Clipka’s POV-Ray Voodoo
How Radiosity Really Works

By Clipka
(povray.text.tutorials 31-12-2008)

I guess it's time to sum up a few things I have learned both during experimenting with POV-Ray radiosity, and digging through (and tampering with) the source code. So here comes my "black magic" tome. I fear it's still as flawed as the 3.6 radiosity code was (I confess I didn't bother to read it again after hacking it in), but it may give you some insight already:

1. How Radiosity Really Works

The Basics:

Radiosity is intended to replace POV-Ray's classic material-based approach of modelling ambient illumination with a more realistic geometry-based model, to take into account diffusely reflected (or possibly emitted) light from other objects, giving a non-uniform ambient illumination.

Although it is correct to speak of ambient illumination here, I will avoid this term to avoid its ambiguity in the POV-Ray radiosity world, and instead speak of "indirect illumination".

The very basic concept is that in order to get the indirect illumination for a certain point, you would shoot a vast number of rays in virtually every direction, checking how much light would come from whatever object happens to be hit.

As a straightforward implementation of this concept would lead to incredibly long render times, it is of course not feasible without optimizations. Furthermore, there are additional details to consider.

Radiosity Samples:

The main optimization is to not compute the indirect illumination for every point encountered, but to take only representative "samples", and to determine the actual illumination for any specific point by interpolating between nearby samples. This can be done because in the real world this indirect illumination typically changes only very gradually.

In POV-Ray terms, this interpolation is called "re-use" of samples.

Re-Use:

Whether a sample can be re-used for a certain point in question depends on a variety of geometric factors, the main of which are:

* The overall distance to nearby geometric features: This comes as a by-product of the shooting of rays for the sample, and is simply an average of the distances the rays have travelled before hitting an object. A "harmonic mean" is used (the inverse of the arithmetic mean of the inverses) so that nearby objects have a higher influence on this value.

* The direction and distance to the nearest object: This comes as a by-product of the shooting of rays as well. A sample is deemed less re-usable for points in or near this direction.

* The curvature of the object: This is measured by the difference in the normal vectors of the sample and the point in question.

From all these factors, a weight is computed for the sample, and all samples interpolated according to their weight.

(In practice of course, there are several mechanisms in place to quickly filter out samples that cannot possibly have a nonzero weight for a given point.)

Test Ray Pattern:
Shooting rays in virtually every direction is, of course, virtually impossible. Therefore, only a limited number of rays can be shot, and their directions should be chosen carefully. The algorithm to choose the number of rays should ideally satisfy the following criteria:

* The algorithm should be fast. Doing things the stupid way but fast may actually be cleverer than spending time on finding the best way.

* The rays should be biased towards the normal, to accommodate for the fact that light coming in at shallower angles contributes less to the overall brightness of a surface. This could be modelled by multiplying the brightness of each ray with a corresponding correctional term, but modelling it instead by biasing the distribution of the rays is much more elegant, because it can do without the correctional term (which would be a trigonometric function and therefore quite expensive in terms of computing time), and also does not waste time on shooting many rays in directions that contribute few to the total brightness.

* Aside from this bias, the rays should be distributed very uniformly; first to not waste computing time on rays that happen to be almost identical and therefore are likely to contribute the same brightness and colour; and second to prevent parts of the scene to have a higher impact on the computed brightness (because they happen to receive more rays) than others.

* The algorithm should provide a way to control the total numbers of rays shot.

* The algorithm should be non-random, to be suitable for animations.

(For certain reasons it would be desirable to have a certain random element in the distribution of rays; however, this typically collides with multiple of the above criteria and is therefore ignored in POV-Ray.)

POV-Ray uses a precomputed table of directions specially designed to meet all of the above. However, the maximum number of rays shot is limited by the size of the table, which contains "only" 1600 entries.

Light Sources:
Radiosity significantly changes how lighting works, and allows for objects themselves to act as light sources.

POV-Ray combines radiosity with its classic lighting model. This allows for beginners to first experiment with it by just "pimping up" classically-lit scenes with radiosity a bit. However, radiosity can also be used stand-alone (although this sacrifices soft highlights unless one is familiar with other advanced techniques), by making objects themselves emit light. Details how to do this are mentioned later.

Recursions:

When POV-Ray starts taking radiosity samples, there is a hen-and-egg problem: To determine the indirect illumination at a sample point, it needs to shoot rays and determine how much light comes from that direction - which in turn may depend on the indirect illumination there, and due to a current lack of (completed) samples cannot be computed right now.

POV-Ray solves this problem in a pragmatic way, by doing a limited-depth recursion, keeping track of separate sets of samples for each level of depth.

The levels of depth can be seen as some kind of "generations" (although the individual generations are not computed in sequential order, but rather as needed):

When shooting rays for a first-generation sample, the world is assumed do be devoid of any indirect illumination, and any object hit is assumed to be illuminated only from classic lighting, maybe reflect or refract, and maybe emit light itself.

When shooting rays for the second-generation sample, the world is assumed to be indirectly illuminated as described by the first-generation samples.

When shooting rays for the third-generation sample, the world is assumed to be indirectly illuminated as described by the second-generation samples - and so on, until the desired number of generations has been reached.

(Note that these "generations" are *not* coincidental with radiosity pre-trace passes.)

Pre-Trace:

The intention of the radiosity sampling pre-trace is to take all samples that the final render may ever need, to avoid any new samples to be taken during the final render, as this may cause visible "jumps" in colour or brightness between areas rendered before and after the new sample was taken.

To make sure the collection of samples is dense enough, by default the pre-trace actually takes more samples than it would normally deem necessary. This is achieved by running multiple passes, and collecting additional samples on each pass until it finds a certain minimum number of samples "within reach" of each spot it examines.

To make sure that the distribution of samples is comparatively homogenous, the pre-trace starts with a very coarse pass, and gradually increases in resolution. In addition, the pre-sampling uses a good deal of jitter to avoid any clear patterns in the sampling distribution.

The pre-trace may not always be successful at achieving these goals, so sometimes taking new samples during the final trace cannot be totally avoided.

2. The Parameters

material ambient and global ambient_light:
With radiosity used to compute indirect illumination, POV-Ray's classic ambient mechanism is basically obsolete. The mathematical principles behind it, however, are exactly what is needed to simulate objects that emit light, allowing them to act as light sources for radiosity.

In scenes with conventional lighting, merely "spiced up" with radiosity for indirect lighting, you will usually want to turn off the ambient mechanism, by either specifying a global ambient_light value of 0, or setting all materials' ambient value to 0. Or both, which is probably the best idea: Setting ambient_light to 0 ensures that no material will emit light whatsoever without having to update all the material library you may be using, while defining your own materials with an ambient of 0 ensures that you can later use them again in radiosity-only scenes.

In radiosity-only scenes, you need some objects to emit light, so your global ambient_light must be nonzero (unless the only light source you will be using is a sky sphere). For ease of use, a value of 1 is ideal, so that you can directly control an object's emissive brightness by its material's ambient value.

Sometimes you may want to use materials from a library which were defined with a nonzero ambient value for conventional lighting. In this case, you may resort to using a very small global ambient_light (e.g. 0.001), and turning up your light- emitting objects' ambient value accordingly (multiply by 1000 in this case), although I recommend to avoid such quirks. Note also that you cannot assign an ambient value to a sky_sphere.

global max_trace_level:
In radiosity-only scenes, you want to set this a good deal higher than normal - Otherwise you *will* end up with stray dark splotches, that can't be really cured by any other means.

adc_bailout:
This parameter specifies an alternate ADC bailout value to use when tracing a radiosity sample ray.

In POV 3.6 and earlier, and 3.7 beta versions up to .29, contrary to documentation this value was actually implemented as a factor to multiply the standard value with; e.g. if your global adc_bailout was set to 0.01 and your radiosity adc_bailout was set to 0.02, you would have an effective ADC bailout value of 0.0002 during radiosity tracing, rendering this mechanism mostly useless.

always_sample:
When set to "off", this effectively sets "nearest_count" to 1 during the final render, so that new radiosity samples are only taken if they are absolutely needed.

For quick test renders, you may want to set this parameter to "on" to get away with a faster pretrace (or none at all).

Otherwise, you will definitely want to set this parameter to "off", as new samples taken during final trace may cause visible "jumps" in colour or brightness between areas rendered before and after the new sample was taken.

(Note that I recommend to set "nearest_count" to 1 anyway, which leaves this option without any effect.)

count:
This parameter is fairly straightforward: It specifies how many rays are shot for every radiosity sample.

If your scene is quite homogenous regarding colour and brightness, you will probably want to set this parameter fairly low to get a fast render.

If your scene has some comparatively small features of distinct colour or brightness, make sure you set this parameter fairly high, otherwise those features will be totally missed for some radiosity samples, while having too much impact on samples if they happen to be hit, resulting in visible splotches. As a rule of thumb, set this value high enough so that every prominently colored feature will typically be hit by multiple rays. Note however that having a lot of such objects of similar colour allows you to reduce the count again.

If you use a High Dynamic Range light probe for illumination with very small areas contributing very much of the total brightness, make sure to crank the count parameter up to maximum, and/or use a heavily blurred version of the probe. I recommend the latter for speed considerations, but it requires either that your light probe itself is not visible in the background or in reflections, that you use strong focal blur, or that you use a version of POV-Ray that supports the no_radiosity flag so that you can set up one sky sphere for radiosity and another for reflections and background. Using a radiosity pre- render with a blurrier sky texture may be a fallback option, but since the final render may need to take additional samples despite best efforts, I recommend some other approach if possible.

low_error_factor:
This parameter determines the relation between the density of samples taken during pretrace, and the density of samples required for the final render. More precisely, it specifies a scaling factor for the distance at which a sample is still considered re-usable.

You may normally want to set this parameter to a value somewhere near 0.5, which ensures that even if the pretrace missed some points, these will still be covered by nearby samples instead of requiring additional ones being taken during the final render.

If your scene is radiosity-lit with small light sources, and the maximum of 1600 sample rays turns out to be still insufficient, you may additionally want to reduce low_error_factor. The effect of reducing low_error_factor by a certain factor is somewhat similar to further increasing the count by the square of that factor, except that it may give a slightly better quality at slightly lower performance, and may require a higher-resolution pretrace.

Note that nearest_count does something similar, but not as good.

minimum_reuse:
The mechanism behind this parameter sacrifices quality to gain speed, by enforcing a certain minimum distance between radiosity samples, depending on how far away (and therefore how small in the final image) they are. The minimum_reuse specifies the ratio between the minimum distance of two radiosity samples and the camera distance.

If your scene contains only large structures, you will probably want to set this parameter fairly high to get a fast render.

However, if your scene contains lots of small details and you rely solely on radiosity for lighting, make sure you set this parameter low enough, using the following formula, where detail_size is the size of your smallest detail on screen:

minimum_reuse = 0.5 * (detail_size / image_width) * tan (camera_angle/2)

Aside from improving render speed (including getting away with a lower- resolution pretrace), there is no benefit of setting this parameter to a high value.

nearest_count:
This parameter specifies how many samples must be "within reach" to re-use them. However, this does not place any upper limit on the number of samples actually re-used.

Note that you normally want this effectively turned off during the final render to prevent additional samples from being taken, by specifying "always_sample off".

I recommend always setting this parameter to 1, as a higher sample density can be achieved just as well by lowering low_error_factor, which has the benefit of enforcing a more homogenous sample distribution, while nearest_count may allow samples to "flock", thereby somewhat reducing the benefit.

Note that increasing nearest_count by a certain factor has the same effect on overall sample density as dividing low_error_factor by the root of that factor.

(It seems that previous versions of POV-Ray used this value to place an *upper* limit on the number of samples re-used for a certain point, which may explain the existence of this otherwise obsolete parameter.)

pretrace_start & pretrace_end:
The pretrace_end value should be low enough to allow all necessary samples to be taken during pretrace already. I recommend using 1/min(image_width,image_height) if you are patient enough, which causes the last pretrace to be taken at the same resolution as the main render; otherwise set it according to the following formula:

pretrace_end = 0.5 * minimum_reuse / tan (camera_angle/2)

If you choose a high nearest_count to work around the 1600 sample ray limit, make sure to further reduce this value according to the following formula:

pretrace_end = 0.5 * minimum_reuse / (tan (camera_angle/2) * sqrt (nearest_count))

This makes sure that you get the full number of samples for every pixel.

For pretrace_start, I recommend using some power-of-2-fold of the value you use for pretrace_end, especially if you intend to go for a full-resolution last pretrace. Basically, any power of 2 will do, although it should not be too low, especially if you have chosen a high nearest_count. I suggest going for dramatically high values, as this will ensure the most homogenous distribution of samples.

recursion_limit:
This parameter specifies how many "generations" of samples will be taken.

For rather open scenes, a value of 1 should usually suffice. If you have many dark edges and hollows though, you may want to set this to 2 or even 3 to shed light into all corners.

Note however that this may also increase the risk of light "leaking" through thin walls, especially at corners.

3.A Troubleshooting

smeared shadows:
horizontally smeared shadows are the results of samples being taken during the final pass. This typically indicates that your pretrace is too coarse. Make sure you have set "always_sample" to "off", and check your pretrace_end. You will probably want to choose a smaller value (i.e. higher resolution), or increase minimum_reuse.

Alternatively, you can render the scene twice, using the save_data keyword on the first and load_data on the second run. Make sure to use exactly the same settings, or you may end up with yet more samples to be taken during the second run.

smooth meshes & triangles:
Radiosity is problematic with meshes and smooth triangles, as these objects' effective normal differs from their true geometric normal.

Radiosity is designed to cope with effective normals differing from "raw" geometric normals (e.g. due to bump mapping); however, the true geometric normal is never reported by the mesh or triangle object in the first place, so Radiosity never finds out about the discrepancy.

The problem results in radiosity samples occasionally picking up light from the inside of a mesh, which may in turn be illuminated through radiosity samples on the inside surface which happen to be picking up light from the outside.

As a workaround, use a recursion_limit of no more than 2. This should prevent any light from leaking into closed meshes and back out again. Alternatively, insert a slightly "slimmed down" copy of the mesh with a pitch black surface texture.

To truly solve this issue, geometric shapes need to be extended by an interface to report both the effective and true geometric normal of intersection points.

black splotches:
There is a bug in 3.6 and all prior versions (and a related flaw in 3.7.0 betas up to .29) regarding the handling of max_trace_level during radiosity sampling, almost inevitably causing stray black splotches, typically flocking near reflective or refractive objects. The only reliable way to get rid of these is to increase your scene's max_trace_level.

splotches in general:
Some people report that reducing both "count" and "error_bound" will help with some very stubborn scenes.

3.B Undocumented Details And Issues

adaptive quality parameters:
Various quality settings are automatically reduced when taking "deeper" radiosity samples. This includes the number of rays shot.

low_error_factor:
Contrary to documentation, this factor is effective during all pretrace steps, not just the last one, and it affects error_bound, minimum_reuse, and a hard- coded kind of "maximum_reuse" value.

Black Splotches:
There is a bug in 3.6 and all prior versions (and a related flaw in 3.7.0 betas up to .29) regarding the handling of max_trace_level during radiosity sampling, almost inevitably causing stray black splotches, typically flocking near reflective or refractive objects. The only reliable way to get rid of these is to increase your scene's max_trace_level.

Save/Load Limitation:
When saving previously gathered radiosity data, POV-Ray 3.6 only saves the "top- level" samples. If the samples are not sufficient for the scene they are loaded into, this will require a lot of "deeper" samples to be taken all over again.

Comments & Answers:

Christoph Hormann:
This is a very comprehensive description of radiosity workings in POV-Ray but i think you are mistaken about one central point: The radiosity pretrace will not be able to (and was never designed to) take all samples necessary during pretrace in a real life scene. The main reason is that a scene with edges and corners in the geometry (i.e. surfaces with infinite curvature) will require an infinite sample density at these corners and unless you do not shoot any camera rays during the final render pass that have not yet been traced during pretrace (and this usually always happens - aa, jitter) you will need additional samples in the render. To avoid this practically you would have to raise the error_bound during final trace by a huge amount (possibly adaptively like possible in latest megapov).

Clipka:
It may not have been *designed* to do so, but for good quality it *must* do so (at least to some good degree). Otherwise you get lots of artefacts everywhere near new samples taken during final render, because some nearby pixels were computed earlier, but would have been influenced by the new sample if it had been there already.

There are a few things to note out here:

- When talking about samples *needed*, I mean needed by *algorithm*, not by *theoretical* considerations.

- The algorithm in POV is designed so that it will enforce a certain minimum effective radius of samples, so in this sense it will never *need* an infinite number of samples.

- I'm advocating nothing more than *good* coverage, not a *perfect* one.

There's a point here to be made though, and it just gives me an idea how to automatically get an exhaustive enough but not too slow pretrace:

Samples taken during final render that affect only a single pixel probably don't hurt anyone. Artefacts appear only where samples are taken that should by algorithm affect other pixels already calculated.

So there must be a way to use this fact to come up with a criterion when to stop pretracing even though sample coverage doesn't seem to be rather incomplete yet.

Note however that just stopping pretrace at the final render's pixel size doesn't suffice: on areas seen from a very shallow angle, a spot that still needs samples may be "thin" enough on screen in one direction to slip through a pixel-sized pretrace, but still "wide" enough in the other direction to do harm.

So the radiosity tracing code should report the "apparent" size of the largest sample gathered for a pretrace ray (or some value directly enough related to it), and if the pretrace finds that it needs to gather no more "oversized" samples then it is (with a certain probability) safe to leave sampling in that area to the final trace.

Anthony D. Baye (povray.general, 10-8-2015):
In an effort to speed the render of the scene I'm working on, I reduced the radiosity settings to default, but wound up with large patches of pure white with jagged edges. I managed to fix some of it by increasing the number of pretrace steps, but there are still large ambient areas in the render.

Can anybody tell me what causes this, and which settings I might adjust to fix it? I can't find anything about this in the documents or tutorials.

Clipka:
An image might help to better diagnose the issue. Are the "patches of pure white light" isolated splotches, or do you have a general problem with jagged edges in the radiosity gradients?

Generally, jagged edges are an indication that a lot of radiosity samples are taken during the main render.

Whenever a sample is taken during the main render, some nearby pixels have already been rendered with the presumption that the new sample doesn't exist, using a different set of samples to interpolate between than any pixel rendered later, so you get a discontinuity in the gradient where the additional sample was taken. This isn't much of a problem if the set of samples to interpolate between is large already, as the discontinuity will be low in amplitude; but as the percentage of samples taken during the main render grows, so does both the total number */and/* amplitude of such discontinuities.

Among the report POV-Ray displays after finishing a render, there is a table listing how many radiosity samples were taken during which pass at which recursion depth, like this:

```
----------------------------------------------------------
      Pass   Depth 0          Total
----------------------------------------------------------
      1           48              148
      2           583             583
      3           2067            2067
      4           2028            2028
      Final       2571            2571
----------------------------------------------------------
      Total       7397            7397
      Weight      0.282
----------------------------------------------------------
```

As a rule of thumb, the number of top-level samples ("Depth 0") taken during the main render ("Final" pass) should be less than half the number of top-level samples taken in total.

If the number of samples taken during the main render is too high, this indicates that your pretrace coverage is too low, and you may need to do the following:

- Set "always_sample" to "off" (this is the default if "#version 3.7" or higher is specified); this will cause POV-Ray to only take samples during the main render if it absolutely needs them. If you instead set this to "on", artefacts are almost guaranteed.

– - Set "nearest_count" to a reasonably large value; this will cause POV-Ray to aim for a higher pretrace coverage. A value of 5 is ok, but 10 is better, and I personally always use 20 except for test renders.

– - Decrease "low_error_factor"; this will also cause POV-Ray to aim for a higher pretrace coverage, while at the same time making it less picky when it comes to re-using samples during the main render. (Note that this comes at the cost of "blurring" the radiosity effect; but when a low pretrace coverage is your problem, then that's probably your least concern.)

– - Increase the number of pretrace steps by decreasing "pretrace_end" ("pretrace_start" usually has only a small impact on the pretrace coverage, as long as it's reasonably large). If the number of pretrace steps is too low, POV-Ray will be unable to achieve the pretrace coverage as requested by the other parameters.

If "pretrace_end" is already at the lowest effective value (the inverse of the image size) and the number of pretrace steps is still lower than necessary to achieve the pretrace coverage prescribed by the other parameters, the following settings may also help; they all affect the pretrace coverage POV-Ray needs, but also the one it aims for:

– - increase "error_bound" to reduce the required and aimed-for coverage in general.
–
– - Increase "minimum_reuse" to reduce the required and aimed-for coverage near corners.
–
–

BUT
if your issue is isolated bright splotches, then the main problem is most certainly entirely different, namely that you have some comparatively small but very bright object in your scene. In that case, you only have two viable options: Up your radiosity "count" parameter, or try to replace that bright object with a conventional point or area light source (maybe with a looks_like object).

Alain:
In this case, using importance can help.

Add:
#default{radiosity{importance 0.001}}

or
#declare Average_Count = 75;
#declare High_Count = 160000;
#default{radiosity{importance Average_count/High_Count}}

In the radiosity block, set count to a large value:
count 160000

or
count High_Count // With declared identifier

In your small and bright object(s), add this:
radiosity{importance 1}

That way, on average, you'll use about 160 samples for most of the render, but up to 160000 for those problematic small and bright objects.

Overview of best parameter use by Alain Martel on 19-05-2020 (p.b.i):
Things that can affect the quality when using radiosity :
pretrace_end : The default of 0.04 is usually to high for renders over 800 pixels wide or high. Good values are under 0.01. 0.005 or 0.0025 are correct for renders size above 1200 pixels. More time in pretrace phase against less time of the render proper.

Reduce **low_error_factor** : Defaults to 0.5. Try values from 0.2 to 0.4. Will increase the sampling quality during the pretrace.

minimum_reuse : Default 0.015. Lowering can solve issues like your example using error_bound 1.8. A suggested starting value would be close to the pretrace_end value, maybe slightly lower.
pretrace_end 0.005
minimum_reuse 0.0045

Using the two values version of **count** : Very often, using count 100 1000 will give less artefacts.

Increasing **nearest_count** : Default is 5. Can go up to 20. Good at smoothing shadowed areas. Use nearest_count Value1 Value2 to enable adaptive pretrace.
Usage :
nearest_count 15 4

In scenes with small bright objects, you need a high **count**, but it greatly increase the rendering time. Use the **importance** mechanism.
Suggestion of use :

```
#declare HighCount = 10000;
#declare MeanCount = 75;
radiosity { ...
  count HighCount
   ...
}
#default radiosity {importance MeanCount/HighCount}
```

And add :
radiosity { importance 1 }
to your small but bright objects.
Larger bright object can get an importance between the set default and 1. Range is 0..1.

This cause POV-Ray to only use a few samples for most of the scene, but, when sampling near the bright and important objects, to use more samples.

Whenever you have sharp inward angles, some samples can leek out on the back side of a surface, causing dark or bright splotches. Any displacement, however small, of the camera will cause those to change. Add some bevel or rounding will greatly help. Often done by adding a clipped cylinder tangent to both surfaces of a flat piece at an intermediate angle. Sometimes, adding a little bit of focal blur can suppress those.