Radiosity (computer graphics)

From Wikipedia, the free encyclopedia
(Redirected from Radiosity (3D computer graphics))

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Čeština
★ Deutsch
Español
Esperanto
Français
한국어
Hrvatski
Italiano
עברית
Nederlands
日本語
Polski
Svenska
中文
Edit links

This article **may be too technical for most readers to understand**. Please help improve this article to make it understandable to non-experts, without removing the technical details. The talk page may contain suggestions. *(July 2009)*

In 3D computer graphics, **radiosity** is an application of the finite element method to solving the rendering equation for scenes with surfaces that reflect light diffusely. Unlike rendering methods that use Monte Carlo algorithms (such as path tracing), which handle all types of light paths, typical radiosity only account for paths (represented by the code "LD*E") which leave a light source and are reflected diffusely some number of times (possibly zero) before hitting the eye. Radiosity is a global illumination algorithm in the sense that the illumination arriving on a surface comes not just directly from the light sources, but also from other surfaces reflecting light. Radiosity is viewpoint independent, which increases the calculations involved, but makes them useful for all viewpoints.

Radiosity methods were first developed in about 1950 in the engineering field of heat transfer. They were later refined specifically for the problem of rendering computer graphics in 1984 by researchers at Cornell University.[2]

Notable commercial radiosity engines are Enlighten by Geomerics (used for games including Battlefield 3 and Need for Speed: The Run); 3ds Max; form•Z; LightWave 3D and the Electric Image Animation System.

Scene rendered with RRV[1] (simple implementation of radiosity renderer based on OpenGL) 79th iteration.

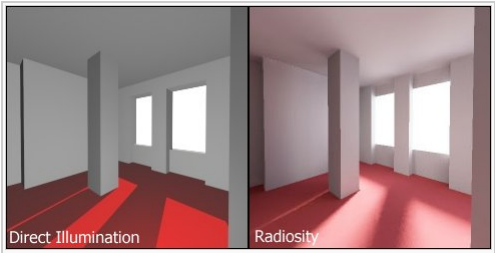
Contents [hide]

Visual characteristics [edit]

The inclusion of radiosity calculations in the rendering process often lends an added element of realism to the finished scene, because of the way it mimics real-world phenomena. Consider a simple room scene.

The image on the left was rendered with a typical **direct illumination renderer**. There are *three types* of lighting in this scene which have been specifically chosen and placed by the artist in an attempt to create realistic lighting: **spot lighting** with shadows (placed outside the window to create the light shining on the floor), **ambient lighting** (without which any part of the room not lit directly by a light source would be totally dark), and **omnidirectional lighting** without shadows (to reduce the flatness of the ambient lighting).

The image on the right was rendered using a **radiosity algorithm**. There is only **one source of light**: an image of the sky placed outside the window. The difference is marked. The room glows with light. Soft shadows are visible on the floor, and subtle lighting effects are noticeable around the room. Furthermore, the red color from the carpet has bled onto the grey walls, giving them a slightly warm appearance. None of these effects were specifically chosen or designed by the artist.



Difference between standard direct illumination without shadow umbra, and radiosity with shadow umbra

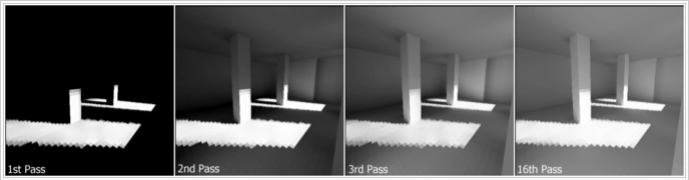
Overview of the radiosity algorithm [edit]

The surfaces of the scene to be rendered are each divided up into one or more smaller surfaces (patches). A view factor (also known as *form factor*) is computed for each pair of patches; it is a coefficient describing how well the patches can see each other. Patches that are far away from each other, or oriented at oblique angles relative to one another, will have smaller view factors. If other patches are in the way, the view factor will be reduced or zero, depending on whether the occlusion is partial or total.

The view factors are used as coefficients in a linear system of rendering equations. Solving this system yields the radiosity, or brightness, of each patch, taking into account diffuse interreflections and soft shadows.

Progressive radiosity solves the system iteratively with intermediate radiosity values for the patch, corresponding to bounce levels. That is, after each iteration, we know how the scene looks after one light bounce, after two passes, two bounces, and so forth. This is useful for getting an interactive preview of the scene. Also, the user can stop the iterations once the image looks good enough, rather than wait for the computation to numerically converge.

Another common method for solving the radiosity equation is "shooting radiosity," which iteratively solves the radiosity equation by "shooting" light from the patch with the most error at each step. After the first pass, only those patches which are in direct line of sight of a light-emitting patch will be illuminated. After the second pass, more patches will become illuminated as the light begins to bounce around the scene. The scene continues to grow brighter and eventually reaches a steady state.



As the algorithm iterates, light can be seen to flow into the scene, as multiple bounces are computed. Individual patches are visible as squares on the walls and floor.

Mathematical formulation [edit]

The basic radiosity method has its basis in the theory of thermal radiation, since radiosity relies on computing the amount of light energy transferred among surfaces. In order to simplify computations, the method assumes that all scattering is perfectly diffuse. Surfaces are typically discretized into quadrilateral or triangular elements over which a piecewise polynomial function is defined.

After this breakdown, the amount of light energy transfer can be computed by using the known reflectivity of the reflecting patch, combined with the view factor of the two patches. This dimensionless quantity is computed from the geometric orientation of two patches, and can be thought of as the fraction of the total possible emitting area of the first patch which is covered by the second.

More correctly, radiosity B is the energy per unit area leaving the patch surface per discrete time interval and is the combination of emitted and reflected energy:

$$B(x)\,dA = E(x)\,dA + \rho(x)\,dA \int_S B(x')\frac{1}{\pi r^2}\cos\theta_x \cos\theta_{x'} \cdot \mathrm{Vis}(x,x')\,\mathrm{d}A'$$

where:

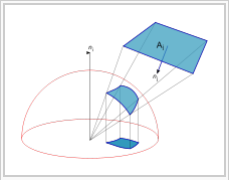
- $B(x)_i\,\mathrm{d}A_i$ is the total energy leaving a small area $\mathrm{d}A_i$ around a point x.
- $E(x)_i\,\mathrm{d}A_i$ is the emitted energy.
- $\rho(x)$ is the reflectivity of the point, giving reflected energy per unit area by multiplying by the incident energy per unit area (the total energy which arrives from other patches).
- S denotes that the integration variable x' runs over all the surfaces in the scene
- r is the distance between x and x'
- θ_x and $\theta_{x'}$ are the angles between the line joining x and x' and vectors normal to the surface at x and x' respectively.
- $\mathrm{Vis}(x,x')$ is a visibility function, defined to be 1 if the two points x and x' are visible from each other, and 0 if they are not.

If the surfaces are approximated by a finite number of planar patches, each of which is taken to have a constant radiosity B_i and reflectivity ρ_i, the above equation gives the discrete radiosity equation,

$$B_i = E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j$$

where F_{ij} is the geometrical view factor for the radiation leaving *j* and hitting patch *i*.

This equation can then be applied to each patch. The equation is monochromatic, so color radiosity rendering requires calculation for each of the required colors.



The geometrical form factor (or "projected solid angle") F_{ij}.

F_{ij} can be obtained by projecting the element A_j onto the surface of a unit hemisphere, and then projecting that in turn onto a unit circle around the point of interest in the plane of A_i. The form factor is then equal to the proportion of the unit circle covered by this projection.

Form factors obey the reciprocity relation $A_i F_{ij} = A_j F_{ji}$

Solution methods [edit]

The equation can formally be solved as matrix equation, to give the vector solution:

$$B = (I - \rho F)^{-1} E$$

This gives the full "infinite bounce" solution for B directly. However the number of calculations to compute the matrix solution scales according to n^3, where *n* is the number of patches. This becomes prohibitive for realistically large values of *n*.

Instead, the equation can more readily be solved iteratively, by repeatedly applying the single-bounce update formula above. Formally, this is a solution of the matrix equation by Jacobi iteration. Because the reflectivities ρ_i are less than 1, this scheme converges quickly, typically requiring only a handful of iterations to produce a reasonable solution. Other standard iterative methods for matrix equation solutions can also be used, for example the Gauss–Seidel method, where updated values for each patch are used in the calculation as soon as they are computed, rather than all being updated synchronously at the end of each sweep. The solution can also be tweaked to iterate over each of the sending elements in turn in its main outermost loop for each update, rather than each of the receiving patches. This is known as the *shooting* variant of the algorithm, as opposed to the *gathering* variant. Using the view factor reciprocity, $A_i F_{ij} = A_j F_{ji}$, the update equation can also be re-written in terms of the view factor F_{ji} seen by each *sending* patch A_j:

$$A_i B_i = A_i E_i + \rho_i \sum_{j=1}^{n} A_j B_j F_{ji}$$

This is sometimes known as the "power" formulation, since it is now the total transmitted power of each element that is being updated, rather than its radiosity.

The view factor F_{ij} itself can be calculated in a number of ways. Early methods used a *hemicube* (an imaginary cube centered upon the first surface to which the second surface was projected, devised by Cohen and Greenberg in 1985). The surface of the hemicube was divided into pixel-like squares, for each of which a view factor can be readily calculated analytically. The full form factor could then be approximated by adding up the contribution from each of the pixel-like squares. The projection onto the hemicube, which could be adapted from standard methods for determining the visibility of polygons, also solved the problem of intervening patches partially obscuring those behind.

However all this was quite computationally expensive, because ideally form factors must be derived for every possible pair of patches, leading to a quadratic increase in computation as the number of patches increased. This can be reduced somewhat by using a binary space partitioning tree to reduce the amount of time spent determining which patches are completely hidden from others in complex scenes; but even so, the time spent to determine the form factor still typically scales as *n* log *n*. New methods include adaptive integration[3]

Sampling approaches [edit]

The form factors F_{ij} themselves are not in fact explicitly needed in either of the update equations; neither to estimate the total intensity $\sum_j F_{ij} B_j$ gathered from the whole view, nor to estimate how the power $A_j B_j$ being radiated is distributed. Instead, these updates can be estimated by sampling methods, without ever having to calculate form factors explicitly. Since the mid 1990s such sampling approaches have been the methods most predominantly used for practical radiosity calculations.

The gathered intensity can be estimated by generating a set of samples in the unit circle, lifting these onto the hemisphere, and then seeing what was the radiosity of the element that a ray incoming in that direction would have originated on. The estimate for the total gathered intensity is then just the average of the radiosities discovered by each ray. Similarly, in the power formulation, power can be distributed by generating a set of rays from the radiating element in the same way, and spreading the power to be distributed equally between each element a ray hits.

This is essentially the same distribution that a path-tracing program would sample in tracing back one diffuse reflection step; or that a bidirectional ray tracing program would sample to achieve one forward diffuse reflection step when light source mapping forwards. The sampling approach therefore to some extent represents a convergence between the two techniques, the key difference remaining that the radiosity technique aims to build up a sufficiently accurate map of the radiance of all the surfaces in the scene, rather than just a representation of the current view.

Reducing computation time [edit]

Although in its basic form radiosity is assumed to have a quadratic increase in computation time with added geometry (surfaces and patches), this need not be the case. The radiosity problem can be rephrased as a problem of rendering a texture mapped scene. In this case, the computation time increases only linearly with the number of patches (ignoring complex issues like cache use).

Following the commercial enthusiasm for radiosity-enhanced imagery, but prior to the standardization of rapid radiosity calculation, many architects and graphic artists used a technique referred to loosely as false radiosity. By darkening areas of texture maps corresponding to corners, joints and recesses, and applying them via self-illumination or diffuse mapping, a radiosity-like effect of patch interaction could be created with a standard scanline renderer (cf. ambient occlusion).

Static, pre-computed radiosity may be displayed in realtime via Lightmaps on current desktop computers with standard graphics acceleration hardware.

Advantages [edit]

One of the advantages of the Radiosity algorithm is that it is relatively simple to explain and implement. This makes it a useful algorithm for teaching students about global illumination algorithms. A typical direct illumination renderer already contains nearly all of the algorithms (perspective transformations, texture mapping, hidden surface removal) required to implement radiosity. A strong grasp of mathematics is not required to understand or implement this algorithm[*citation needed*].

A modern render of the iconic Utah teapot. **Radiosity** was used for all diffuse illumination in this scene.

Limitations [edit]

Typical radiosity methods only account for light paths of the form LD*E, i.e. paths which start at a light source and make multiple diffuse bounces before reaching the eye. Although there are several approaches to integrating other illumination effects such as specular[1] and glossy [2] reflections, radiosity-based methods are generally not used to solve the complete rendering equation.

Basic radiosity also has trouble resolving sudden changes in visibility (e.g. hard-edged shadows) because coarse, regular discretization into piecewise constant elements corresponds to a low-pass box filter of the spatial domain. Discontinuity meshing [3] uses knowledge of visibility events to generate a more intelligent discretization.

Confusion about terminology [edit]

Radiosity was perhaps the first rendering algorithm in widespread use which accounted for diffuse indirect lighting. Earlier rendering algorithms, such as Whitted-style ray tracing were capable of computing effects such as reflections, refractions, and shadows, but despite being highly global phenomena, these effects were not commonly referred to as "global illumination." As a consequence, the term "global illumination" became confused with "diffuse interreflection," and "Radiosity" became confused with "global illumination" in popular parlance. However, the three are distinct concepts.

The radiosity method in the current computer graphics context derives from (and is fundamentally the same as) the radiosity method in heat transfer. In this context radiosity is the total radiative flux (both reflected and re-radiated) leaving a surface, also sometimes known as radiant exitance. Calculation of Radiosity rather than surface temperatures is a key aspect of the radiosity method that permits linear matrix methods to be applied to the problem.

See also [edit]

- Raytracing

References [edit]

1. ^ Dudka, Kamil. "RRV - Radiosity Renderer and Visualizer". Retrieved 1 February 2013.
2. ^ "Cindy Goral, Kenneth E. Torrance, Donald P. Greenberg and B. Battaile,Modeling the interaction of light between diffuse surfaces",, *Computer Graphics*, Vol. 18, No. 3.
3. ^ G Walton, *Calculation of Obstructed View Factors by Adaptive Integration*, NIST Report NISTIR-6925, see also http://view3d.sourceforge.net/

Further reading [edit]

- Radiosity Overview, from HyperGraph of SIGGRAPH (provides full matrix radiosity algorithm and progressive radiosity algorithm)
- Radiosity, by Hugo Elias (also provides a general overview of lighting algorithms, along with programming examples)
- Radiosity, by Allen Martin (a slightly more mathematical explanation of radiosity)
- ROVER, by Dr. Tralvex Yeap (Radiosity Abstracts & Bibliography Library)
- Radiosity: Basic Implementations (Basic radiosity survey)

External links [edit]

- RADical, by Parag Chaudhuri (an implementation of shooting & sorting variant of progressive radiosity algorithm with OpenGL acceleration, extending from GLUTRAD by Colbeck)
- Radiosity Renderer and Visualizer (simple implementation of radiosity renderer based on OpenGL)
- Enlighten (Licensed software code that provides realtime radiosity for computer game applications. Developed by the UK company Geomerics)


Computer graphics portal

Categories: Global illumination algorithms | Heat transfer | 3D computer graphics