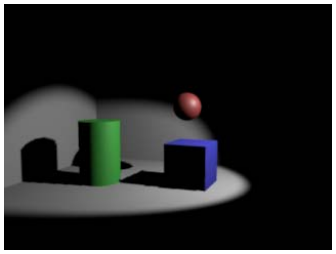
coneangle = 0.7

float conedeltaangle = 0.10472 // = radians(5)

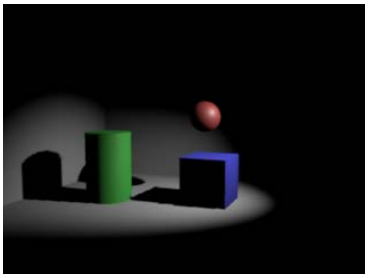
The angle *within* `coneangle` over which the light fades out at the edge of the cone. This angle is expressed in radians (PI/2 radians are 90 degrees). Small values for `conedeltaangle` make a very sharp transition at the edge of the spotlight cone. Larger values make smoother transitions.



conedeltaangle = 0



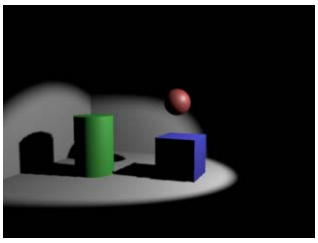
conedeltaangle = 0.10472 (default)



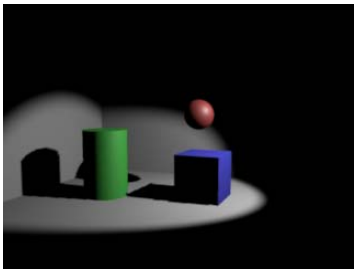
conedeltaangle = 0.5

float beamdistribution = 2

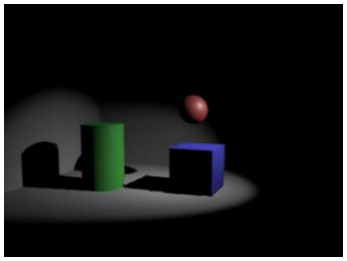
Describes the angular falloff -- that is, how rapidly the intensity of light diminishes as the angle deviates from the axial direction of the spot light. A `beamdistribution` of 0 will result in no angular falloff. Larger values will result in more rapid diminishing of light across the face of the spot. A value of 2 (the default) is somewhat physically motivated and is only slightly brighter near the center of the spotlight than toward the edges.



beamdistribution = 0



beamdistribution = 2

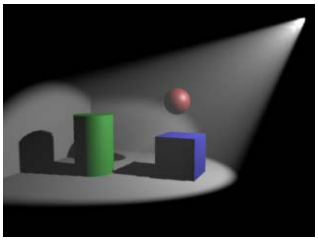


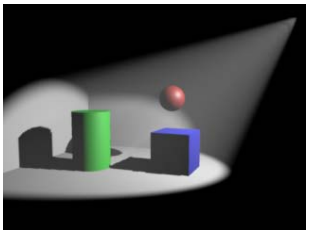
beamdistribution = 20

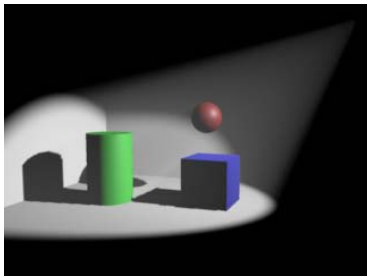
float falloff = 2

Controls the rate at which illumination diminishes with distance from the light source. It is actually the *exponent* to which the distance (in "common" space) is raised. A falloff value of 2 indicates "$1/r^2$" falloff, which is physically realistic but sometimes hard to control. A falloff value of 1 indicates linear (1/r) falloff. A falloff value of 0 indicates no falloff, that is, the light will not appear more dim to distant objects -- a distinctly non-physical effect that nonetheless is sometimes very helpful.

Notice that when using `falloff`, and especially depending on the scale of your scene, you may need to adjust the `intensity` of the light in order to adequately illuminate distant objects (see the examples below). In the images below, we have introduced "fog" to more easily visualize how the falloff works.


falloff = 2 (default)
intensity = 50

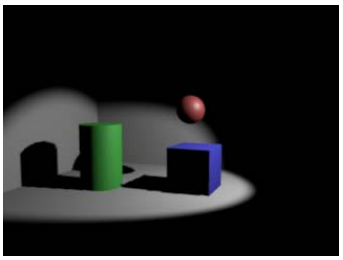

falloff = 1
intensity = 10

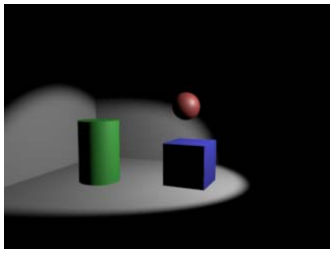

falloff = 0
intensity = 1

string shadowname = ""

The `shadowname` parameter specifies how the light should cast shadows. If no shadow name is supplied (the default), this light will not cast shadows. If `shadowname` is the name of a ray-traced geometry set, ray-traced shadows will be used. If `shadowname` is the name of a shadow map, the map will provide shadows (this is true for any of normal shadow maps, Woo shadows, volume shadows, or dynamic shadows).

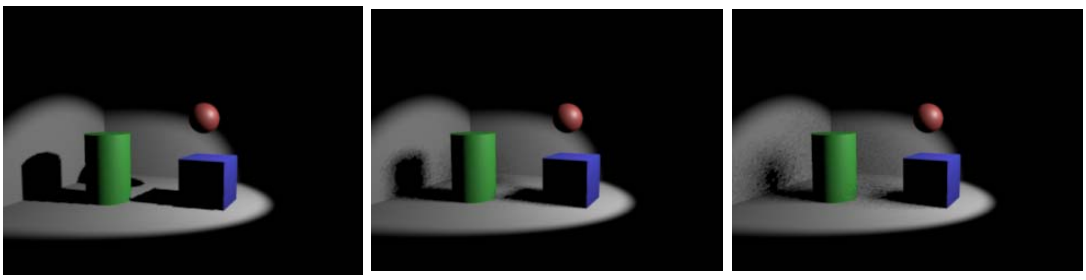
If shadow maps are used, it is strongly recommended that the shadow map be created using a perspective projection, rendered from the position of the light. If shadow-casting objects may be in all directions, you may wish to use a cube-face shadow map.


ray-traced shadows


no shadows

float shadowblur = 0

Controls how blurry the shadows are. A value of 0 will make perfectly sharp shadows. Larger values will make the shadows blurrier. This is true of both ray-traced and mapped shadows, although a particular blur value will not make an identical image with the two techniques.

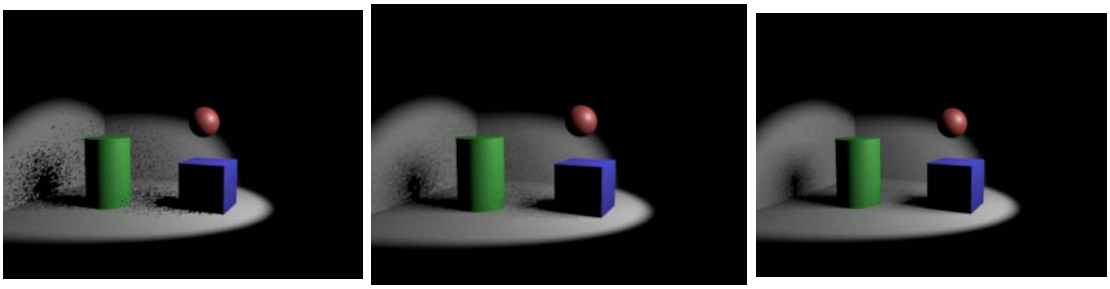


shadowblur = 0 (default) shadowblur = 0.1 shadowblur = 0.25

float shadowsamples = 1

Controls the number of samples used to compute the blurry shadows. As `shadowblur` increases, you may find that you also need to increase `shadowsamples`.

When using shadow maps, Gelato clamps `shadowsamples` at 16; that is, you will always get a minimum of 16 shadow samples for map lookups, even if you ask for less. But for ray-traced shadows, no such minimum exists (since ray tracing is so expensive).



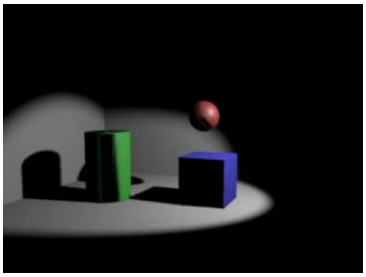
shadowsamples = 1 shadowsamples = 4 shadowsamples = 16

float shadowbias = -1

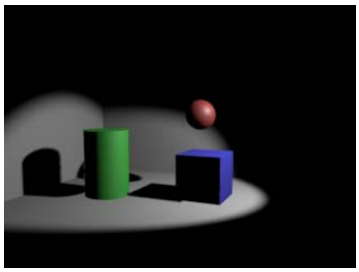
Particularly when using shadow maps, there is a tendency for surfaces to incorrectly shadow themselves due to numerical precision issues. It is customary to add a "bias" to shadow lookups, which simply means a minimum distance that objects will appear to cast shadows. By making this bias larger than the numerical precision limits, you can usually avoid the self-shadowing artifacts.

The default of -1 indicates that the global shadow bias should be used (set by `Attribute("shadow:bias")`). Values larger than zero will set the shadow bias for this light only, independently of the global bias value.

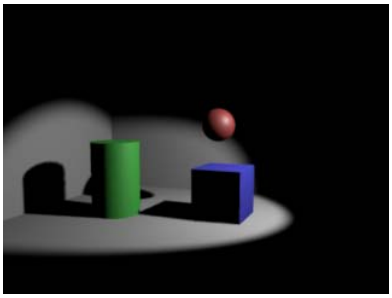
Shadow bias applies to ray tracing as well as shadow map lookups. When using ray-traced shadows, the bias is simply the minimum distance that objects will appear to cast shadows.



shadow bias = 0.025
(too little -- self-shadowing apparent)



shadowbias = 2.5
(too big -- detatched shadows)



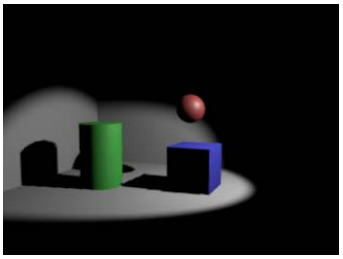
shadowbias = 0.25
(just right)

float __nonspecular = 0

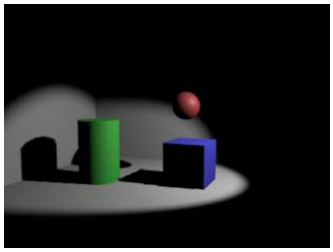
By default, the light will supply both diffuse illumination and specular highlights. By adjusting `__nonspecular` (yes, there are two underscores -- historical compatibility), you can coax the light into not contributing to specular highlights.

By default, `__nonspecular` is zero, so the light will contribute to specular highlights. If `__nonspecular` = 1, the light will not contribute to specular highlights at all. Intermediate values will give partial specularity as expected.

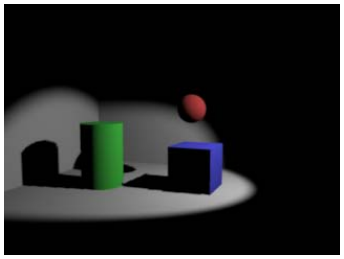
Non-specular lights are good for "fill lights" -- those lights that contribute to overall illumination but that you specifically wish to not add specular highlights to the scene.



__nonspecular = 0

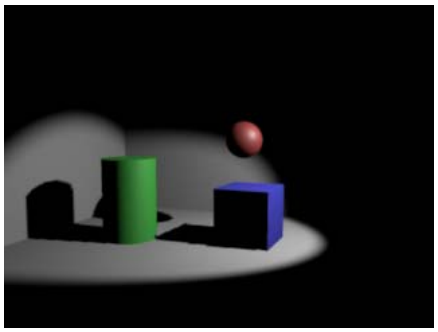


__nonspecular = 0.5



__nonspecular = 1

uberlight.gsl



Description

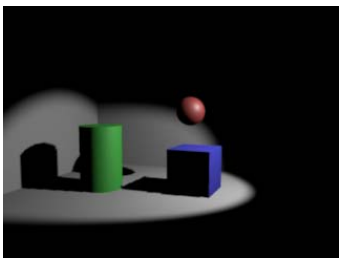
A very flexible light with oodles of artistic controls. It can behave like a point light or like a spot light. When acting like a spot light, there are many controls for the shape of the cross-section of the spot. Many shadow modes are supported.

The light shines from the origin of the light's local coordinate system. If it is a "spot" light, the one will be pointed either in the direction of the +*z* or -*z* axis of the light's local coordinate system (depending on the `zreverse` parameter). Therefore the light is pointed by positioning and rotating the light's local coordinate frame.

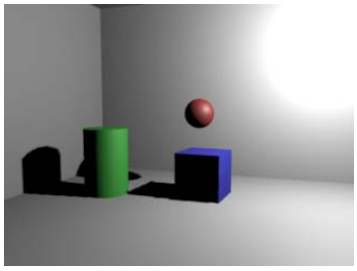
Parameters

string lighttype = "spot"

The type of light source. If "spot" (the default), the light will behave roughly like a spotlight, illuminating a cone (though a very flexibly-shaped cone). If "omni", the light will behave like a pointlight, illuminating in all directions.



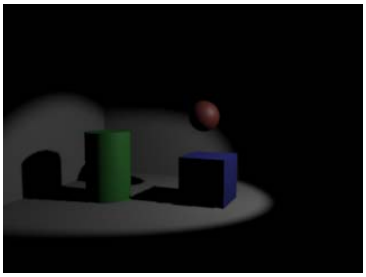
lighttype = "spot"

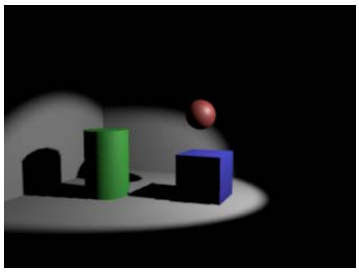


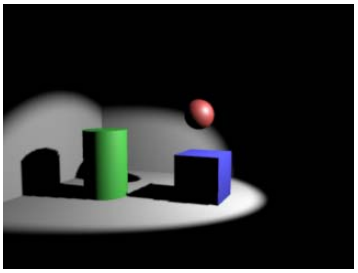
lighttype = "omni"

float intensity = 1

Controls the brightness of the light.

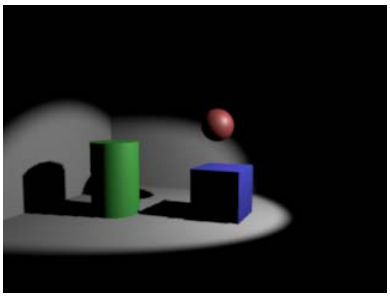

intensity = 10

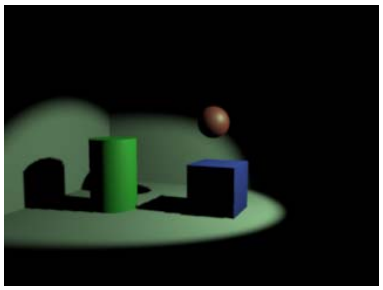

intensity = 50


intensity = 100

color lightcolor = (1,1,1)

Controls the color of the light.


lightcolor = (1,1,1)


lightcolor = (0.5,1,0.5)

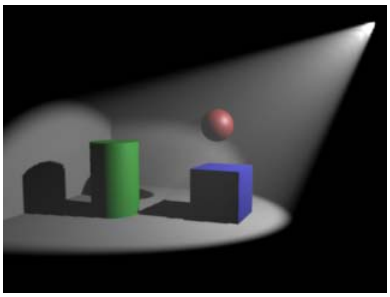
float falloff = 0
float falloffdist = 1

Controls the rate at which illumination diminishes with distance from the light source. The `falloff` is actually the *exponent* to which the distance (in "common" space) is raised. A falloff value of 2 indicates "$1/r^2$" falloff, which is physically realistic but sometimes hard to control. A falloff value of 1 indicates linear (1/r) falloff. A falloff value of 0 (the default) indicates no falloff, that is, the light will not appear more dim to distant objects -- a distinctly non-physical effect that nonetheless is sometimes very helpful.

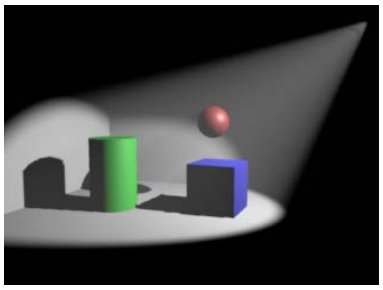
`falloffdist` is the distance at which the incident energy is actually equal to `intensity*lightcolor`. In other words, the intensity is related to distance by:

```
I = pow (falloffdist / distance, falloff)
```

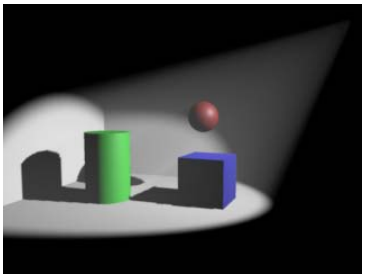
Notice that when using `falloff`, and especially depending on the scale of your scene, you may need to adjust the `intensity` of the light in order to adequately illuminate distant objects (see the examples below). In the images below, we have introduced "fog" to more easily visualize how the falloff works.



falloff = 2
falloffdist = 1
intensity = 50



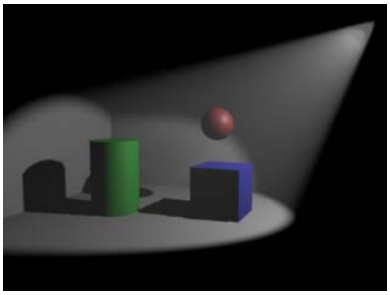
falloff = 1
falloffdist = 1
intensity = 10



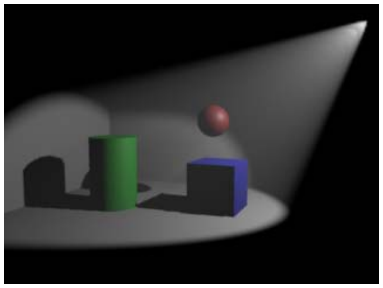
falloff = 0
falloffdist = 1
intensity = 1

float maxintensity = 2

To prevent the light from becoming unboundedly bright when the distance is less than `falloffdist`, the intensity is smoothly clamped to `maximumintensity`. The default value of 2 has no basis in physical reality, but is very handy to ensure that you do not over-illuminate your scene for objects close to the light. Physically-correct lights (if they are point-like) continue to get brighter as you get closer to them, so to ensure physical accuracy if you are using very bright lights, you may wish to use a much larger value for `maxintensity`.



maxintensity = 2
falloff = 2
intensity = 25

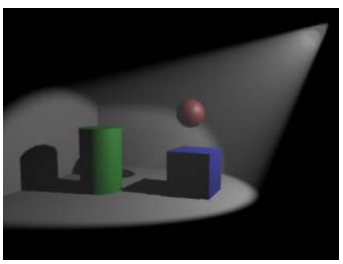


maxintensity = 2000
falloff = 2
intensity = 25

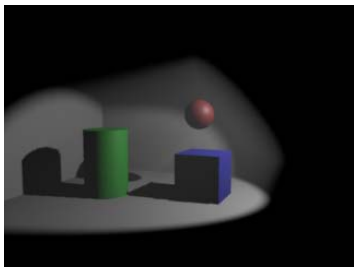
float cuton = 0.01
float cutoff = 1.0e6

These parameters define the depth range (that is, the *z* range from the light's local origin) over which the light is active. By default, they illuminate over the entire range that you are likely to need, though depending on the size of your scene you may need to adjust them if you have objects closer or farther from your light. In

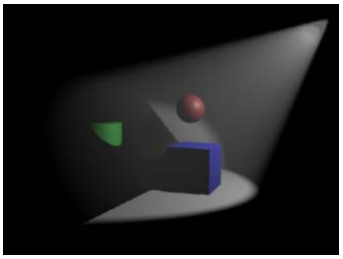
the real world, of course, an actual light would illuminate at all depths. But for reasons of artistic control, it can be handy to use these non-physical controls to limit the depth range that your light illuminates.



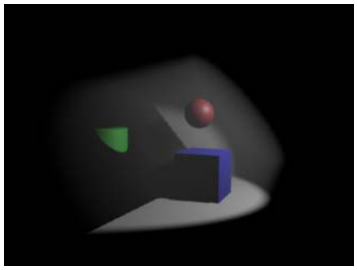
defaults



cuton = 5, default cutoff



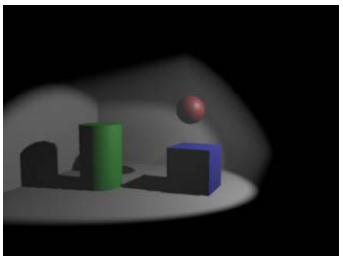
cutoff = 10, default cuton



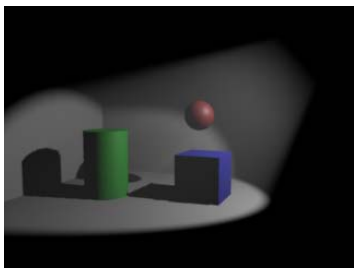
cuton = 5, cutoff = 10

float nearedge = 0
float faredge = 0

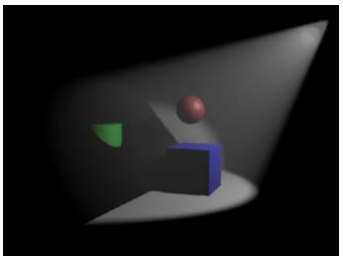
These parameters define the width (in depth range) of the transition retion of `cuton` and `cutoff`. The light will smoothly turn on as depth ranges between `cuton-nearedge` and `cuton`, and will smoothly turn off between `cutoff` and `cutoff+faredge`.



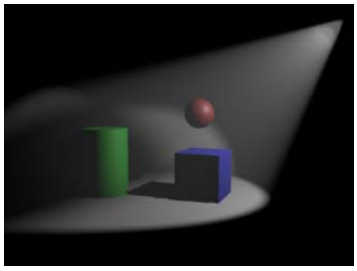
cuton = 5, nearedge = 0



cuton = 5, nearedge = 3



cutoff = 10, faredge = 0



cutoff = 10, faredge = 3

float parallelrays = 0

If zero (the default) light rays and shadows will eminate from the origin of the light's local coordinate system. That is, the light will be emitted radially from that point and the light rays diverge. But if `parallelrays` is nonzero, the light rays will be parallel, as if the light source was infinitely far away (such as from the sun).

You can make an uberlight behave more like a `distantlight` by setting `lighttype = "spot"` and `parallelrays = 1`.

float zreverse = 0

If zero (the default), the light rays will radiate in the direction of the light's local coordinate system's +*z* axis. If `zreverse` is nonzero, the light rays will radiate in the -*z* direction, thus reversing the direction of the light. This is useful when using uberlight within a modeling system that otherwise adheres to the "light in -*z*" convention (such as Maya), and whose UI's insist on pointing their lights in the -*z* direction.

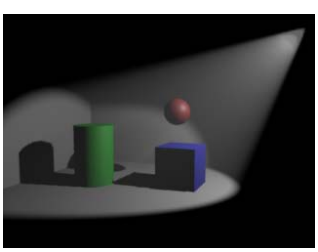
float shearx = 0
float sheary = 0

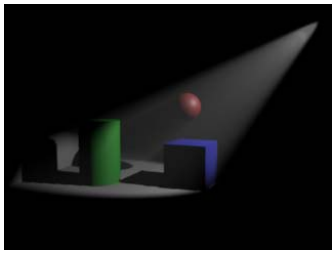
Define the amount of shear applied to the light cone direction, making an off-center cone. shearx shears in the light's local +/- *x* direction, while sheary shears in the light's local +/- *y* direction. A value of zero indicates no shear, that is, the light cone is straight.

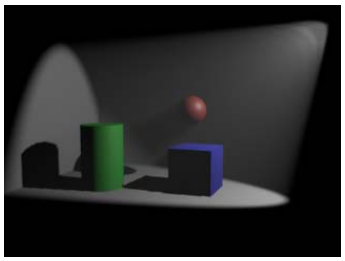
float width = 1
float height = 1

Define the dimensions of the "barn door" opening of the light. They are the cross-sectional dimensions at a distance of 1 from the light. If `width=1` and `height=1`, the light will have a 90 degree full cone angle (that means that they define the tangent of the cross-sectional cone angle). Smaller numbers make a smaller angle to the light cone; larger numbers make a wider light cone angle. If `width` and `height` are equal, the cone cross-sectional shape will be just as tall as it is wide (that is, circular, if `roundness` = 1); if `width` and `height` are not equal, the light cross-sectional shape will be wider than it is tall, or vice versa (an ellipse, rather than a circle).

It should be clear that the function of `width` and `height` is much like the `coneangle` parameter of spotlight.gsl, except that it's measured as the tangent rather than the angle, and can be controlled separately in the two major directions.


width = 0.5, height = 0.5

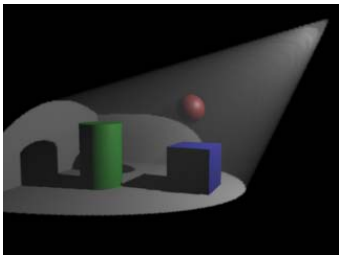

width = 0.25, height = 0.25


width = 0.25, height = 0.75

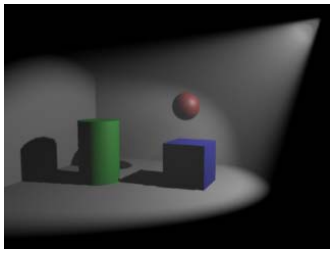
float hedge = 0.1
float wedge = 0.1

The amount of "fuzz" to the `width` and `height`, that is, how rapidly the light cuts off at the edges of the cone.

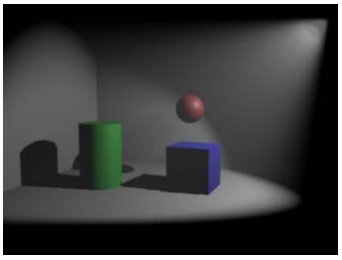
It should be clear that the function of `wedge` and `hedge` is much like the `conedeltaangle` parameter of spotlight.gsl, except that it's measured as the fraction of width and height rather as the angle, and can be controlled separately in the two major directions.



wedge = 0, hedge = 0



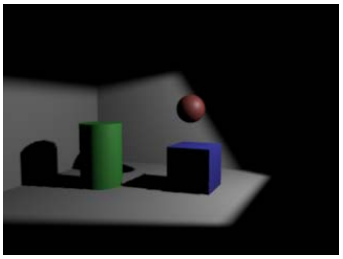
wedge = 0.25, hedge = 0.25



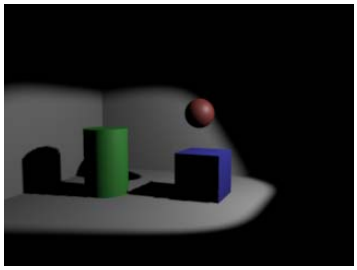
wedge = 0.1, hedge = 0.5 (blurrier edge vertically than horizontally)

float roundness = 1

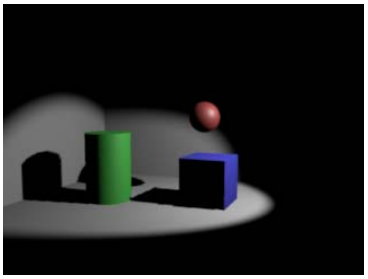
The cross-sectional shape of the uberlight is a *superellipse*, which is a shape that may be smoothly varied between a circle and a square. The roundness Controls how rounded are the corners of the light's cross-sectional shape. When roundness = 1, the cross-sectional shape will be circular. When roundness = 0, the cross-sectional shape will be a square. Intermediate values will make squarish shapes with rounded edges, as expected.



roundness = 0



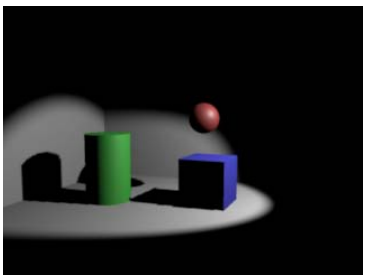
roundness = 0.5

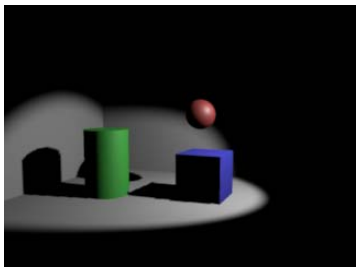


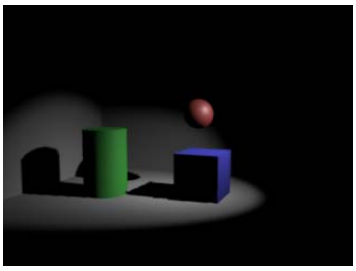
roundness = 1 (default)

float beamdistribution = 2

Describes the angular falloff -- that is, how rapidly the intensity of light diminishes as the angle deviates from the axial direction of the spot light. A `beamdistribution` of 0 will result in no angular falloff. Larger values will result in more rapid diminishing of light across the face of the spot. A value of 2 (the default) is somewhat physically motivated and is only slightly brighter near the center of the uberlight than toward the edges.

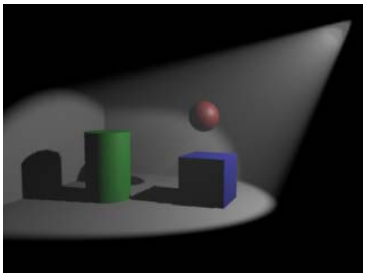

beamdistribution = 0

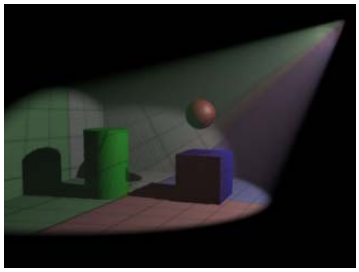

beamdistribution = 2


beamdistribution = 20

string slidename = ""

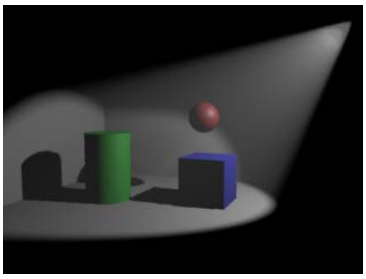
If a file name is supplied, the file will be used as a texture that will filter the color of the light (based on the cross-section). This allows you to use a texture to create a custom shape for the light cross-section, or to color the cross-section as if the light was being passed through a colored material.


slidename = ""

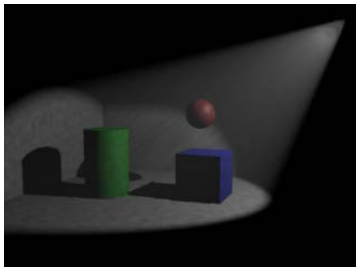

slidename = "grid.tx"

float noiseamp = 0
float noisefreq = 4
vector noiseoffset = 0

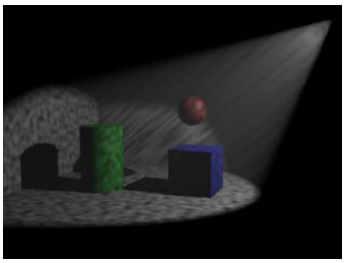
These parameters allow you to add some variation, or noise, so the light intensity across the profile of the light cross-section. The `noiseamp` parameter controls the amplitude of the noise, with 0 indicating no noise effect. The `noisefreq` controls the frequency of the noise (higher frequencies mean smaller "grains"). The `noiseoffset` is an offset into the "noise field," this allowing you to select a different noise pattern (or even animate the noise pattern.



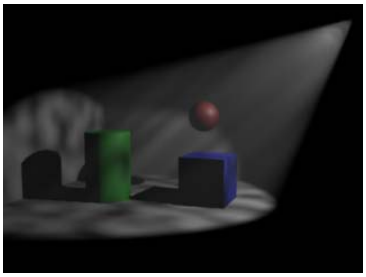
noiseamp = 0 (default)



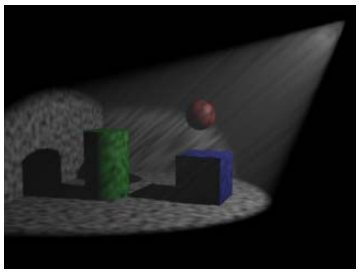
noiseamp = 0.25



noiseamp = 1



noisefreq = 1
noiseamp = 1

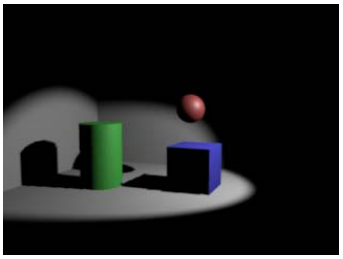
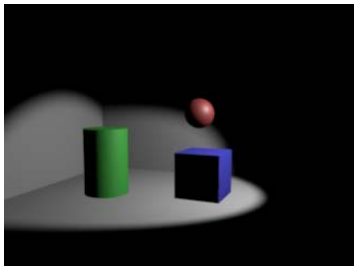


noisefreq = 4 (default)
noiseamp = 1

string shadowname = ""

The `shadowname` parameter specifies how the light should cast shadows. If no shadow name is supplied (the default), this light will not cast shadows. If `shadowname` is the name of a ray-traced geometry set, ray-traced shadows will be used. If `shadowname` is the name of a shadow map, the map will provide shadows (this is true for any of normal shadow maps, Woo shadows, volume shadows, or dynamic shadows).

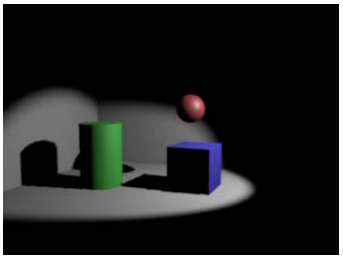
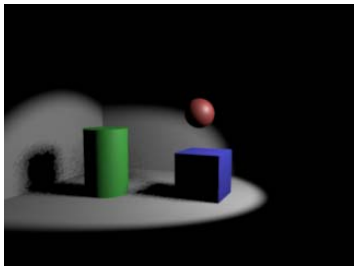
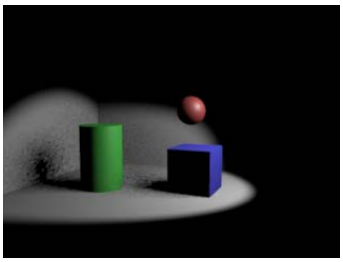
If shadow maps are used, it is strongly recommended that the shadow map be created using a perspective projection, rendered from the position of the light. If shadow-casting objects may be in all directions, you may wish to use a cube-face shadow map.

ray-traced shadows no shadows

float shadowblur = 0

Controls how blurry the shadows are. A value of 0 will make perfectly sharp shadows. Larger values will make the shadows blurrier. This is true of both ray-traced and mapped shadows, although a particular blur value will not make an identical image with the two techniques.

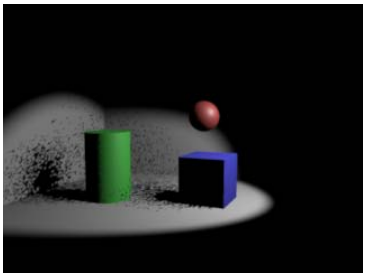
shadowblur = 0 (default) shadowblur = 0.1 shadowblur = 0.25

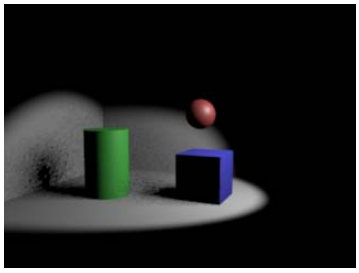
float shadowsamples = 1

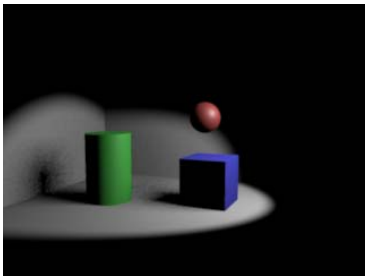
Controls the number of samples used to compute the blurry shadows. As `shadowblur` increases, you may find that you also need to increase `shadowsamples`.

When using shadow maps, Gelato clamps `shadowsamples` at 16; that is, you will always get a minimum of 16 shadow samples for map lookups, even if you

ask for less. But for ray-traced shadows, no such minimum exists (since ray tracing is so expensive).


shadowsamples = 1


shadowsamples = 4

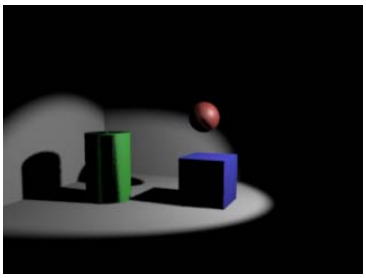

shadowsamples = 16

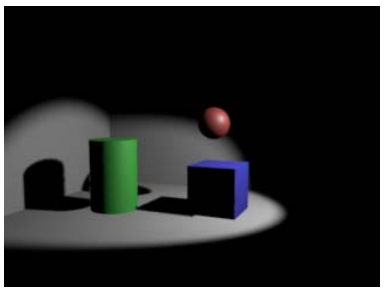
float shadowbias = -1

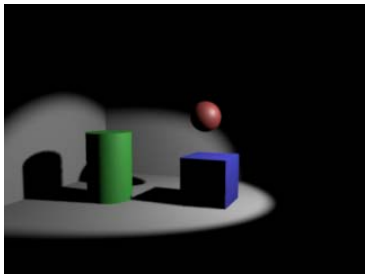
Particularly when using shadow maps, there is a tendency for surfaces to incorrectly shadow themselves due to numerical precision issues. It is customary to add a "bias" to shadow lookups, which simply means a minimum distance that objects will appear to cast shadows. By making this bias larger than the numerical precision limits, you can usually avoid the self-shadowing artifacts.

The default of -1 indicates that the global shadow bias should be used (set by `Attribute("shadow:bias")`). Values larger than zero will set the shadow bias for this light only, independently of the global bias value.

Shadow bias applies to ray tracing as well as shadow map lookups. When using ray-traced shadows, the bias is simply the minimum distance that objects will appear to cast shadows.

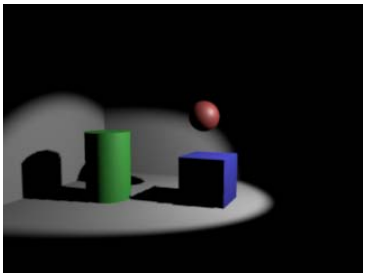

shadow bias = 0.025
(too little -- self-shadowing apparent)


shadowbias = 2.5
(too big -- detatched shadows)

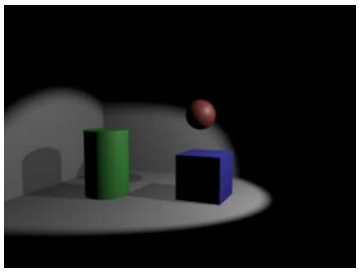

shadowbias = 0.25
(just right)

color shadowcolor = (0, 0, 0)

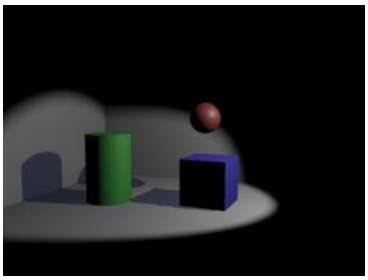
The uberlight shader does not actually block light when shadows are present. Rather, the *color* of the light is scaled by `shadowcolor` when in shadow. By default, `shadowcolor` is black, so light appears to be completely blocked. But you can use `shadowcolor` to cause shadows to only partially block light, or simply to tint light.



shadowcolor = (0,0,0)
(default)



shadowcolor = (0.5, 0.5, 0.5)
(partially block light with shadows)

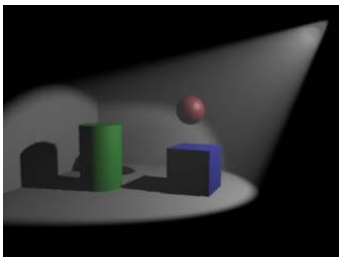


shadowcolor = (.25, .25, .5)
(cast bluish shadows)

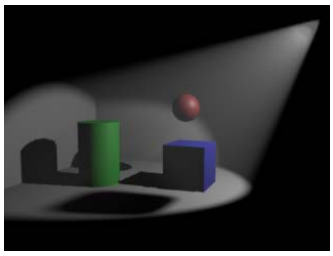
string blockercoords = ""
float blockerwidth = 1
float blockerheight = 1
float blockerwedge = 0.1
float blockerhedge = 0.1
float blockerround = 1

These parameters allow you to place a "blocker" -- essentially an area that blocks light in the scene, even though it does not correspond to a real object. The shape of the blocker is a superellipse lying on the *x*-*y* plane and of the coordinate system named by `blockercoords`; the blocker can be moved around in 3D space by positioning and orienting that coordinate system (for example in Maya, with a "locator node"). The size of the superellipse is given by `blockerwidth` and `blockerheight`, its roundness is given by `blockerround`, and the "softness" of its edges by `blockerhedge` and `blockerwedge`. Note the correspondence to the similarly-named parameters that control the shape of the cross-section of the light cone.

The main use of blockers is to help "shape" the light by cutting off light in ways that can't be accomplished with any of the other parameters of this shader, but without actually adding real objects in the scene. An alternate approach (possibly easier in many modeling systems) would be to add actual objects to the scene that cast shadows but are not visible to the camera.



no blocker



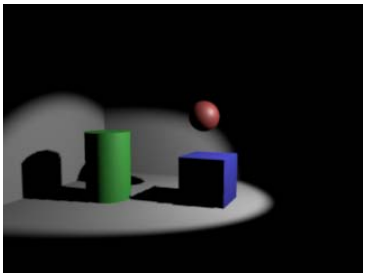
blocker
(note "extra" shadow from blocker)

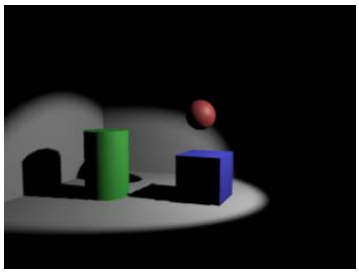
float __nonspecular = 0

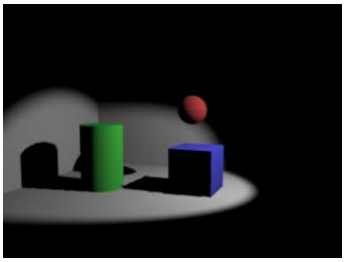
By default, the light will supply both diffuse illumination and specular highlights. By adjusting `__nonspecular` (yes, there are two underscores -- historical compatibility), you can coax the light into not contributing to specular highlights.

By default, `__nonspecular` is zero, so the light will contribute to specular highlights. If `__nonspecular` = 1, the light will not contribute to specular highlights at all. Intermediate values will give partial specularity as expected.

Non-specular lights are good for "fill lights" -- those lights that contribute to overall illumination but that you specifically wish to not add specular highlights to the scene.


__nonspecular = 0


__nonspecular = 0.5

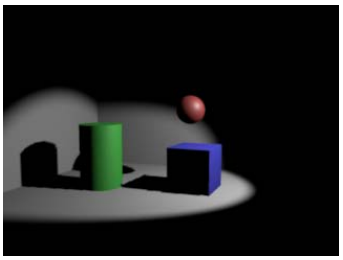

__nonspecular = 1

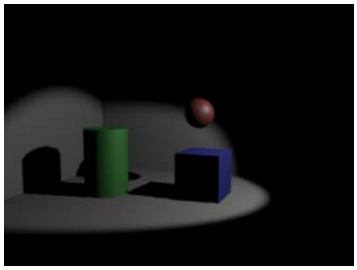
float __nondiffuse = 0

By default, the light will supply both diffuse illumination and specular highlights. By adjusting `__nondiffuse` (yes, there are two underscores -- historical compatibility), you can coax the light into not contributing to diffuse illumination.

By default, `__nondiffuse` is zero, so the light will contribute to diffuse illumination. If `__nondiffuse` = 1, the light will not contribute to diffuse illumination at all. Intermediate values will give partial specularity as expected.

Non-diffuse lights are good when you want to add just a specular highlight, but not to otherwise contribute to the overall illumination of the scene.


__nondiffuse = 0


__nondiffuse = 0.5


__nondiffuse = 1

float __foglight = 1

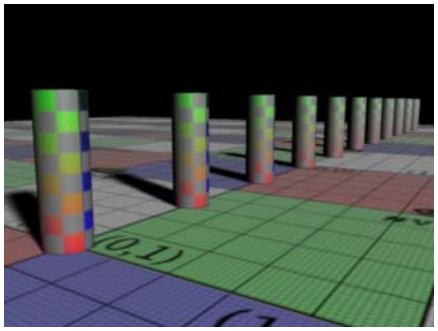
Set `__foglight` (yes, two underscores again) to 0 in order to get the `smoke.gsl` shader, and any other volume shaders that follow this convention, to *exclude* this light from the volumetric effects. This is very handy if you want a light to illuminate objects but not to light up any fog, smoke, or other volumetric effects.

Acknowledgements and References

The functionality of this shader is based on the uberlight presented in Apodaca & Gritz, *Advanced RenderMan: Creating CGI for Motion Pictures*, Morgan-Kaufmann, 1999 (ISBN 1-55860-618-1), which in turn was based on the lighting model presented by Ronen Barzel, "Lighting controls for computer cinematography," *Journal of Graphics Tools* **2**(1): 1-20, 1997.

Volume Shaders

desatdist.gsl



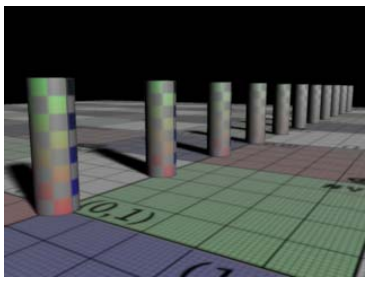
Description

This volume shader desaturates the color of objects by an amount that varies exponentially with the distance of the object (with more distant objects being more desaturated, and closer objects retaining their original color). This provides a visual cue that certain objects are very distant. This is not a very realistic simulation of atmospheric phenomena (compared to the `smoke.gsl` shader, for example), but its performance is great -- it's practically free -- and may be a decent (though slightly cartoonish) substitute under many circumstances..

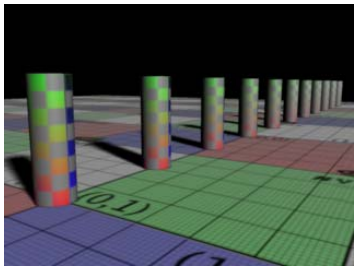
Parameters

float extinctiondist = 100

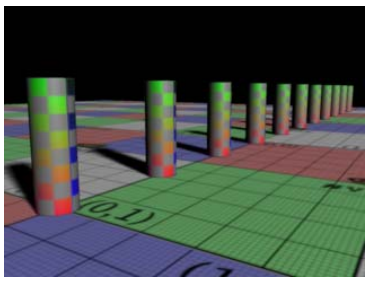
The distance at which the saturation falls to 1/e of its original value (e is Euler's number, approximately 2.71828). This is the basic parameter that controls the rate of falloff -- how quickly objects appear desaturated as they move into the background. The scale in which this parameter is measured is given by the `units` parameter.



extinctiondist = 10



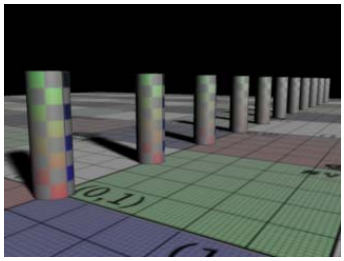
extinctiondist = 20



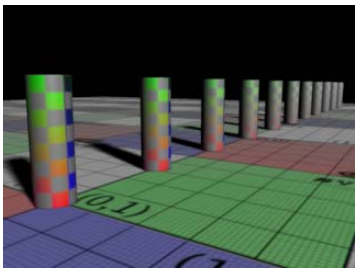
Extincttiondist = 200

float mindist = 0

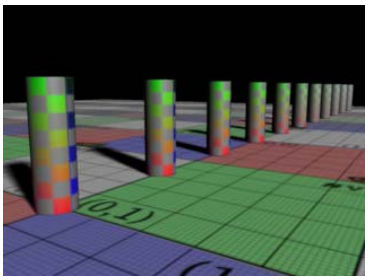
The saturation falloff starts at this distance from the camera. That is, objects closer than this distance will not have their apparent saturation altered at all. The scale in which this parameter is measured is given by the `units` parameter.



mindist = 0
extinctiondist = 10



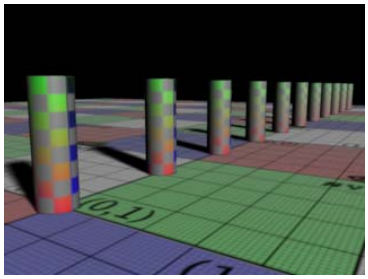
mindist = 10
extinctiondist = 10



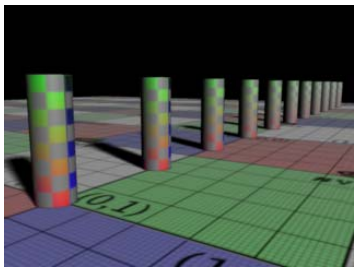
mindist = 20
extinctiondist = 10

float maxdist = 1e6

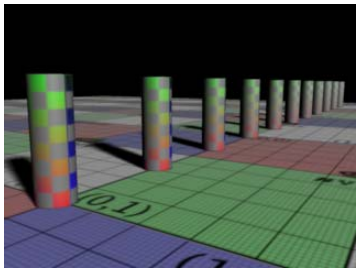
The saturation falloff ends at this distance. That is, objects at this distance are as desaturated as they'll get; farther objects will be no more desaturated than if they were at this distance. The scale in which this parameter is measured is given by the `units` parameter.



maxdist = 20
extinctiondist = 20



maxdist = 40
extinctiondist = 20

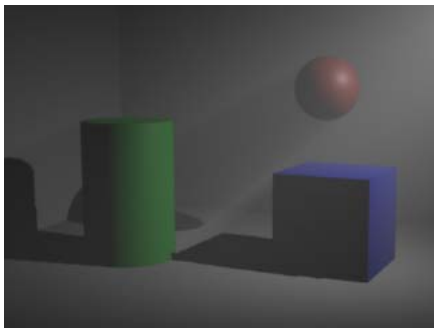


maxdist = 1e6
extinctiondist = 20

string units = "common"

This parameter gives the units in which the other distance parameters (`extinctiondist`, `mindist`, `maxdist`) are measured. By default, distances are measured simply in "common" space units. But if your scene specifies its physical units using `Attribute("string units:length")`, then you may use any units as the distance scale (for example, `"m"`, `"km"`, `"ft"`). The value for `units` does not need to match the scene modeling units; in fact, that's the point -- this lets you specify shader behavior in physical units independently of the units used to model your scene.

smoke.gsl



Description

A volume shader for atmospheric effects such as fog, smoke, or other fine particles suspended in the atmosphere.

The two main atmospheric effects are (1) extinction, the absorption and/or out-scattering of light as it travels through the volume; and (2) in-scattering, the tendency of the particles to reflect light toward the view direction.

The basic technique used is *ray marching*, which steps along the viewing ray (between the camera and shaded object), sampling the light and smoke density at each step. There is a basic trade-off between rendering speed and accuracy, depending on the number of steps taken along the viewing ray.

Note: when gathering light, this volume shader will scale contribution of any lights that has a message (or output variable) called `__foglight` (yes, two underscores) by the value of that variable. Lights with no __foglight message will be handled as usual. Note that lights with a __foglight message whos value is 0 will be ignored by the smoke.

Parameters

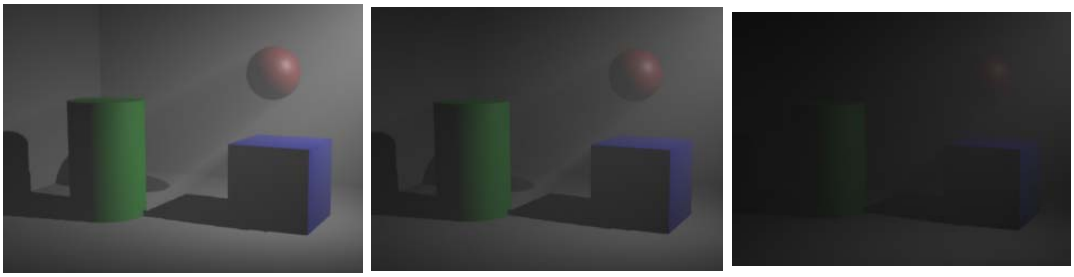
float opacdensity = 1

Density of the material, as it affects its tendency to absorb light (extinction). Higher numbers increase the tendency of the volume to block the view of distant objects.

The appropriate value is quite dependent on the scale of your scene, so don't worry if the default (1) is wildly inappropriate for your scene.

If you can't see as far into the volume as you'd like, lower `opacdensity`.

A value of 0 indicates a volume that doesn't absorb the light from objects behind it at all, but only experiences in-scattering.

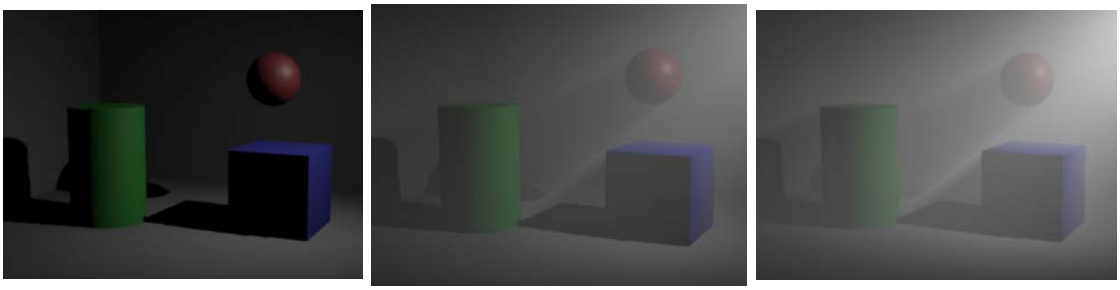


opacdensity = 0 opacdensity = 0.3 opacdensity = 1

float lightdensity = 1

Density of the material, as it affects its tendency to scatter light into the view direction. Higher numbers will make the smoke reflect more light.

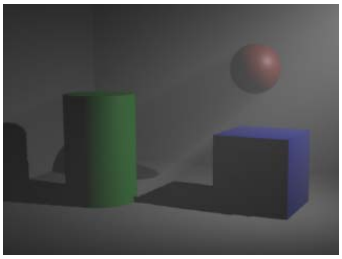
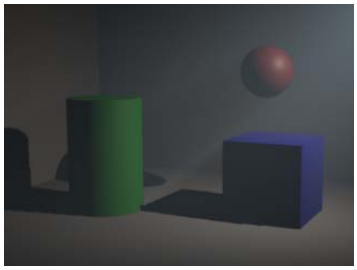
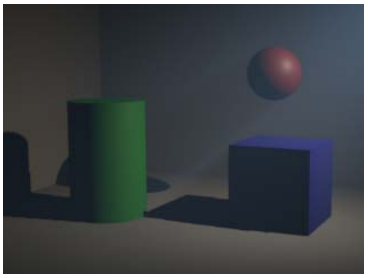
The appropriate value is quite dependent on the scale of your scene, so don't worry if the default (1) is wildly inappropriate for your scene.



lightdensity = 0 lightdensity = 0.2 lightdensity = 0.5

color scatter = 1

Gives a wavelength-dependent scaling of the tendency to scatter light. In real air, atmospheric dust tends to scatter shorter wavelengths (blue) more than longer wavelengths (red). This is why sunsets are red and the sky is blue. This effect can be achieved by giving a scatter parameter with lower values for blue than red.

scatter = (1, 1, 1) scatter = (0.5, 0.75, 1) scatter = (0.25, 0.5, 1)

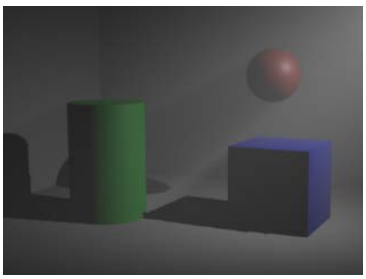
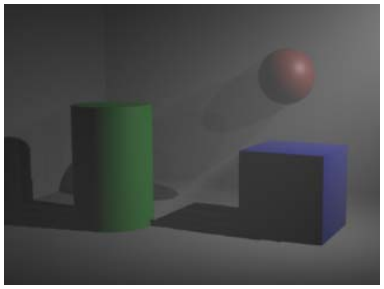
float integstart = 0
float integend = 1000

Controls the distance from the camera at which the 'integration' (summing of volumetric properties) starts and ends. Appropriate values are dependent on the scale of your scene.

These parameters may be used for efficiency -- limiting the expensive integration to only those depths and distances in which "interesting" things are happening. They may also be used to limit the volumetric effects to a certain range of distances for artistic control.

float stepsize = 0.1
float maxsteps = 100

This shader accumulates light and adjusts opacity by stepping through the volume along the view ray. The `stepsize` parameter gives the distance of each of these steps (in units of "common" space), *however* the total number of steps will never be more than `maxsteps`. Adjusting these parameters is the primary means by which you can trade off accuracy versus rendering speed.

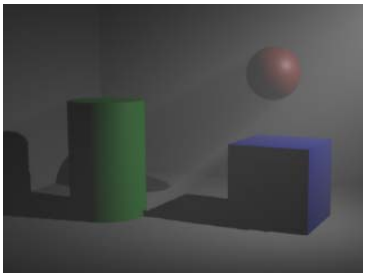



maxsteps = 100 maxsteps = 10
Adequate number of steps Inadequate - note banding in shadows

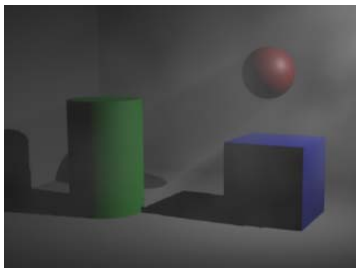
float smokeoctaves = 0
float smokevary = 1
float smokefreq = 1

These parameters control a noise field that makes the smoke appear non-uniform. If either `smokeoctaves` or `smokevary` are 0, there will be no apparent variation to the density of the smoke. If there is variation, then `smokeoctaves` is the number of octaves of noise that will be summed, `smokevary` is the

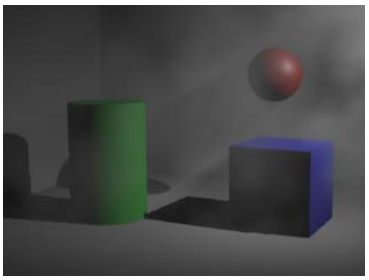
amount of variation (0 means uniform smoke, larger values make for more variable smoke densities), and `smokefreq` is the frequency of smoke variation (higher frequencies make the "lumps" closer together).



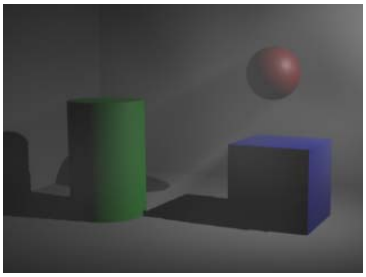
smokevary = 0 (smokeoctaves = 2, smokefreq = 0.5)



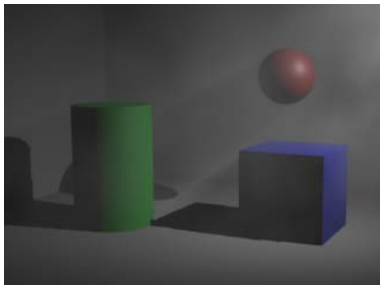
smokevary = 1 (smokeoctaves = 2, smokefreq = 0.5)



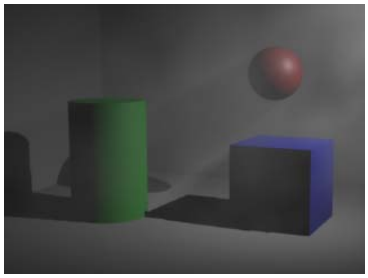
smokevary = 2 (smokeoctaves = 2, smokefreq = 0.5)



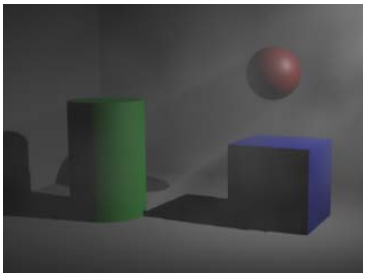
smokeoctaves = 1



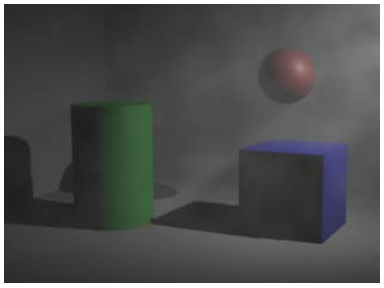
smokeoctaves = 3



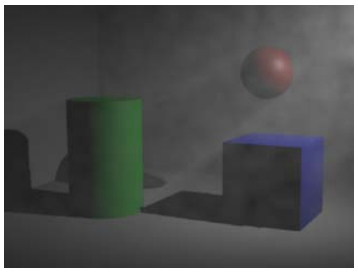
smokeoctaves = 5



smokefreq = 0.5



smokefreq = 1



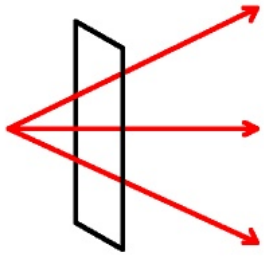
smokefreq = 2

Acknowledgements and References

The functionality of this shader is based on the smoke presented in Apodaca & Gritz, *Advanced RenderMan: Creating CGI for Motion Pictures*, Morgan-Kaufmann, 1999 (ISBN 1-55860-618-1).

Debugging Shaders

raygoggles.gsl



Description

This material colors itself simply by tracing a ray in the view direction.

The main use is to use this material on a "front plane" shader (an object that covers the camera view, in front of all other objects), so you can "see" the scene as it appears to the ray tracing engine. You may notice, for example, that the ray tracer has a more coarsely-tessellated view of the scene than you would ordinarily see through the camera, and of course the ray goggles will not see any objects that are not ray-traceable.

Here is an example of Pyg code to set up such a front plane object:

```
PushAttributes ()
Shader ("surface", "raygoggles", "string envname", "relection")
Input ("frontplane.pyg")
PopAttributes ()
```

Parameters

string envname = "reflection"

The name of the geometry set that will be visible in the view.

showdudv.gsl

Description

This debugging shader sets the color of the object so that the red and green channels hold the numerical values of the magnitudes of the derivatives *u* and *v* parameters, respectively. That is, `filterwidth(u)` and `filterwidth(v)`. The blue channel is always set to 0.1. Lights are not consulted or necessary.

This is occasionally useful if you need to help visualize the curvature of surfaces.

Hint: if you are using iv to display images made with this debugging shader, use the `R` and `G` hotkeys to show the u and v derivatives (i.e., red and green channels) individually.

Parameters

float uscale = 1
float vscale = 1
Scale the u and v derivatives when turning them into

uscale = 1, vscale = 1 (default)

uscale = 2, vscale = 1 (default)

showfacing.gsl

Description

This debugging shader colors front-facing surfaces green and back-facing surfaces red. Back-facing surfaces are those whose normals face away from the camera. This is very useful to find which surfaces have inconsistently-facing normals.

Parameters

In the examples below, the teapot spout and handle are backfacing, and the body and lid are frontfacing.

float Ka = 0.25
float Kd = 0.75

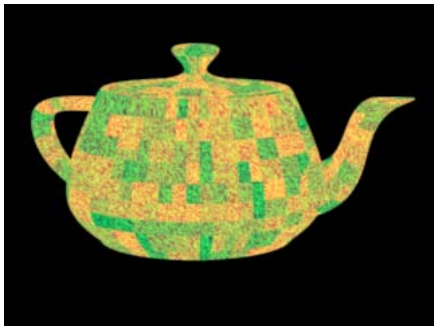
`Ka` and `Kd` allow you to modify the degree to which the appearance of the surface indicates relative orientation. When `Ka` is high and `Kd` is low, the appearance is "flat" merely coloring green or red depending on whether the surface is front- or back-facing. When `Ka` is low and `Kd` is high, there is a directional component similar to `defaultsurface.gsl` that allows you to also visualize more clearly the shape of the surface.


Ka = 0, Kd = 1


Ka = 0.25, Kd = 0.75 (default)


Ka = 1, Kd = 0

showgrids.gsl



Description

This debugging shader sets the color of the object so that the red channel has a different random color for each "grid" (collection of points shaded at once) and the green channel has a different random color for each vertex (point shaded). The blue channel is set to a constant value of 0.1 No lights are needed nor used.

This is helpful to visualize how the renderer splits and dices geometry.

Hint: if you are using `iv` to display images made with this debugging shader, use the `R` and `G` hotkeys to show the grids and points separately.

Parameters

None.

shownormals.gsl

Description

This debugging shader lets you visualize the surface normal, `N`.

Parameters

string space = ""

The normal `N` will be transformed into this named coordinate system before encoded into a color. The name of the coordinate system may be `"world"`, `"shader"`, or any other named coordinate system. If the empty string is passed (the default), `"common"` will be used.

float bias = 0

If bias is 0, the normals will be rescaled to the 0..1 range. That is, the color will be `0.5 + N/2`.

If bias is 1, normals will not be rescaled, and therefore may have negative values.

If bias is -1, normals will be negated (but not otherwise rescaled) before being encoded.

showst.gsl

Description

This debugging shader sets the color of the object so that the red and green channels hold the numerical values of the surface's *s* and *t* coordinates, respectively. The blue channel is set to a constant value of 0.1. Optionally, lights may or may not be used or required. If the surface is not given attached `"s"` and `"t"` coordinates, `u` and `v` will be used instead.

Hint: if you are using iv to display images made with this debugging shader, use the `R` and `G` hotkeys to show the s and t values (i.e., red and green channels) individually.

Parameters

float use_lighting = 0

If zero, the surface's color will be set so that the red and green channels hold the numerical values of the surface's *s* and *t* coordinates, respectively, and the blue channel is set to a constant value of 0.1. No lights are necessary, the surface will appear to "glow."

If `use_lighting` is nonzero, this color coding will simply be used as the base color for a plastic-like material and lights will be used as with normal shaders.

use_lighting = 0 (default)

use_lighting = 1

float Ka = 1
float Kd = 0.5
float Ks = 0.5
float roughness = 0.1
color specularcolor = color (1,1,1)

These parameters are identical to the their functionality in the `plastic` shader. These are only used if `use_lighting` is nonzero.

float desaturate_backfacing = 0

If nonzero, the color will be "desaturated" for backfacing geometry (those surfaces whose normals face away from the camera). This makes it easy to spot backfacing geometry.

In the examples below, the teapot spout and handle are backfacing:

desaturate_backfacing = 0 (default)

desaturate_backfacing = 1

desaturate_backfacing = 1, use_lighting = 1

showuv.gsl

Description

This debugging shader sets the color of the object so that the red and green channels hold the numerical values of the surface's *u* and *v* coordinates, respectively. The blue channel is set to a constant value of 0.1 No lights are needed nor used.

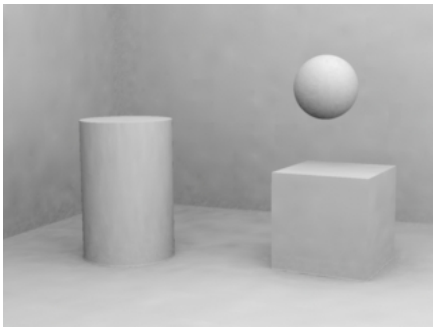
Hint: if you are using iv to display images made with this debugging shader, use the `R` and `G` hotkeys to show the u and v values (i.e., red and green channels) individually.

Parameters

None.

Special-Purpose Shaders

ambocclude.gsl



Description

A special-purpose surface shader that computes *ambient occlusion*, which is a measurement of how much of the hemisphere above each surface point is occluded by other objects in the scene. This shader colors points white if they have a completely unoccluded view of the hemisphere, and black if the view of the hemisphere is completely occluded. This color-coding is put in the surface color `C`, as well as output variable `unocc`. Additionally, the output variable `Nunocc` receives the average unoccluded direction.

The main use of this shader is for gathering the `unocc` and `Nunocc` outputs into spatial databases or NDC texture maps in an initial rendering pass. During a later beauty pass, these results may be used to scale light intensity in order to make a more realistic quality to the illumination.

The occlusion is computed by tracing rays distributed around the hemisphere above each shaded point. In order to reduce the expense of ray tracing, fully-sampled hemispheres are computed *sparsely*, that is, only every once in a while, and the results are smoothly interpolated. When full hemispheres are computed, their results are stored in a spatial database (SDB), which is then used for the sparse interpolation for points in which the full sampling is not performed.

There are several parameters that control the work per hemisphere and how frequently the hemispheres are computed. Many rays per hemisphere is more expensive than few rays per hemisphere, and more frequent full sampling of hemispheres is more expensive than very sparsely computed hemispheres.

Parameters

string occlusionname = ""

The name of the geometry set containing the objects that provide occlusion.

string sdbname = ""

The file name of the spatial database, in the case that the SDB is to be read from or written to disk.

string sdbmode = ""

The file mode for the spatial database. Valid choices are:

`""`

The SDB is kept in memory only. New samples will be computed and added to the in-memory SDB as required, but will not be saved to disk.

`"r"`

Before the first sample, the SDB will be read from disk (if a disk SDB file exists with that name). New samples will be computed and added to the in-memory SDB as required.

`"ro"`

Before the first sample, the SDB will be read from disk (if a disk SDB file exists with that name). No new samples will ever be computed -- existing samples will always be interpolated, even if it needs to ignore the `maxhitdist` value.

`"w"`

New samples will be computed and added to the in-memory SDB as required. After rendering is complete, the SDB will be written to disk.

`"rw"`

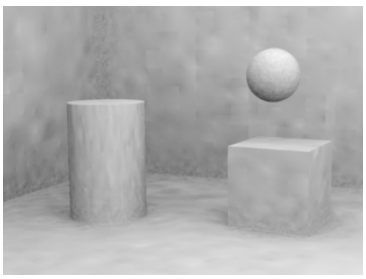
Both "r" and "w" options apply -- the SDB will be read from disk before rendering begins (if it exists), new samples will be added if necessary during rendering, and the final SDB will be written to disk after rendering is complete.

`"wo"`

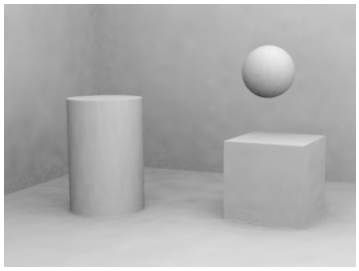
Computed hemisphere samples will be saved to disk, but never interpolated (thus forcing full hemisphere computation at every shading point).

float samples = 64

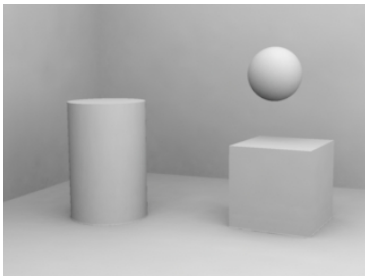
The maximum number of rays used to sample the hemisphere. Total rendering time increases as the number of samples increases, but more samples make for more accurate (less noisy) estimates of the ambient occlusion. The default value of 64 is fine for rapid previews, but for full production frames a value of 256 or higher is probably necessary to reduce the sampling noise.



samples = 64
(time: 0:06)



samples = 256
(time: 0:10)

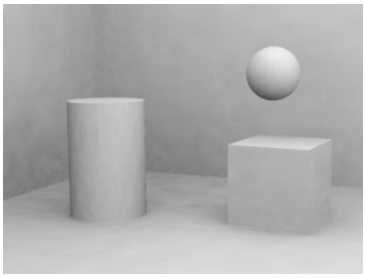


samples = 1024
(time: 0:29)

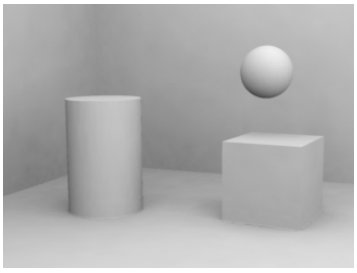
float adaptive = 1

When zero, the maximum number of rays will always be used when sampling a hemisphere. When nonzero (the default), fewer than the full number of samples will be used where possible. Larger values will result in more adaptive sampling (closer to zero will result in the full sampling being used most of the time, values larger than one will be more aggressive in its attempt to use fewer samples).

For most scenes, adaptive sampling makes only a minor difference in the image but can speed up rendering. But for some scenes, adaptive sampling can lead to unacceptable artifacts and in those cases you will need to set `adaptive = 0`.



adaptive = 1, samples = 256
(time: 0:10)



adaptive = 0, samples = 256
(time: 0:16)

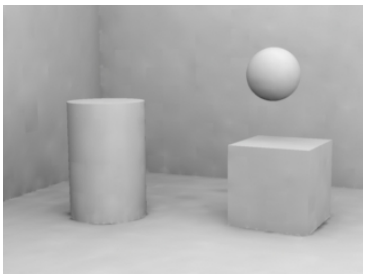
In this example, you can see that turning off adaptive sampling increases both rendering time and quality. It's up to you to decide how to make the tradeoff.

float maxerror = -1

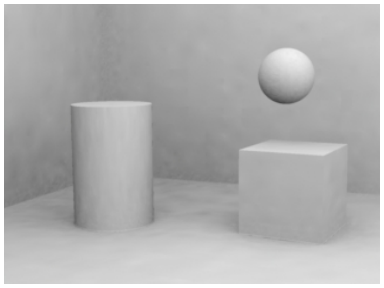
Gives the allowable error metric that in part determines when values already in the occlusion SDB can be interpolated, and when new hemispheres need to be sampled. Larger values indicate a higher allowable error -- less frequent full

hemisphere recomputation. Smaller values indicate a lower tolerance for error, and therefore more frequent full recomputation. A value of 0 will force a full hemisphere computation at every shading point, never reusing previous hemispheres already in the SDB (thus giving a maximal quality image, though at maximal expense).

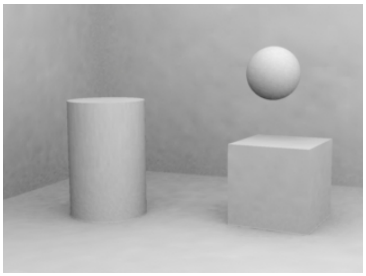
A value less than zero will use the value set by `Attribute("occlusion:maxerror")`, which defaults to 0.25. The default parameter value is -1, which means that unless this value is changed, the attribute will be used to supply the value.



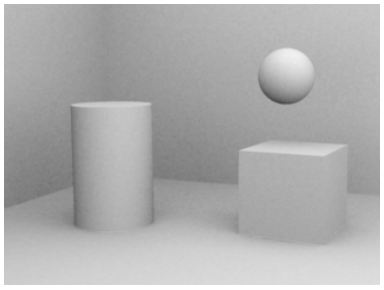
maxerror = 0.5
(time: 0:05)



maxerror = 0.1
(time: 0:10)



maxerror = 0.05
(time: 0:17)



maxerror = 0
(time: 1:07)

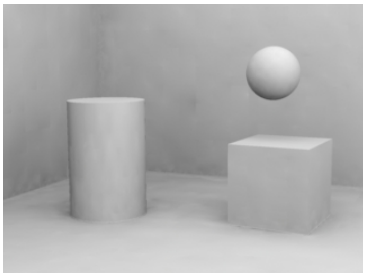
float maxpixeldist = -1

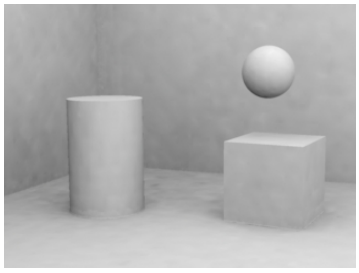
In addition to the `maxerror` metric, full hemisphere computation will be performed at any point that is farther away from the next nearest full hemisphere by this distance, expressed roughly in units of final image pixels. Thus, a value of `maxpixeldist` = 10 will compute fully sampled hemispheres no less frequently than about every 10x10 pixel block (although they are not aligned to pixels, and it may sample more frequently, depending on the error metrics).

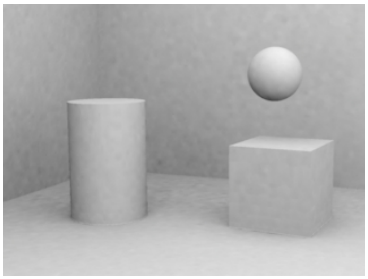
A value of `maxpixeldist` = 0 will force full recomputation of occlusion on every shading point (just as with `maxerror` = 0), achieving maximal quality at maximal expense.

A value less than zero will use the value set by `Attribute("occlusion:maxpixeldist")`, which defaults to 20. The

default parameter value is -1, which means that unless this value is changed, the attribute will be used to supply the value.


maxpixeldist = 30
(time: 0:06)

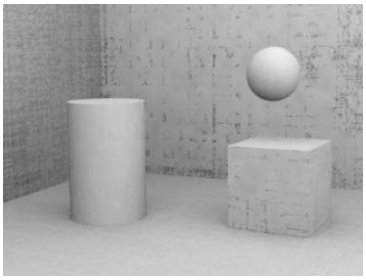

maxpixeldist = 10
(time: 0:07)

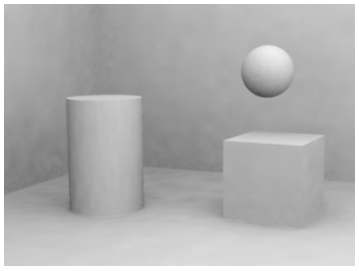

maxpixeldist = 5
(time: 0:10)

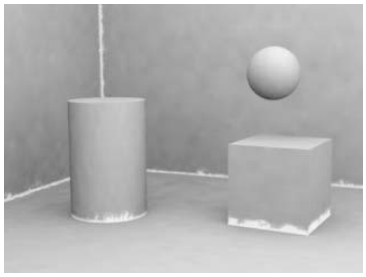
float bias = -1

As with shadows and ray tracing, objects closer than the bias distance are ignored when computing occlusion. This helps to eliminate self-intersection artifacts. But be careful not to use a bias too large, or you will see a different kind of artifact, particularly visible near corners and edges.

A value less than zero (the default is -1) will use the global `Attribute("shadow:bias")` as the bias amount.


bias = 0.000001
(self-intersection artifacts)


bias = 0.01
(just right)


bias = 0.1
(too large -- "misses")

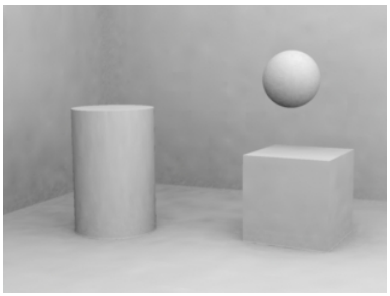
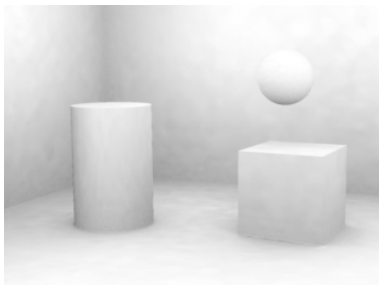
float maxhitdist = 1.0e6

Objects farther than this distance will be ignored when computing occlusion. This has the effect of making occlusion only count "local" (very close to the shading point) objects.

Reducing `maxhitdist` can be very helpful when computing ambient occlusion in *closed* environments. For example, in an interior room with no windows, all points will ordinarily be occluded (since no rays "escape" the scene). This is probably not helpful, but by adjusting `maxhitdist`, you can make the occlusion be a more local effect.

In the examples on this page, the scene is not actually enclosed -- the "ceiling"

and "back wall" are actually missing. Otherwise, the occlusion scenes would be completely black except for the images where `maxhitdist` is reduced.

maxhitdist = 1.0e6 (default) maxhitdist = 5

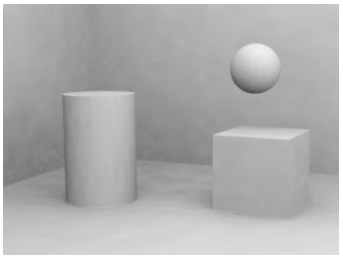
float falloff = 0
float falloffmode = 0

If `falloff` is 0 (the default), all ray hits are equally occluding. But if `falloff` is nonzero, the influence of occluders falls off with distance according to `falloffmode` (in both cases, `dist` is the distance to the closest object along the ray):

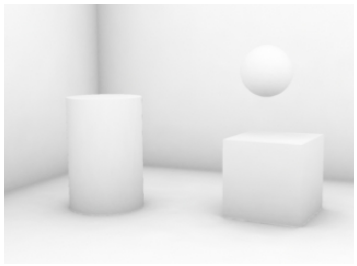
`falloffmode = 0`
Exponential falloff where the amount of occlusion is `exp(-falloff/dist)`
`falloffmode = 1`

Polynomial falloff that drops to 0 as distance approaches `maxhitdist`, where the amount of occlusion is `pow (1-dist/maxhitdist, falloff)`

The big difference between the two modes is that with exponential falloff, even very distant objects still contribute a little to the occlusion amount (though the occlusion diminishes rapidly with distance), though occlusion drops immediately to zero once distance is >= `maxhitdist`. In contrast, with polynomial falloff, occlusion smoothly drops to zero as distance approaches `maxhitdist`, with no abrupt cutoff (and also with a linear or polynomial, rather than exponential diminishing of occlusion).

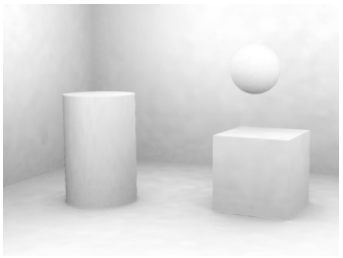


falloff = 0 (no falloff)
falloffmode = 0

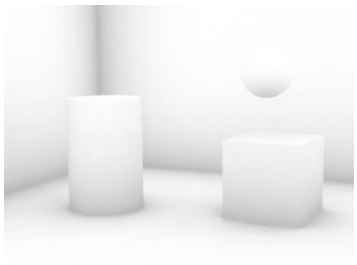


falloff = 0.25
falloffmode = 0

falloff = 1
falloffmode = 0



falloff = 0 (no falloff)
falloffmode = 1
maxhitdist = 5



falloff = 1
falloffmode = 1
maxhitdist = 5

falloff = 2
falloffmode = 1
maxhitdist = 5

float twosided = 1

If nonzero (the default), occlusion will be "gathered" from the side of the surface facing the camera. This is what you want for normal rendering.

If zero, ambient occlusion will only be computed on the side toward which the surface normal faces, regardless of its orientation with respect to the camera. This is what you want if you are "baking" ambient oclusion to 2D texture maps. But it obviously does not work if your normals are not consistently oriented.

Outputs

color C

The output color gets the unoccluded amount -- 0 for points for which their views are completely occluded, 1 for points for which their views are completely

unoccluded, and varying gray levels in between. This identical to `unocc`, except it's a color.

output float unocc

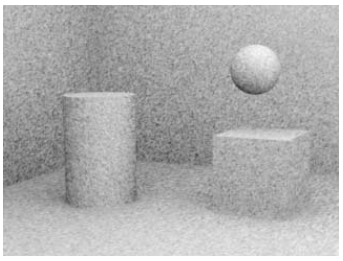
This output gets the unoccluded amount -- 0 for points for which their views are completely occluded, 1 for points for which their views are completely unoccluded, and varying gray levels in between. This identical to `C` except it's a float.

output normal Nunocc

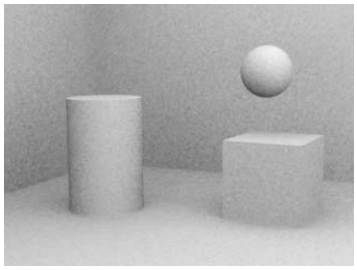
Gets the average unoccluded direction (some people call this the "bent normal"). The reason this is useful is that it may be used to look up from an environment map to determine light color for "image based lighting" (IBL), in which an environment map gives lighting that should be occluded by local objects. If `unocc` is the amount the scale the light (how much of the "background" is visible to the point), then `Nunocc` gives the direction to look up in the environment map.

Hints and Tips

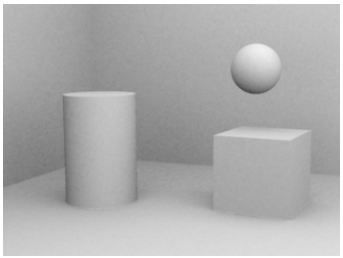
The sparse sampling and interpolation results in "blotchy" artifacts, as you can see in some of the examples above. The blotches are less noticeable as `maxerror` and/or `maxpixeldist` is reduced, and as `samples` is increased. Some people prefer sampling noise to blotches, and so leave `maxerror` = 0 but reduce samples in order to achieve good rendering times, accepting the noise. Below are some examples showing how appearance changes with `samples` when `maxerror` = 0:



maxerror = 0
samples = 16
(0:14)



maxerror = 0
samples = 64
(0:27)



maxerror = 0
samples = 256
(1:07)