

TNM005 - Computer Graphics

Essay

Global Illumination



Fredrik Larsson, 790501-1917, frela306@student.liu.se
David Kästel, 790425-6257, davka237, facknr.: 1190

Abstract:

This essay gives a brief introduction to common lighting-model issues in 3D-graphics. We are then trying to give a deeper explanation of two major theories behind global illumination: Ray tracing and Radiosity.

It is written as an article for some kind of magazine in the popular science domain. It should not require any prior knowledge.

Sources:

- 3D Computer Graphics, Alan Watt, Addison Wesley
- Radiosity by Allen Martin
(<http://www.cs.wpi.edu/~matt/courses/cs563/talks/radiosity.html>)
- Radiosity by Hugo Elias
(<http://freespace.virgin.net/hugo.elias/radiosity/radiosity.htm>)
- What is Raytracing by Siddharta Chaudhuri
(<http://fuzzyphoton.tripod.com/whatisrt.htm>)

Isn't that a photo?

Most of today's Hollywood movies contain some computer graphic-produced environments. New products can be visualized before manufacturing them. We can buy a non-existing house by just looking at some computer-generated pictures. For amusement we play games that take place in highly realistic settings.

The funny part of this is the fact that we often cannot decide whether we're looking at photos, film or just some reproduction of real life surroundings. How is it possible to generate these syntheses? Which methods do computer graphic artists use to implement physical phenomena such as light? We have with this essay the intention of giving the background and application of two illumination-models called, Ray tracing and Radiosity.

How about a cup of tea?

Computer graphics is vastly spread in all kind of visualization-matters. Over the past 30 years we have been able to simulate three-dimensional pictures with computers. At first just easy models. By building an object with several triangles we have created a so-called polygon mesh model.

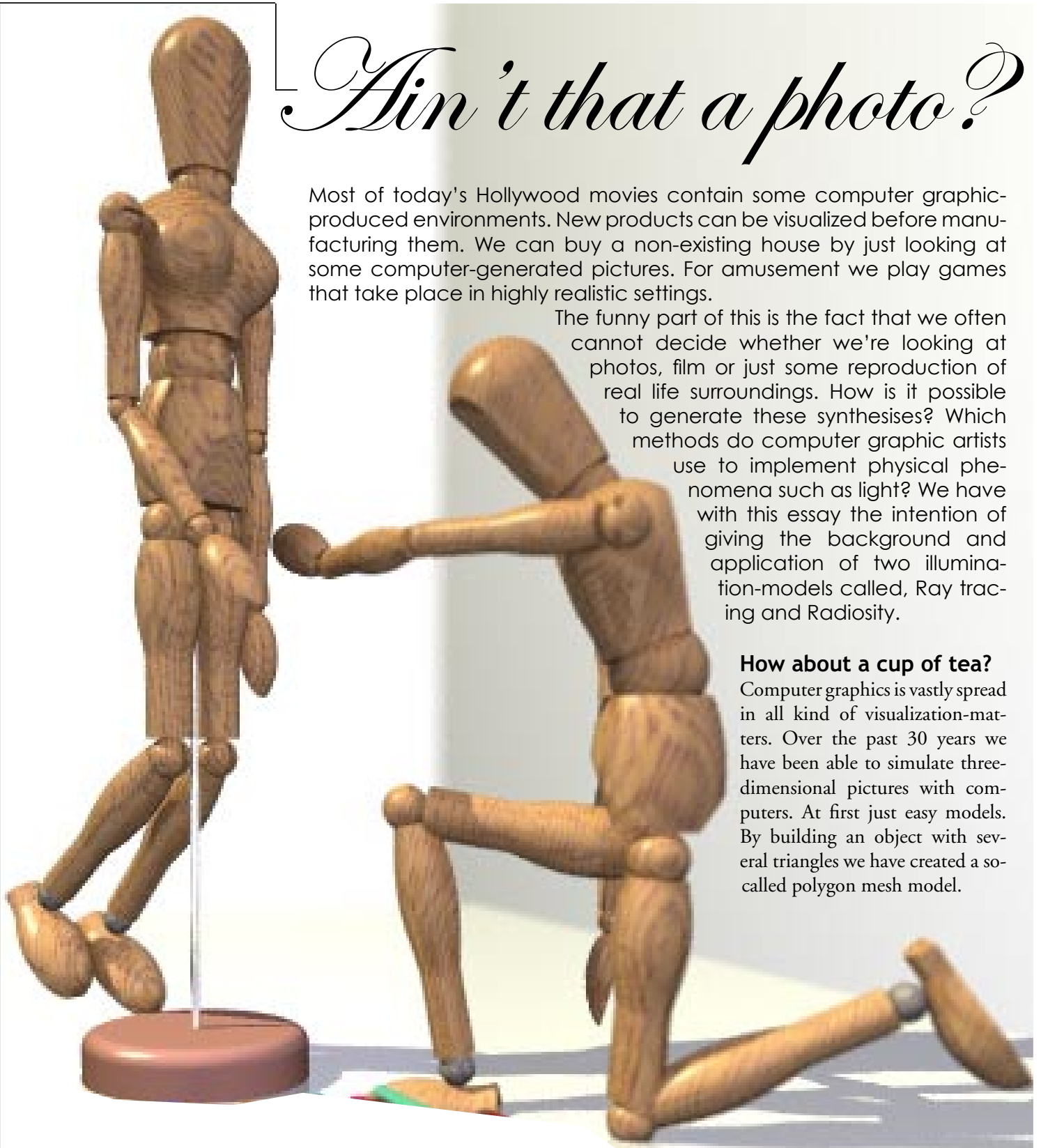
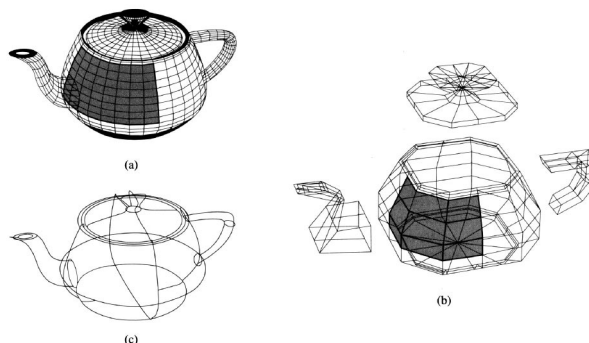


Fig. 1
(a) The Utah Teapot, wire-model.
The teapot is in (a) shown as a polygonal mesh. It is modelled with some bézier-curves (c). These so-called splines are quite easy described mathematical curves.
Figure (b) shows the control point of the curves.



The next step was to use mathematically described curves, such as the Bézier-curve. With this technique they developed a teapot at the University of Utah. Something that has become a symbol for 3D-graphics. *Fig 1.*

Things have changed a lot and nowadays, as mentioned above, almost everything around us can

be reproduced with photo-realistic precision. To achieve this effect we have to know how nature behaves. Especially how light interacts with all the objects in an environment.

A particle and a wave

At the most fundamental level, we see objects when light bounces off the surfaces (or when light is produced by the objects themselves) and reaches our eyes. Light is a form of radiant energy with two properties. Combining Planck's (Max Planck, the father of quantum theory) equation of light's energy $E = hf$ and Einstein's famous ditto $E = mc^2$ gives us that light has a mass: $m = hf / c^2$. Thus, light behaves like a particle. But we often speak of "light waves". Like sound, light may be considered as a periodic disturbance of a medium, which propagates from one place to another. This duality explains all the "features" of light.

We are not interested in the details of these two theories. Instead we will from now on consider the concept of a light ray. This may be understood in the languages of both the particle theory and the wave theory. For the optics used in computer graphics the ray theory will be enough to explain certain important phenomena.

A ray is always a straight line along the path of propagation of the light. This straight line is defined as the shortest distance between two points, which gives us the path taken by the light. An interesting fact used when calculating how light behaves: objects can reflect or refract light according to known physical rules shown in figure 2.

Building a 3D-scene

First of all we create and develop objects in three dimensions, that's *modeling*. Very often we use so-called wire-presentation of the scene we're building. (fig. 1) After a while we want to see a first draft of the result, we have to *render* the scene. That's a projection (like a photo camera) onto a two-dimensional plane, where the image consists of so-called pixels (=picture elements). Most 3D-software (e.g. 3D Studio Max or Maya.) includes a render-engine but there are stand-alone applications as well.

When rendering a modelled scene, normally, we won't see anything of the objects created. Why? We haven't defined a light source yet. There are two

kinds of sources, which are interesting for this essay: point sources and ambient light.

The easiest way of implementing light in 3D is with a point source, comparable with a spotlight. This gives us *local illumination*. Ambient light, on the contrary, is the general brightness in an environment.

Another important issue when it comes to illumination-models in the simulating of realistic computer graphics is the application of the Phong-model. Here the light at any given point is made up of three components: diffuse, specular and ambient. The additive components are combined to determine the illumination of a point or polygon.

These given background details leads us to the actual topic of this essay: We use *global illumination* to obtain photo-realism.



When rendering 3D-images we have to consider the different physical properties of light.

Global illumination

We are now interested in the difficult problem of simulating the interaction of light with an entire environment. Light has to be tracked through the environment from emitters to sensors, rather than just from an emitter to a surface then directly to the sensor or eye, as with direct illumination-models.

It requires some kind of algorithm to calculate the way light travels through the scene. Lit objects generate shadows. The effect of reflection of objects in each other and transparency effects are matters that have to be implemented. Another main problem in the rendering of 3D-scenes is the elimination of hidden surfaces, i.e. surfaces that are not visible from the position of the eye.

With today's methods we are able to get very close to photo-realism but the two established global algorithms are just partial. They are just dealing with some of the physical properties of light, which is enough for their purpose. Even though the idea behind the algorithms can look simple they generate a great amount of calculations that have to be made.

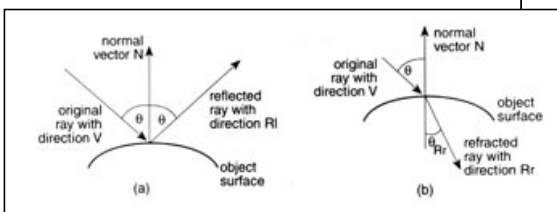


Fig. 2.
(a) Reflected ray (b) Refracted ray

Ray tracing

We are now ready to understand what ray tracing involves. The name itself is a clue. First we have to consider that ray tracing can produce hyper-realistic images but it can only do this when given a scene with point light sources, and perfectly shiny objects. Now, a house probably isn't full of perfectly shiny spheres and point light sources. In fact, unless one live in a universe with completely different physics, a house probably contains hardly any super-sharp shadows. Ray tracing is very spread and common used anyway because the result is often very astonishing, *fig. 3*.

The method

A ray tracing program calculates the illumination effects of a surface by tracking, or tracing, the path of a light ray as it bounces off or is refracted through the surface. Starting at the light source and travelling to the eye's position, the view plane, gives us an immensely wasteful function. In most cases, about 99.9% of the tested rays will not reach the eye. Therefore the method starts with a ray per pixel from the view plane. When this ray intersects a surface, two child rays in the reflected and refracted directions are generated.

For the reflected ray the direction is given by the reflection rule whereas the refracted directions is given by the refraction-index. *Fig. 2*. The intensity of one particular point (e.g. P_1 in *fig. 4*), where the ray is divided in sub rays, consists of three components:

1. The contribution of the direct light source. According to the local illumination-model.
2. The intensity of the reflected sub ray.
3. The intensity of the refracted sub ray.

The total intensity of one pixel

In our example, *fig. 4*;

$$I_{\text{pixel}} = I(P_0) + I(P_1) + I(P_2) + I(P_3)$$

can be illustrated with a tree. We traverse this tree, which gives a sum of all contributions to the pixel.

The subdivisions of the rays have to be stopped after a while; otherwise the



Fig. 3. Rendering with the ray tracing illumination model gives this perfect result. This is possible because our scene contains objects with only perfect shiny surfaces.

algorithm will be too heavy calculated. This occurs when either a ray leaves the scene or hits a diffuse surface. If this doesn't happen we have to define some criteria regarding when to stop the ray division, i.e., the primary ray can give

intersection points. This demands a lot of attention, 90-95% of the total calculations needed. Another problem is that ray traced shadows seem too sharp (aliasing-effects) and are too black.

To overcome the biggest problem, the intersection calculus, there are several optimization methods. We here present two of them:

1. Bounding volumes

We encapsulate a complicated modelled object within an easily defined one, a sphere, for instance. This reduces the number of intersection tests needed. To overcome problems with details in the bounding volumes one can construct a hierarchy of bounding volumes.

2. Spatial subdivision

We divide the 3D-space into a certain amount of cubic elements, so-called voxels. All the objects that are completely or partly within the voxel are linked

to a list. When the ray passes, only the objects in these "filled" voxels have to be examined.

Aliasing-effects could be reduced by sample with more than one ray per pixel. For example, we sample one ray per pixel corner from which we calculate the average intensity for the whole pixel.

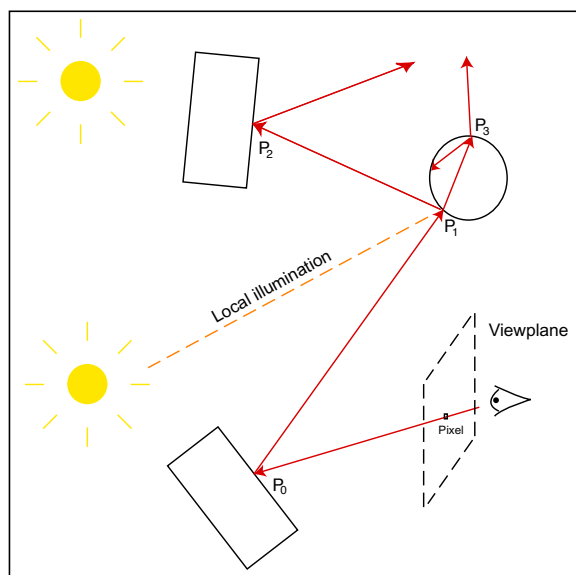


Fig. 4. Simple example of calculating the total intensity of a certain pixel with ray tracing method.

birth to a maximum of 5 children. This is actual the depth of the ray-tree. The depth can be adaptive, when a ray gives less then a certain given value of intensity the subdivision stops.

Optimize

What are the problems with ray tracing? One example is calculating the

Radiosity

As we pointed out earlier, the ray tracing method doesn't support diffuse surfaces. We will describe another method to go around this obvious disadvantage. The method is called radiosity. Using radiosity gives us very realistic lightning for diffuse surfaces. That means: the 3D pictures look even more like realistic photos, because the way radiosity brighten up the environment is closer to reality, and the laws of physics. Realistic because all the shadow edges are softer (as gradients) than with the ray tracing method.

As we can see, this image has been rendered with both ray tracing and radiosity. With radiosity we get no sharp shadows.



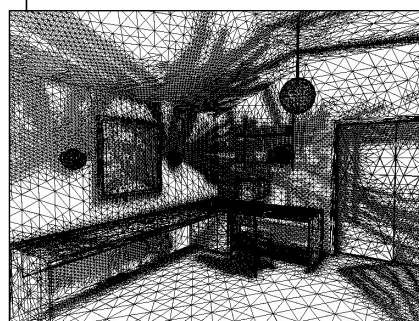
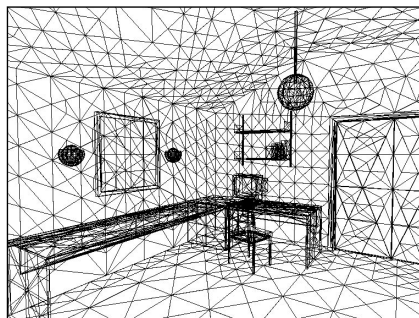
Radiosity is view independent. We can place the camera (our previously mentioned view-plane) wherever we want without the need of recalculating the illumination.

The method

The radiosity method analyses the light reflections in something named patches. A patch is a very small part of the picture. It's up to the creator to decide how many patches one object in the picture needs. Obviously the lightning in the picture gets "better" (more realistic) if every object is built with a large number of patches.

In the pictures beside (fig. 5) it's easy to see the difference in the result depending on how many patches that are used.

*Fig. 5.
A fewer amount of patches (upper images) is easier to calculate but doesn't look as realistic as scenes with more patches (lower images)*



Patches

So how do we use the patches? There are two important principles for the understanding of patches and radiosity in general:

- There is no difference between light sources and objects.
- A surface in the scene is lit by all parts of the scene that are visible to it.

This is easy to understand when we implement the radiosity method. Every patch stands for an individual analysis of their own lightning condition. Imagine that you are one of this thousands and thousands of patches in a picture. This single patch calculates the total amount of light in the scene that reaches just this patch. Since all patches are located in different places in the scene, every patch gets different results in this calculation.



Example

If we are working with a scene like the one in picture (a) in *fig. 7* the result should turn out to be something like this (c) in *fig. 7*. This is because the conditions for the scene are that a light source is placed on the outside (c) in *fig. 6*. Every patch that is directly hit by the ray gets lit. But the result of this is not a very realistic scene. So we have to do the calculation process once again. In this second round (*Round 2*) some of the patches that were dark are going to receive some light from other patches that were hit by the light in the first step. The scene is beginning to take on a more realistic look.

You could say that by making this second calculation round you have decided that reflections in the scene are allowed to appear once. As we all know and can study empirically just by looking around the room light has a tendency to reflect more than one time. For that reason we have to do our famous calculation round again. For every round the result looks more and more like a photo (e.g. *round 5*) It's impossible to tell how many rounds one have to do, it depends on the complexity of the scene, and the lightness of the surfaces.



The resulting image from our example.

The Algorithm

The radiosity model can be mathematically described like this:

$$B_i = E_i + \lambda_i \sum B_j \cdot F_{ji}$$

This can seem strange or even hard to understand but after our depth-look of

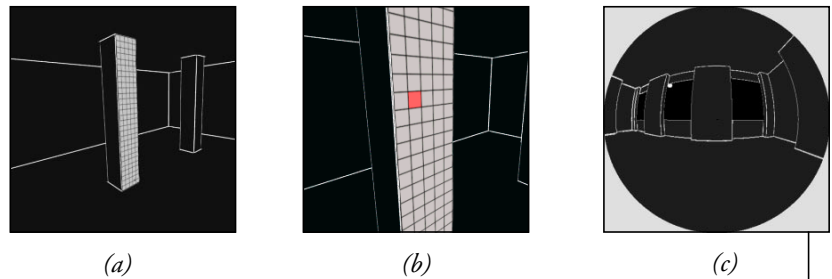


Fig. 6. Our scene (below, *fig. 7*) contains two pillars (a). If we look from the perspective of one single patch (b) on this pillar we will see the sun (c) as a small light source.

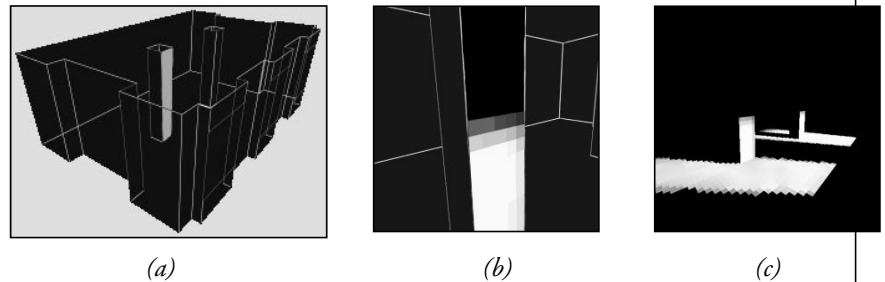
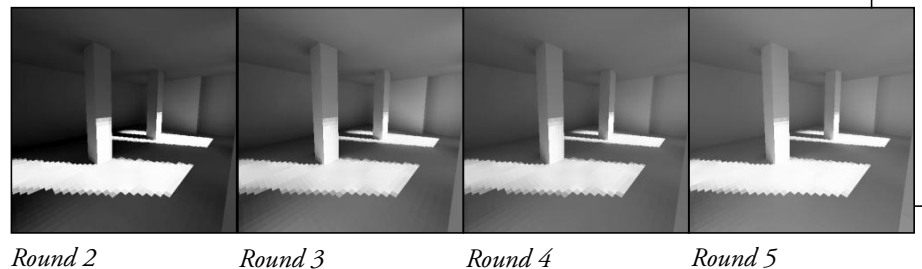


Fig. 7. This scene (a) is lit by the sun just once but gets too dark (c, "Round 1"). By calculating the contribution of the rest of the scene's all patches' reflections we will get from round to round (below) a more realistic appearance.



this formula every question mark will be straightened out.

As some readers might remember we establish the fact that there is no difference between light sources and objects. In the real world it's easy to decide what object that is a light source and what object that is not. The chair you are sitting on is not a light source, but the lamp over your head is (or even the sun if you are practising outdoor education while reading this). In the computer world this might not be that obvious. Therefore we have to calculate if the patch emit light or not. This is E_i . If it's equal to 0 the patch is not emitting light. So in most cases we set E_i to 0. It's only direct light sources that have another (positive) value.

The next part of the formula is λ_i . This value describes how much the

patch reflexes the light.

The last part might be the one that is most tricky. B_j describes how much light that is leaving any given patch (j). And F_{ji} describes how much this effects the single patch that we are calculating (i). So $B_j \cdot F_{ji}$ is the product of the influence from the j -patch. We now have to summarize the contribution of every single patch in the scene. $\sum B_j \cdot F_{ji}$ typically you write this as .

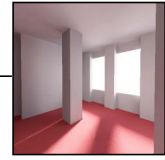
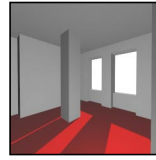
These are the things that we have to have in mind when calculating how much light that emits from every single patch. As anyone familiar with advance mathematics can see, this gives us a complex system with many variables. But it's no problem solving it, because there are a similar number of equation systems that there are variables.

Conclusion

As discussed in this essay we cannot produce a synthesis of the world around us without considering how light effects our surroundings.

The two methods' major advantages and disadvantages have been discribed and can also be seen in the list to the right.

We think that one should combine the use of radiosity and raytracing to obtain the best result, as in the beautiful image below.



Advantages

Ray Tracing:

- Do handle point sources and shiny surfaces
- Can render both mathematically described objects and polygons
- Allows you to do some cool volumetric effects

Radiosity:

- Very realistic lighting for diffuse surfaces
- Conceptually simple and easy to implement
- Easy to optimise with 3D hardware

Disadvantages

- Slow
- Very sharp shadows and reflections (aliasing)

- Slow
- Does not handle point sources well
- nor shiny surfaces

