Radiosity In English

May 20, 1999

Paul Nettle

I've found many people shy away from radiosity simply because of the mystique surrounding it. I'll admit that I was a bit intimidated by the topic when I decided to tackle it. But the truth be known, it boils down to some pretty simple techniques, most of which are very common. As a matter of fact, if you've got access to a rendering architecture that supports z-buffering, you've got 90% of a basic radiosity processor that can produce some excellent results.

I understand that "Aggravated Nosebleed" (the source of this Q&A entry) has some basic knowledge of radiosity, but I'd like to take this opportunity to cover the basics from the very beginning. Hopefully he/she as well as some other readers will gain enough of a fundamental understanding to help ease the learning curve from some of the more notable references (which I will list later.)

In the beginning:

The introduction of Radiosity came in 1984 from Cornell University in a paper titled "Modelling the Interaction of Light Between Diffuse Surfaces" written by Goral, Torrance & Greenberg. The idea was to simulate energy (light) transference from diffuse surfaces. Diffuse surfaces are surfaces that reflect light equally in all directions – the opposite of a shiny surface.

This result was considered "view independent." This simply meant that the illumination on a surface looked the same no matter what angle you were viewing it from. For the sake of clarity, an example of the opposite ("view dependent") would be a reflective surface. Reflective surfaces are view dependent because the specular highlights would appear at a different position on the surface based on the angle at which the surface was viewed.

This view independence was nice, especially considering the cost (in processor power & running time) of radiosity processing. The illumination could be calculated once and the scene could then be rendered very quickly from any angle. This translates directly into many of today's modern "first-person shooter" games.

The first approach:

Consider a simple room with only four walls, a ceiling and a floor. Can you see it in your mind's eye? You better not be able to; I haven't specified a light source yet. :-) In radiosity, light sources aren't your typical point or spot light sources. In radiosity, light is emitted from surfaces. So, rather than adding a surface for a light source, lets just make the entire ceiling an "area light source." In the real world, this would translate to a cubic room where the ceiling was a huge panel of fluorescent lights behind a huge diffuse reflector (those smoked-glass looking things that spread light out.)

This example is a simple one since every surface can see every other surface. In other words, there's nothing to block the light from reaching any surface (i.e. no shadows.)

Each surface has two values associated with it. An amount of how brightly it is illuminated (its illumination) and how much of a surplus of energy it has (its radiative energy.) To start with, only the ceiling will have any radiative energy and all other surfaces will have no radiative energy or illumination.

What we need to do now is calculate the interaction of energy from every surface to every other surface. This is an n^2 problem since we need to calculate this interaction from each surface to every other surface in the scene. This can be calculated based on their geometrical relationships (distance between surfaces, relative orientation, relative area, etc.) The math that calculates this relationship results in a single value. This value is called a “form factor.”

The attentive readers (the one’s that are still awake) might have already guessed that there are only (n^2)/2 individual form factors since the relationship between surfaces 5&6 is the same as the relationship between 6&5. However, this is not true since relative area is taken into consideration.

We can calculate all the form factors in a scene and store them in a grid that is n elements wide by n elements tall. This grid is referred to as the “radiosity matrix” and it works just like a 2-dimensional table. Each element in this matrix contains a form factor for the interaction from the surface indexed by the column and the surface indexed by the row.

Remember how I said that there are n^2 interactions and not (n^2)/2? This is because each form factor is like a diode in that it only handles energy going in one direction: from a “source surface” to a “destination surface.” In this case, we’ll say that the source surfaces are index by columns and destination surfaces are indexed by rows. Source surfaces will emit their energy to the destination surfaces.

Now lets solve the matrix. To do this, we simply visit each column (source) in the matrix and emit energy to each row (destination) in that column. When we do this, we’ll be placing some of that radiated energy (from the source) in the illumination value for the destination. But these surfaces are reflectors, which means they’re going to reflect SOME energy back into the scene. Based on the surface’s reflectivity, we’ll add a little bit of energy to the destination’s radiative energy. This radiative energy will eventually make its way back into the scene (i.e. to the other surfaces) as we progress through the matrix.

If the destination is a perfect reflector (i.e. it reflects every single bit of energy it receives – a mirror) then there will be no energy stored in the destination’s illumination, it would all go to its radiative energy. The inverse is also true: a perfectly black surface might not reflect any energy back into the scene, absorbing it all, so every bit of energy it receives is stored in its illumination value. If you’re starting to think that we’re making a black surface white, we’re not. Remember, we’re dealing with light, so the color of a surface is ultimately multiplied by its illumination. In the case of the perfectly black surface, the surface remains visually black.

Once we’ve gone through the matrix once, we do it all over again. This is necessary because we we’re storing some energy as illumination, and some as radiative energy. Now it’s time to go through the matrix again and start distributing that reflected radiative energy.

We’ll go through this matrix over and over again until the total amount of radiative energy for all surfaces is relatively small.

The next step:

If you made it this far without getting lost, you’re in the home stretch. There’s still a lot we haven’t covered yet, so let’s move on. I’ll start with a few shortcomings of the basic radiosity matrix as I’ve described it thus far and common solutions to these issues.

Our surfaces have only one illumination value for the entire surface, so there is no change in illumination across a single surface. To solve this problem, we can simply subdivide each surface into a series smaller polygons called “patches.” If you do this, you simply treat each patch as its own surface as a replacement for the original surface. Your matrix will grow to the number of patches in the scene squared.

This brings us to our next issue: the matrix can be quite large (especially if you subdivide into a number of patches) If the scene is very simple (say, a meager 1,000 polygons) then our illustrious matrix will be pretty big (1,000,000 elements.) If you've subdivided each of those surfaces to a meager 8x8 grid of patches per surface, then we're talking about 4,096,000,000 total elements in our matrix (8*8 = 64 patches per surface, 64*1000 = 64000 patches per scene, 64000*64000 is = 4,096,000,000 total matrix elements.) This is pretty tough for any computer to swallow.

Before I discuss the solutions to this ever-increasing matrix, let's talk about a related issue: a matrix of this magnitude would take a long time to solve. Especially considering the fact that we'll have to solve it multiple times. If a mistake was made in the modeled scene, wouldn't it be nice to know this sooner rather than later?

Progressive Refinement:

In 1988, Cohen, Chen, Wallace & Greenberg published a paper called "A Progressive Refinement Approach to Fast Radiosity Image Generation." This paper described a new way of solving radiosity. It was quite clever in that it reordered the way things were done.

In the matrix method, illumination was gathered by each destination element from its source element. Ironically, this is called "gathering." The progressive refinement approach reversed this and defined (the other incredibly ironic term) "shooting."

The basic idea behind progressive refinement starts by finding the surface with the most energy to contribute to the scene (i.e. has the highest amount of radiative energy.) This surface would then iterate through all other surfaces, distributing its energy along the way. After this process was completed, the image was then rendered for the user, and the process began again, finding the surface with the most energy to contribute to the scene. Each pass would cause another render of the scene, allowing the user to progressively evaluate the progress. If the progress showed a problem along the way (an illumination surface was in the wrong place or the wrong color) they could stop the process and make the needed adjustments.

During this process, the user would see a completely dark scene progress to a fully lit scene. To accommodate this sharp contrast in visual difference from beginning to end, the progressive refinement technique added something called the "ambient term".

Before I continue, I want to point something out that is pretty important in radiosity. There is no such thing as ambient light in real life. Ambient light is something that was invented to accommodate the need for what appears to be a "global light" in real life. But in reality, ambient light doesn't exist. Rather, light is always being reflected from surface to surface, which is how it finds its way into all the nooks and crannies of real-world detail. Before the advent of radiosity, ambient light was the best thing available to the typical rendering architectures. It is safe to think of radiosity is a more accurate solution to ambient (global) light. This is why radiosity is considered a technique for "global illumination."

The ambient term starts off as a "differential area sum" of the radiative energy for the entire scene. What this means is that it's a number that represents the average amount of light that each surface will receive throughout the processing of the entire radiosity solution. We can calculate that average without doing all the work simply because it's an average amount of energy, not a specific amount of energy for a single surface.

As each progressive pass emits the radiative energy for a surface, the ambient term is slowly decreased. As the total radiative energy of the scene approaches zero, so does the ambient term (though, at different rates, of course.) A nice advantage here is that you can use the ambient term to figure out when you've distributed enough energy as to make only a negligible difference. At this point, you can stop processing.

So, the progressive approach has solved the massive memory requirements for the radiosity matrix by simply not storing it, and it partially solves the processing time issue by speeding things up, and further improving this by allowing users to preview their works in progress.

A Note on Patches:

Before I continue, I want to cover the topic of patch subdivision just a little. I only touched on it lightly so as not to confuse the reader. It's time we dive just a little bit deeper in to these ever useful things.

First, let's be perfectly clear on something. If you use subdivision in your radiosity code, then you will not be using "surfaces" since the patches are a higher resolution representation of the original surface geometry. It will be the patches that shoot and gather energy amongst themselves, not the surfaces. If you use patch subdivision, you can probably discard your original surfaces since they have been replaced by a higher resolution representation, their patches.

Patches are how we simulate area light sources. Rather than actually treating the surface like an area light source, we simply split it up into lots of smaller light sources across the entire area of the original surface. If the surface is subdivided enough, then the results can be quite pleasing.

Patch subdivision can be done blindly or intelligently. An example of blind subdivision might be to subdivide every surface into a set of patches that are one square foot each. This can be quite a waste, since we only really need the subdivision in high-contrast areas (i.e. an area of a surface that has a dramatic change in energy across a relatively small area – like a shadow boundary.)

There is a multitude of intelligent subdivision techniques. One of the most common is to subdivide progressively by adding another step to the process. Once a surface has fully emitted its energy, each patch in the existing data-set is visited and a decision is made if two adjoining patches have too much of a difference in their illumination values. If they do, there will be a sharp contrast between these two patches so you should subdivide each of them. You can pick any threshold you wish to contain your subdivisions to a minimum. You can also set a maximum subdivision level to prevent from subdividing too much.

Patches, however, are just the first step to subdivision. Patches themselves can be subdivided into "elements". The usefulness of elemental subdivision is for performance reasons as well as aesthetic reasons. Patch subdivision can be pre-set to a specific resolution. In this case, the entire scene is subdivided evenly into patches of a specific size. This sounds like a waste, but let's not get hasty. The subdivision resolution can be quite low in this case. As the radiosity solution progresses, the patches are intelligently subdivided into elements based on high contrast areas (or whatever intelligent subdivision technique you decide to use.)

You can think of elements as a higher resolution representation of their "parent" patches. But unlike patch subdivision where the surfaces are discarded and replaced by patches, patch subdivision does not discard the patches. The advantage here, is that the patches are maintained for shooting, while the elements are used for gathering.

Let's look at that a little more closely. A patch is subdivided into a grid of 8x8 elements. During the distribution process, the patch with the highest amount of radiative energy is chosen for energy distribution. Energy is distributed from that patch to all of the ELEMENTS in the scene. The elements retain their illumination value (for beauty's sake) and the radiative energy that would be reflected from all the elements is then sent up to their parent patch. Later, the patch will do the shooting, rather than each individual element. This allows us to have a high resolution of surface geometry with a lower resolution distribution. This can save quite a lot of processing time, especially if the average patch is subdivided into 8x8 elements.

For the sake of this example, I'll just assume we're not at the elemental subdivision stage yet, and work from patches.

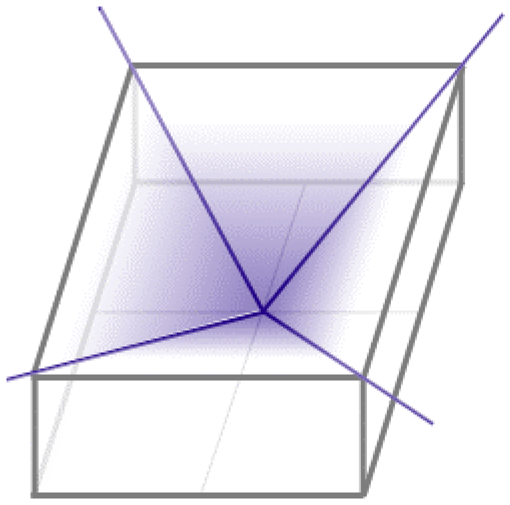
The Hemicube:

Did somebody say shadows? I didn't. Not yet, at least. :-)

To obtain shadows, we need to have some visibility information, so we'll know how much of a patch is vi sible from another patch. One of the most common ways of doing this in today's world is to use a z-buffer. And radiosity is no different. To do this, however, we'll need a way to generate a z-buffer from a patch. This is where the hemicube comes in handy.

A hemicube is exactly what it sounds like. It's exactly one half of a cube, split orthogonally along one axis. This gives us one whole face, and four half-faces.

What's it for? Try to picture this: place a pin-hole camera at the base of the hemicube (i.e. the center of the cube prior to cutting it in half) and point the camera at the center of the top face. Now set your camera to a 90-degree frustum.



You can consider the top face of the hemicube now, to be the rendering surface of the camera. This surface has a pixel resolution (which I'll discuss shortly.) If you render the scene from this perspective, you'll "see" what the patch "sees".

Remember when I said that we need to take the relative distance and relative orientation of two patches into account to calculate their form factors? Well, in this case, we no longer need to do that. The hemicube takes care of that for us. As patches are rendered onto the surface of the hemicube, they'll occupy "hemicube pixels". The farther away the surface is, the fewer pixels it will occupy. This is also true for patches at greater angles of relative orientation. The greater the angle, the fewer pixels it will occupy. Using a z-buffer we can let some patches partially (or fully) occlude other patches, causing them to occupy even fewer pixels (or none at all) which gives us shadows.

For this to work, we need to translate these renders into energy transmission. Let's talk about that for a bit.

A standard z-buffer renderer will render color values to a frame buffer and store depth information into a z-buffer. A hemicube implementation is very similar. It keeps the z-buffer just like normal. But rather than storing color values into a frame buffer, it stores patch IDs into a frame buffer.

When the render is complete, you have partial form factor information for how much energy gets transmitted from one patch to another. I say “partial form factor information” because we’re missing one piece.

This information is lacking some of the relative angle information between two patches. The relative angles are used to decrease the amount of energy shot from one patch to another. The greater the angle, the less energy is transmitted. Our hemicube gives us part of this information by only telling us (in an indirect way) how much of an angle the destination patch is relative to us. But we also need to take the shooter’s relative angle into account as well. It’s much like Lambert shading. As the surface turns away from the light, the surface receives less light. We’ve got this information (indirectly) in the hemicube frame buffer. But our light source is also an area, which means it can turn, too. So we’ll need to take its angle into consideration before we shoot any energy to anybody.

The hemicube has a wonderful mechanism for this. It’s called the “delta form factor.” This is simply a table of values. It is the same resolution as the surface of the hemicube and it contains values that are used to scale the amount of energy that each hemicube pixel can transmit. The values in this table associated with the center pixels of the top face have the highest value, and the values fall off as they get near the edges of the hemicube face. The reason for this is simple. The values associated with the center of the hemicube face have the highest value since anything rendered to this area of the “screen” will be directly in front of the hemicube (i.e. the least incident angle.) The values in the table associated with the edges of the hemicube face are at a 45-degree angle, so they are considerably less than those found near the center.

There is a very specific calculation for the “delta form factor” table which can be found in most radiosity references.

To finish up our hemicube explanation, we need to pull it all together.

Rather than shooting light from the source patch to the destination patches, we do this through each pixel in the hemicube’s frame buffer (remember, we’ve stored patch IDs in there so we can reference them later, and THIS is later :-). Visiting each hemicube pixel, we simply scale the amount of the shooter’s total radiative energy by the delta form factor associated with that pixel. This means that each patch will receive a little bit of energy for each pixel it resides in, in the hemicube’s frame buffer. Each of these partial energy transmissions to an individual destination patch will all add up to the proper amount of total transmitted energy, just like magic.

How do we know that we’ve transmitted all the energy from the shooter? Well, if you add up all the delta form factors for the hemicube, you’ll find they add up to 1.0. This is a good test, by the way, to make sure your hemicube delta form factor table is correct. Remember to account for error, so the value might not equal exactly 1.0, rather something very close.

A typical hemicube resolution might be 128x128. However, you may decide to go with a higher resolution. Either way, remember this: each pixel in the hemicube’s delta form factor table contains a very small fractional value. You should consider using doubles to store these values as they can get VERY small.

To save confusion, I purposely neglected to mention a few things. Each hemicube has five sides. I only described the process for rendering the top face. The remaining half-faces also require rendering, but the process is identical to that of the top face. Don’t worry, your radiosity references will cover how to calculate the delta form factors for ALL faces of the hemicube. And don’t forget that when you run your little test that adds up all the delta form factors to a result of 1.0, you’ll need to include ALL of the delta form factors, not just those for the top face.

In closing, I should mention that there are issues with hemicubes (like aliasing artifacts under certain circumstances.) There are some solutions to these issues as well as totally different techniques. But hemicubes are a great place to start your radiosity adventures.

Recommended references:

“Advanced Animation & Rendering Techniques” by Watt & Watt
“Computer Graphics Principles & Practice” by Foley, vanDam, Feiner & Hughes
“Radiosity: A Programmer's Perspective” by Ashdown
“Radiosity and Realistic Image Synthesis” by Cohen & Wallace
“Radiosity and Global Illumination” by Sillion & Puech

Personally, I originally learned the concepts & fundamentals of radiosity from “Advanced Animation & Rendering Techniques.” I learned enough to get my first radiosity processor up and running. Since this book has such a wealth of other information, I highly recommend it for first-timers on the subject. From there, you can graduate to any of the other references listed. If you make it to the more advanced stuff, you’re welcome to visit my site and grab some research papers on the subject.

- Paul Nettle
- midnight@GraphicsPapers.com
- http://www.GraphicsPapers.com
- http://www.FluidStudios.com