interior {

The interior contains items which describe the properties of the interior of the object. This is in contrast to the texture which describes the surface properties only. The interior of an object is only of interest if it has a transparent texture which allows you to see inside the object. It also applies only to solid objects which have a well-defined inside/outside distinction. Note that the open keyword, or clipped_by modifier also allows you to see inside but interior features may not render properly. They should be avoided if accurate interiors are required.

Interior identifiers may be declared to make scene files more readable and to parameterize scenes so that changing a single declaration changes many values.

ior

Default : 1.0 = no refraction
Typical values : 0.0 -> 2.4 or higher
Example :

```
interior {
        ior 1.9 }
```

The index of refraction for air is 1.0, water is 1.33, glass is 1.5 and diamond is 2.4.

When light passes through a surface either into or out of a dense medium the path of the ray of light is bent. Such bending is called refraction.
The amount of bending or refracting of light depends upon the density of the material. Air, water, crystal and diamonds all have different densities and thus refract differently. The index of refraction or ior value is used by scientists to describe the relative density of substances. The ior keyword is used in POV-Ray in the interior to turn on refraction and to specify the ior value. For example:

object{ MyObject pigment{Clear} interior{ior 1.5}}

Normally transparent or semi-transparent surfaces in POV-Ray do not refract light. Earlier versions of POV-Ray required you to use the refraction keyword in the finish statement to turn on refraction. This is no longer necessary. Any non-zero ior value now turns refraction on.

In addition to turning refraction on or off, the old refraction keyword was followed by a float value from 0.0 to 1.0. Values in between 0.0 and 1.0 would darken the refracted light in ways that do not correspond to any physical property. Many POV-Ray scenes were created with intermediate refraction values before this bug was discovered so the feature has been maintained. A more appropriate way to reduce the brightness of refracted light is to change the filter or transmit value in the colors specified in the pigment statement or to use the fade_power and fade_distance keywords. See "Attenuation". Note also that neither the ior nor refraction keywords cause the object to be transparent.

Transparency only occurs if there is a non-zero filter or transmit value in the color.

The refraction and ior keywords were original specified in finish but are now properly specified in interior. They are accepted in finish for backward compatibility and generate a warning message.

media {

The media statement is used to specify particulate matter suspended in a medium such air or water. It can be used to specify smoke, haze, fog, gas, fire, dust etc. Previous versions of POV-Ray had two incompatible systems for generating such effects. One was halo for effects enclosed in a transparent or semi-transparent object. The other was atmosphere for effects that permeated the entire scene. This duplication of systems was complex and unnecessary. Both halo and atmosphere have been eliminated. See "Why are Interior and Media Necessary?" for further details on this change. See "Object Media" for details on how to use media with objects. See "Atmospheric Media" for details on using media for atmospheric effects outside of objects. This section and the sub-sections which follow explains the details of the various media options which are useful for either object media or atmospheric media.

Media works by sampling the density of particles at some specified number of points along the ray's path. Sub-samples are also taken until the results reach a specified confidence level. When used in an object's interior statement, sampling only occurs inside the object. When used for atmospheric media, the samples run from the camera location until the ray strikes an object. Therefore for localized effects, it is best to use an enclosing object even though the density pattern might only produce results in a small area whether the media was enclosed or not.

There are three types of particle interaction in media: absorbing, emitting, and scattering. All three activities may occur in a single media. Each of these three specifications requires a color. Only the red, green, and blue components of the color are used. The filter and transmit values are ignored. For this reason it is permissible to use one float value to specify an intensity of white color. For example the following two lines are legal and produce the same results:

emission 0.75

emission rgb<0.75,0.75,0.75>

The interior statement may contain one or more media statements. Media is used to simulate suspended particles such as smoke, haze, or dust. Or visible gasses such as steam or fire and explosions. When used with an object interior, the effect is constrained by the object's shape.

The calculations begin when the ray enters an object and ends when it leaves the object. This section only discusses media when used with object interior. The complete syntax and an explanation of all of the parameters and options for media is given in the section "Media".

Typically the object itself is given a fully transparent texture however media also works in partially transparent objects. The texture pattern itself does not effect the interior media except perhaps to create shadows on it. The texture pattern of an object applies only to the surface shell. Any interior media patterns are totally independent of the texture.

In previous versions of POV-Ray, this feature was called halo and was part of the texture specification along with pigment, normal, and finish. See "Why are Interior and Media Necessary?" for an explanation of the reasons for the change.

Note a strange design side-effect was discovered during testing and it was too difficult to fix. If the enclosing object uses transmit rather than filter for transparency, then the media casts no shadows. For example:

object{MyObject pigment{rgbt 1.0} interior{media{MyMedia}}} //no shadows

object{MyObject pigment{rgbf 1.0} interior{media{MyMedia}}} //shadows

Media may also be specified outside an object to simulate atmospheric media. There is no constraining object in this case. If you only want media effects in a particular area, you should use object media rather than only relying upon the media pattern. In general it will be faster and more accurate because it only calculates inside the constraining object. See "Atmospheric Media" for details on unconstrained uses of media.

You may specify more than one media statement per interior statement. In that case, all of the media participate and where they overlap, they add together.

Any object which is supposed to have media effects inside it, whether those effects are object media or atmospheric media, must have the hollow on keyword applied. Otherwise the media is blocked. See "Empty and Solid Objects" for details.

fade_distance

Default : 0.0
Typical values : any distance

fade_power

Default : 0.0
Typical values :1.0 -> 2.0 or higher
Example :

```
interior {
        ior 1.5
        fade_distance 10
        fade_power 2 }
```

Light attenuation is used to model the decrease in light intensity as the light travels through a transparent object. The keywords fade_power and fade_distance keywords are specified in the interior statement.

The fade_distance value determines the distance the light has to travel to reach half intensity while the fade_power value determines how fast the light will fall off. For realistic effects a fade power of 1 to 2 should be used. Default values for both keywords is 0.0 which turns this feature off.

The attenuation is calculated by a formula similar to that used for light source attenuation.

$$attenuation = \frac{1}{1 + \left(\frac{d}{fade\_distance} \right)^{fade\_power}}$$

The fade_power and fade_distance keywords were original specified in finish but are now properly specified in interior. They are accepted in finish for backward compatibility and generate a warning message.

caustics

Default : 0.0
Typical values : 0.0 -> 2.0
Example :
interior { caustics 1.5 }

Caustics are light effects that occur if light is reflected or refracted by specular reflective or refractive surfaces. Imagine a glass of water

standing on a table. If sunlight falls onto the glass you will see spots of light on the table. Some of the spots are caused by light being reflected by the glass while some of them are caused by light being refracted by the water in the glass.

Since it is a very difficult and time-consuming process to actually calculate those effects (though it is not impossible) POV-Ray uses a quite simple method to simulate caustics caused by refraction. The method calculates the angle between the incoming light ray and the surface normal. Where they are nearly parallel it makes the shadow brighter. Where the angle is greater, the effect is diminished. Unlike real-world caustics, the effect does not vary based on distance. This caustic effect is limited to areas that are shaded by the transparent object. You'll get no caustic effects from reflective surfaces nor in parts that are not shaded by the object.

The caustics Power keyword controls the effect. Values typically range from 0.0 to 1.0 or higher. Zero is the default which is no caustics. Low, non-zero values give broad hot-spots while higher values give tighter, smaller simulated focal points.

The caustics keyword was originally specified in finish but is now properly specified in interior. It is accepted in finish for backward compatibility and generates a warning message.

absorption

Default : rgb <0,0,0> which means no light is absorbed -- all light passes through normally.
Typical values : any color
Example :

```
interior {
        media {
                absorption rgb <0.4,0.9,0.7> } }
```

The absorption keyword specifies a color of light which is absorbed when looking through the media. For example absorption rgb<0,1,0> blocks the green light but permits red and blue to get through. Therefore a white object behind the media will appear magenta.

emission

Default : rgb <0,0,0> which means no light is emitted.
Typical values : any color
Example :

```
interior {
        media {
                emission 0.01 }
            }
        }
```

The emission keyword specifies a color of the light emitted from the particles. Although we say they "emit" light, this only means that they are visible without any illumination shining on them. They to not really emit light that is cast on to nearby objects. This is similar to an object with high ambient values. The default value is rgb<0,0,0>

scattering {

Exemple :
scattering { 1, rgb <0.4,0.7,0.1> }

The first float value specifies the type of scattering. This is followed by the color of the scattered light. The default value if no scattering
statement is given is rgb<0,0,0> which means no scattering occurs.

The scattering effect is only visible when light is shining on the media from a light source. This is similar to difuse reflection off of an object. In
addition to reflecting light, a scattering media also absorbs light like an absorption media. The balance between how much absorption occurs
for a given amount of scattering is controlled by the optional extinction keyword and a single float value. The default value of 1.0 gives an
extinction effect that matches the scattering. Values such as extinction 0.25 give 25% the normal amount. Using extinction 0.0 turns it off
completely. Any value other than the 1.0 default is contrary to the real physical model but decreasing extinction can give you more artistic
flexibility.

Type

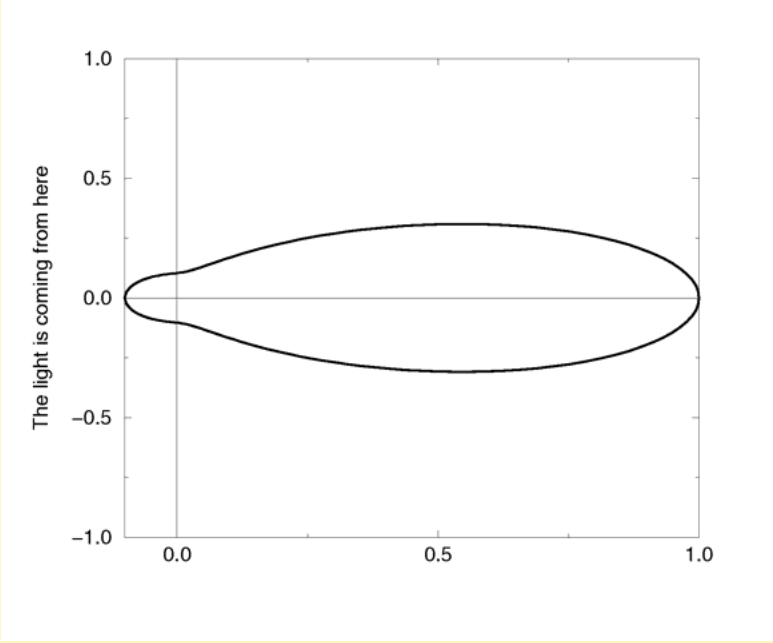
Valid values : 1 | 2 | 3 | 4 | 5
Example :

```
interior {
        media {
                scattering { 3, rgb 0.2 }
        }
```

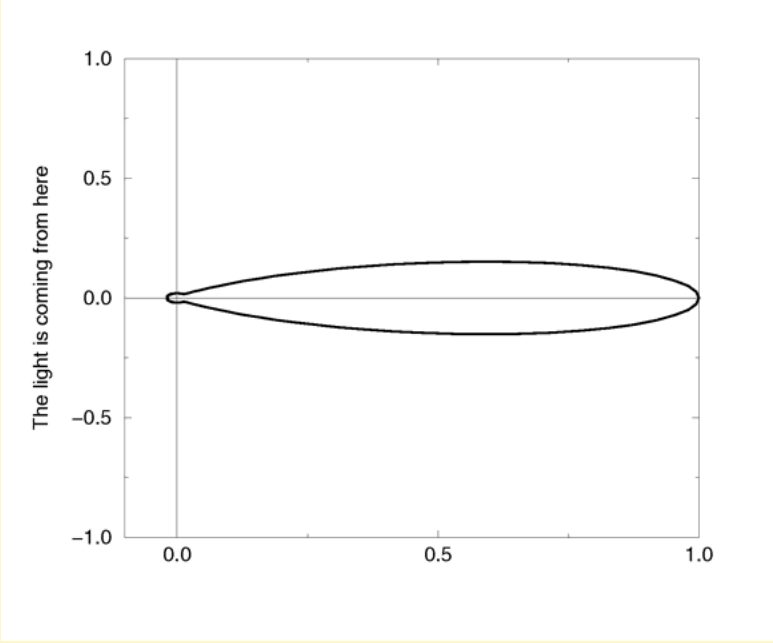
The integer value Type specifies one of five different scattering phase functions representing the different models: isotropic, Mie (haze and
murky atmosphere), Rayleigh, and Henyey-Greenstein.

Type 1, isotropic scattering is the simplest form of scattering because it is independent of direction. The amount of light scattered by particles
in the atmosphere does not depend on the angle between the viewing direction and the incoming light.

Types 2 and 3 are Mie haze and Mie murky scattering which are used for relatively small particles such as minuscule water droplets of fog,
cloud particles, and particles responsible for the polluted sky. In this model the scattering is extremely directional in the forward direction i.e. the
amount of scattered light is largest when the incident light is anti-parallel to the viewing direction (the light goes directly to the viewer). It is
smallest when the incident light is parallel to the viewing direction. The haze and murky atmosphere models differ in their scattering
characteristics. The murky model is much more directional than the haze model.

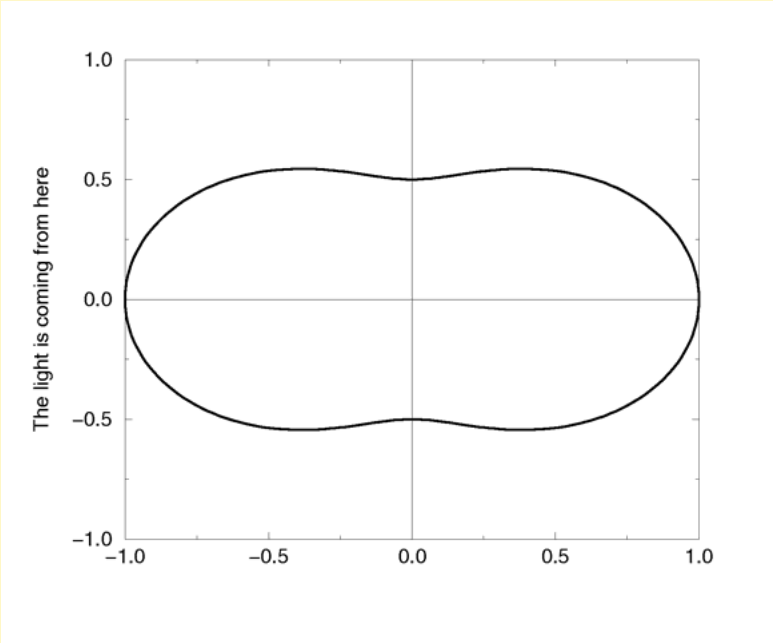


The Mie "haze" scattering function.



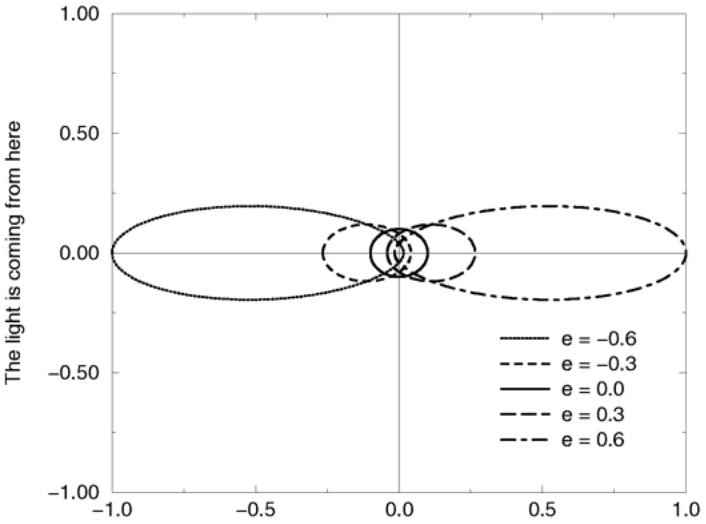
The Mie "murky" scattering function.

Type 3 Rayleigh scattering models the scattering for extremely small particles such as molecules of the air. The amount of scattered light
depends on the incident light angle. It is largest when the incident light is parallel or anti-parallel to the viewing direction and smallest when the
incident light is perpendicular to the viewing direction. You should note that the Rayleigh model used in POV-Ray does not take the dependency
of scattering on the wavelength into account.



The Rayleigh scattering function.

Type 5 is the Henyey-Greenstein scattering model. It is based on an analytical function and can be used to model a large variety of different
scattering types. The function models an ellipse with a given eccentricity e. This eccentricity is specified by the optional keyword eccentricity
which is only used for scattering type five. The default eccentricity value of zero defines isotropic scattering while positive values lead to
scattering in the direction of the light and negative values lead to scattering in the opposite direction of the light. Larger values of e (or smaller
values in the negative case) increase the directional property of the scattering.



The Heyney-Greenstein scattering function for different eccentricity values.

color

variance

Default : 1/128
Typical values : 1/128 -> 1/1000 The lower, the slower (and the more accurate, but always more than 0)
Example :

```
interior {
        media {
                emission 0.03
                variance 1/256
                }
        }
```

As each interval is sampled, the variance is computed. If the variance is below a threshold value, then no more samples are needed. The
variance and confidence keywords specify the permitted variance allowed and the confidence that you are within that variance. The exact
calculations are quite complex and involve chi-squared tests and other statistical principles too messy to describe here.

confidence

Default : 0.9
Typical values : 0.9 -> 0.9999999 The higher, the slower (and the more accurate, but no more than 1)
Example :

```
interior {
        media {
                emission 0.04
                confidence 0.99 } }
```

As each interval is sampled, the variance is computed. If the variance is below a threshold value, then no more samples are needed. The
variance and confidence keywords specify the permitted variance allowed and the confidence that you are within that variance. The exact
calculations are quite complex and involve chi-squared tests and other statistical principles too messy to describe here.

ratio

Default : 0.9

The ratio keyword distributes intervals differently between lit and unlit areas.
The default value of ratio 0.9 means that lit intervals get more samples than unlit intervals. Note that the total number of intervals must exceed
the number of illuminated intervals. If a ray passes in and out of 8 spotlights but you've only specified 5 intervals then an error occurs.

samples

Default : 1,1
Typical values : 1,10 -> 1,50 The higher, the slower (and the more accurate)
Example :

```
interior {
        media {
                absorption rgb 0.03
                samples 1,10 }
        }
```

The samples Min, Max keyword specifies the minimum and maximum number of samples taken per interval.

<pattern>

Valid values :
agate | average | boxed | bozo | brick | bumps | checker | crackle | cylindrical | density_File | dents | gradient | granite | hexagon | leopard | mandel | marble | onion | planar | quilted | radial | ripples | spherical | spiral1 | spiral2 | spotted | waves | wood | wrinkles
Example :

```
pigment {
        ripples
        color_map {
                [ 0 color rgb <0.1,0.1,0.3> ]
                [ 1 color rgb <0.3,0.7,0.4> ]
                  }
    }
```

POV-Ray uses a method called three-dimensional solid texturing to define the color, bumpiness and other properties of an object. You specify
the way that the texture varies over a surface by specifying a pattern. Patterns are used in pigments, normals and texture maps as well as
media density.

All patterns in POV-Ray are three dimensional. For every point in space, each pattern has a unique value. Patterns do not wrap around a
surface like putting wallpaper on an object. The patterns exist in 3d and the objects are carved from them like carving an object from a solid
block of wood or stone.

Consider a block of wood. It contains light and dark bands that are concentric cylinders being the growth rings of the wood. On the end of the
block you see these concentric circles. Along its length you see lines that are the veins. However the pattern exists throughout the entire block. If
you cut or carve the wood it reveals the pattern inside. Similarly an onion consists of concentric spheres that are visible only when you slice it.
Marble stone consists of wavy layers of colored sediments that harden into rock.

These solid patterns can be simulated using mathematical functions. Other random patterns such as granite or bumps and dents can be
generated using a random number system and a noise function.

In each case, the x, y, z coordinate of a point on a surface is used to compute some mathematical function that returns a float value. When used
with color maps or pigment maps, that value looks up the color of the pigment to be used. In normal statements the pattern function result
modifies or perturbs the surface normal vector to give a bumpy appearance. Used with a texture map, the function result determines which
combinations of entire textures to be used. When used with media density it specifies the density of the particles or gasses.

The following sections describe each pattern. See the sections "Pigment", "Normal", "Patterned Textures" and "Density" for more details on
how to use patterns. Unless mentioned otherwise, all patterns use the ramp_wave wave type by default but may use any wave type and may be
used with color_map, pigment_map, normal_map, slope_map, texture_map, density, and density_map.

density {

Example :

```
interior {
        media {
                scattering { 1, rgb 0.5 }
                density {
                        bumps
                        color_map {
                                [ 0 color rgb <1,0,0> ]
                                [ 1 color rgb <0,1,0> ]
                                }
                        }
                }
          }
```

Particles of media are normally distributed in constant density throughout the media. However the density statement allows you to vary the
density across space using any of POV-Ray's pattern functions such as those used in textures. If no density statement is given then the density
remains a constant value of 1.0 throughout the media. More than one density may be specified per media statement.

The density statement may begin with an optional density identifier. All subsequent values modify the defaults or the values in the identifier. The
next item is a pattern type. This is any one of POV-Ray's pattern functions such as bozo, wood, gradient, waves, etc. Of particular usefulness are
the spherical, planar, cylindrical, and boxed patterns which were previously available only for use with our discontinued halo feature. All
patterns return a value from 0.0 to 1.0. This value is interpreted as the density of the media at that particular point. See "Patterns" for details on
particular pattern types. Although a solid COLOR pattern is legal, in general it is used only when the density statement is inside a density_map.

<waveform>

Valid values :
ramp_wave | triangle_wave | sine_wave | scallop_wave | cubic_wave | poly_wave
Example :

```
pigment {
        spiral 1 5
        sine_wave
        color_map {
                [ 0.0 color rgbf 1 ]
                [ 0.2 color rgbf 1 ]
                [ 1.0 color rgb <1,1,0.5> ]
                }
        }
```

POV-Ray allows you to apply various wave forms to the pattern function before applying it to a blend map. Blend maps are color_map,
pigment_map, normal_map, slope_map, density_map, and texture_map.

Most of the patterns which use a blend map, use the entries in the map in order from 0.0 to 1.0. The effect can most easily be seen when these
patterns are used as normal patterns with no maps. Patterns such as gradient or onion generate a grove or slot that looks like a ramp that
drops off sharply. This is called a ramp_wave wave type and it is the default wave type for most patterns. However the wood and marble patterns
use the map from 0.0 to 1.0 and then reverses it and runs it from 1.0 to 0.0. The result is a wave form which slopes upwards to a peak, then
slopes down again in a triangle_wave. In earlier versions of POV-Ray there was no way to change the wave types. You could simulate a
triangle wave on a ramp wave pattern by duplicating the map entries in reverse, however there was no way to use a ramp wave on wood or
marble.

Now any pattern that takes a map can have the default wave type overridden. For example:

```
pigment { wood color_map { MyMap } ramp_wave }
```

Also available are sine_wave, scallop_wave, cubic_wave and poly_wave types. These types are of most use in normal patterns as a type of
built-in slope map. The sine_wave takes the zig-zag of a ramp wave and turns it into a gentle rolling wave with smooth transitions. The
scallop_wave uses the absolute value of the sine wave which looks like corduroy when scaled small or like a stack of cylinders when scaled
larger. The cubic_wave is a gentle cubic curve from 0.0 to 1.0 with zero slope at the start and end. The poly_wave is an exponential function. It is
followed by an optional float value which specifies exponent. For example poly_wave 2 starts low and climbs rapidly at the end while poly_wave
0.5 climbs rapidly at first and levels off at the end. If no float value is specified, the default is 1.0 which produces a linear function identical to
ramp_wave.

Although any of these wave types can be used for pigments, normals, textures, or density the effect of many of the wave types are not as
noticeable on pigments, textures, or density as they are for normals.

Wave type modifiers have no effect on block patterns checker, brick, and hexagon nor do they effect image_map, bump_map or material_map.
They also have no effect in normal statements when used with bumps, dents, quilted, ripples, waves, or wrinkles because these normal
patterns cannot use normal_map or slope_map.

turbulence

Default : 0.0
Typical values : 0.0 -> 2.0
Example :

```
pigment {
        gradient z
        turbulence 0.4
        lamda 1.8
        octaves 7
        omega 0.8
        color_map {
                [ 0 color rgb <0.9,0.4,0.1> ]
                [ 1 color rgb <0.1,0.1,0.1> ]
                  }
    }
```

The keyword turbulence followed by a float or vector may be used to stir up any pigment, normal, texture, irid or density. A number of
optional parameters may be used with turbulence to control how it is computed.

Typical turbulence values range from the default 0.0, which is no turbulence, to 1.0 or more, which is very turbulent. If a vector is specified
different amounts of turbulence are applied in the x-, y- and z-direction. For example

```
turbulence <1.0, 0.6, 0.1>
```

has much turbulence in the x-direction, a moderate amount in the y-direction and a small amount in the z-direction.

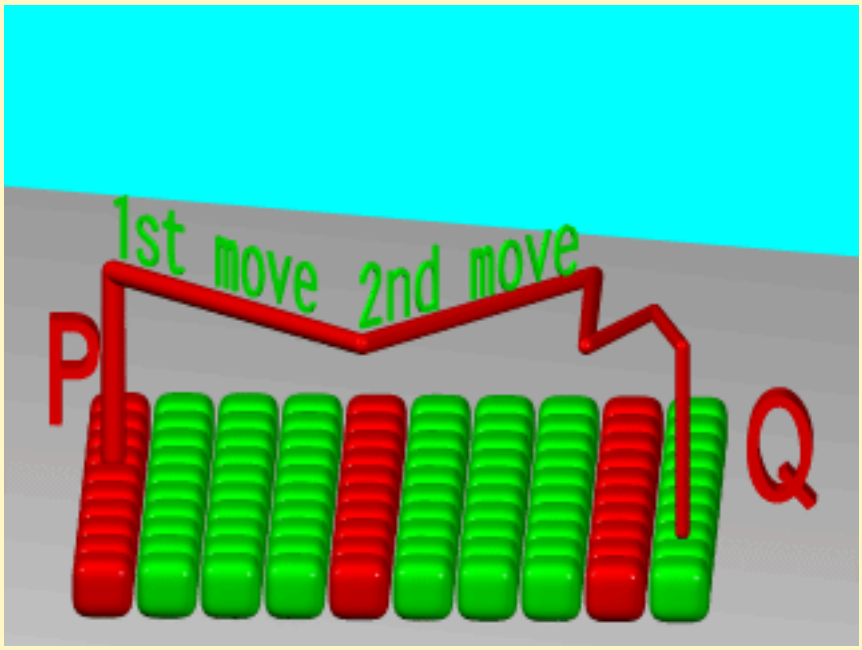
Turbulence uses a random noise function called DNoise. This is similar to the noise used in the bozo pattern except that instead of giving a
single value it gives a direction. You can think of it as the direction that the wind is blowing at that spot. Points close together generate almost
the same value but points far apart are randomly different.

In general the order of turbulence parameters relative to other pattern modifiers such as transformations, color maps and other maps is not
important. For example scaling before or after turbulence makes no difference. The turbulence is done first, then the scaling regardless of
which is specified first. See section "" for a way to work around this behavior.

In general, the order of turbulence parameters relative to each other and to other pattern modifiers such as transformations or color_map and
other maps is not important. For example scaling before or after turbulence makes no difference. The turbulence is done first, then the scaling
regardless of which is specified first. However the order in which transformations are performed relative to warp statements is important. You
can also specify turbulence inside warp and in this way you can force turbulence to be applied after transformations. See "Warps" for details.

Turbulence uses DNoise to push a point around in several steps called octaves. We locate the point we want to evaluate, then push it around a
bit using turbulence to get to a different point then look up the color or pattern of the new point.

It says in effect "Don't give me the color at this spot... take a few random steps in different directions and give me that color". Each step is
typically half as long as the one before.



Turbulence random walk.

The magnitude of these steps is controlled by the turbulence value. There are three additional parameters which control how turbulence is
computed. They are octaves, lambda and omega. Each is optional. Each is followed by a single float value. Each has no effect when there is no
turbulence.

lambda

Default : 2
Typical values : 1 -> 4
Example :

```
pigment {
        gradient z
        turbulence 0.4
        lamda 1.8
        octaves 7
        omega 0.8
        color_map {
                [ 0 color rgb <0.9,0.4,0.1> ]
                [ 1 color rgb <0.1,0.1,0.1> ]
                  }
    }
```

The lambda parameter controls how statistically different the random move of an octave is compared to its previous octave. The default value is
2.0 which is quite random. Values close to lambda 1.0 will straighten out the randomness of the path in the diagram above. The zig-zag steps
in the calculation are in nearly the same direction. Higher values can look more swirly under some circumstances.

octaves

Default : 6
Valid values : 1 -> 10
Example :

```
pigment {
        gradient z
        turbulence 0.4
        lamda 1.8
        octaves 7
        omega 0.8
        color_map {
                [ 0 color rgb <0.9,0.4,0.1> ]
                [ 1 color rgb <0.1,0.1,0.1> ]
                  }
    }
```

The octaves keyword may be followed by an integer value to control the number of steps of turbulence that are computed. The default value of 6 is a fairly high value; you won't see much change by setting it to a higher value because the extra steps are
too small. Float values are truncated to integer. Smaller numbers of octaves give a gentler, wavy turbulence and computes faster. Higher
octaves create more jagged or fuzzy turbulence and takes longer to compute.

omega

Default : 0.5
Typical values : 0.1 -> 3
Example :

```
pigment {
        gradient z
        turbulence 0.4
        lamda 1.8
        octaves 7
        omega 0.8
        color_map {
                [ 0 color rgb <0.9,0.4,0.1> ]
                [ 1 color rgb <0.1,0.1,0.1> ]
                  }
    }
```

The omega value controls how large each successive octave step is compared to the previous value. Each successive octave of turbulence is
multiplied by the omega value. The default omega 0.5 means that each octave is 1/2 the size of the previous one. Higher omega values mean
that 2nd, 3rd, 4th and up octaves contribute more turbulence giving a sharper, crinkly look while smaller omegas give a fuzzy kind of turbulence
that gets blurry in places.

color_map

Density with color_map

Typically a media uses just one constant color throughout. Even if you vary the density, it is usually just one color which is specified by the
absorption, emission, or scattering keywords. However when using emission to simulate fire or explosions, the center of the flame (high
density area) is typically brighter and white or yellow. The outer edge of the flame (less density) fades to orange, red, or in some cases deep
blue. To model the density-dependent change in color which is visible, you may specify a color_map. The pattern function returns a value from
0.0 to 1.0 and the value is passed to the color map to compute what color or blend of colors is used. See "Color Maps" for details on how
pattern values work with color_map. This resulting color is multiplied by the absorption, emission and scattering color. Currently there is no
way to specify different color maps for each media type within the same media statement.

Consider this example:

```
media{
  emission 0.75
  scattering {1, 0.5}
  density { spherical
    color_map{
      [0.0 rgb <0,0,0.5>]
      [0.5 rgb <0.8, 0.8, 0.4>]
      [1.0 rgb <1,1,1>]
    }
  }
}
```

The color map ranges from white at density 1.0 to bright yellow at density 0.5 to deep blue at density 0. Assume we sample a point at density
0.5. The emission is 0.75*<0.8,0.8,0.4> or <0.6,0.6,0.3>. Similarly the scattering color is 0.5*<0.8,0.8,0.4> or <0.4,0.4,0.2>.

For block pattern types checker, hexagon, and brick you may specify a color list such as this:

```
density{checker rgb<1,0,0>, rgb<0,0,0>}
```

density_map {

Example :

```
density {
      bozo
      turbulence .15
      sine_wave
      density_map {
            [0 rgb <0.125 ,0.50 ,0.33>]   (here VALUE=0 , DENSITY_BODY = rgb <0.125 ,0.50 ,0.33> )
            [1 rgb <0.250 ,0.67 ,0.25>]   (here VALUE=1 , DENSITY_BODY = rgb <0.250 ,0.67 ,0.25> )
            }
       }
```

In addition to specifying blended colors with a color map you may create a blend of densities using a density_map. The syntax for a density
map is identical to a color map except you specify a density in each map entry (and not a color).

The VALUE is a float value between 0.0 and 1.0 inclusive and each DENSITY_BODY is anything which can be inside a density{...}
statement. The density keyword and {} braces need not be specified.

Note that the [] brackets are part of the actual DENSITY_MAP_ENTRY. They are not notational symbols denoting optional parts. The
brackets surround each entry in the density map. There may be from 2 to 256 entries in the map.

Density maps may be nested to any level of complexity you desire. The densities in a map may have color maps or density maps or any type of
density you want.

Entire densities may also be used with the block patterns such as checker, hexagon and brick. For example...

```
 density {
   checker
   density { Flame scale .8 }
   density { Fire scale .5 }
 }
```

Note that in the case of block patterns the density wrapping is required around the density information.

A density map is also used with the average density type. See "Average" for details.

You may declare and use density map identifiers but the only way to declare a density block pattern list is to declare a density identifier for the
entire density.