# NeXtRad:
# A Radiosity-Lighted Walkthrough Package

The Theory and Implementation of Radiosity

Jason R. Wilson

Contents

## 1. NeXtRad--A Radiosity-Lighted Walkthrough Package

NeXtRad is a three-dimensional interactive walkthrough simulation which uses radiosity-based lighting calculations. This document gives the theory behind the radiosity method as well as implementation notes.

## 2. What is Radiosity?

### Realistic Image Generation

The goal of realistic image synthesis is to generate an image of a scene that evokes from the visual system a response indistinguishable from that evoked by the actual environment. Although the current state of the art does not realize this goal, there are several methods that incorporate global illumination algorithms such as ray-tracing and radiosity that give good approximations.

**Modelling a Scene**

A scene is what we are trying to view. An example of a scene is a living room. The scene may contain objects such as chairs, lamps, and tables. In order for realistic image generations to produce an image of a scene, the scene must be described in a format that is recognizable by the computer. This process is called modelling. NeXtRad models are specified using a set of input surfaces (modelled as polygons) and vertices (for details on NeXtRad's modeler consult the NeXtRad user's manual). These polygons and vertices are the input to the realistic image generator.

**Global and Local Illumination**

Illumination is the process of tracking light in a room and determining how much light falls on each surface. The local illumination model describes how the surface reflects and transmits light. So, given a description of the light incident on a surface, it predicts the intensity, spectral character, and directional distribution of the light leaving the surface. So the next question is, "How do we determine the description of light incident to a surface?" For this, we need the global illumination model. To get an idea of the complexity of the global illumination model, consider a scene with three surfaces labeled **a**, **b**, and **c**. To determine the amount of light incident on surface **a**, we need to know how much light from surfaces **b** and **c** reach surface **a**. But we don't know how much light is reflected off surface **b** until we know the amount of light incident on **b**--a computation which requires that we know the amount of light from surface **a** that reaches **b**!! Hopefully, from this simple example, you can see that all of the global illumination computations are interdependent.

Radiosity is a method of illumination that includes global illumination (sometimes called indirect lighting). The method was originally used in thermal engineering for finding heat/light distributions.

## A Simplifying Assumption : Diffuse Reflections

If you recall, one of the jobs of the local illumination model is to compute a directional distribution of the light leaving a surface. Radiosity assumes only diffuse reflections. Diffuse reflections are assumed to be uniform in all directions. This turns out to be a dramatic simplification and allows the radiosity algorithm to be view-independent (another global illumination algorithm, ray-tracing, is view-dependent). So, with radiosity only one solution is needed for any view but with ray-tracing, each view has a different solution. This makes radiosity an ideal method for walkthroughs, such as NeXTRad, which allow the user to change the view point on the fly. So what is the catch? Yes, we do pay for this simplification. Specular surfaces (surfaces that have directionally dependent reflection functions) such as mirrors and marble floors are not modeled correctly. However, since radiosity is mostly used for building interiors with diffuse objects such as chairs and walls, this is a major problem. Some more recent algorithms have been developed that incorporate specular effects into the radiosity solution.

**Model Discretization**

As mentioned, surfaces are modelled with polygons. In order to achieve a finite solution, the polygons of a scene are subdivided into smaller patches that are assumed to have constant local illumination. One goal of the mesh generator, discussed in the mesh-generation paper, is to determine the set of patches that make up the scene.

## 3. The Radiosity Equations

Radiosity is essentially a finite-element method. Although each patch has three radiosity values (one for each color band), I will assume monochromatic surfaces for simplicity. The radiosity of a patch is given in terms of the radiosities of the other patches. This system of equations is then solved using a iterative linear solver. The radiosity of patch i is given below:

$$B_i = E_i + p_i \sum B_j F_{ij}$$

These equations generate the following system of linear equations:

The following lists the meanings of each part of the equation:

**Radiosity : ($B_i$)** The basic quantity we want to compute for each patch i.
**Emission : ($E_i$)** Light that the patch i emits itself, as in the case of a light source. Known at run-time.
**Reflectivity : ($p_i$)** A number between 0 and 1 which indicates the fraction of light which is reflected from patch i. Also known at run-time.
**Form-Factor :($F_{ij}$)** The fraction of light leaving patch i that arrives at patch j.

**The Form-Factors**

The form factors are based on the geometry of the scene and computing them is the most time-consuming part of radiosity. Fortunately, they never have to be recomputed (unless scene geometry changes). The following derivation of the form-factor refers to the figure given below.

The form factor specifies the fraction of the energy leaving one surface which lands on another. For non-occluded environments, the form factor for a differential area is given by :

$$F_{dAidAj} = \frac{\text{Cos}ø_i \, \text{Cos}ø_j}{\pi r^2}$$

The form-factor for a patch and a differential area is given by :

$$F_{dAiAj} = \int_{A_j} \frac{\text{Cos } ø_i \text{ Cos } ø_j}{\pi r^2} \, dA_j$$

The patch-to-patch form factor is defined to be the area average and is thus:

$$F_{AiAj} = (1/A_i) \iint_{A_i A_j} \frac{\text{Cos } ø_i \text{ Cos } ø_j}{\pi r^2} \, dA_j \, dA_i$$

Finally, to account for the possibility of hidden surfaces we have :

$$F_{AiAj} = (1/A_i) \iint_{A_i A_j} \frac{\text{Cos } ø_i \text{ Cos } ø_j}{\pi r^2} \, \text{HID} \, dA_j \, dA_i$$

where HID = 1 if and only if differential area i can see differential area j.

## 4. The Hemi-Cube Algorithm

In complex scenes with occluded (hidden) surfaces, this integral cannot be evaluated in the general case. Fortunately, the Hemi-Cube algorithm for approximating form factors has been developed.

### Assumptions

Assume that the distance between the two patches is large compared to their size and they are not partially occluded. If these conditions are not met, further subdivision of patches (the mesh-generator's job) will be necessary. When these conditions are met, it can be seen that the value of the inner integral remains almost constant. Therefore, the patch-to-patch form factor can be approximated with the value of the inner integral for the center point of patch i.

## Nusselt's Analog

For a finite area, the form-factor (inner integral) is equivalent to the fraction of the circle covered by projecting the area onto the hemisphere and then orthographically down into the circle (see figure below).

## From Hemi-Sphere to Hemi-Cube

However projecting onto the sphere for form-factor computation is impractical (too slow). From the figure below, it can be seen that any two patches in the environment, which when projected onto the hemisphere occupy the same area and location, will have the same form-factor value.

For example, patch A has the same form-factor as patch D which has the same form-factor as patch E. Thus to compute the form-factor of patch A, we can compute the form-factor of patch D which is a much easier task than computing the form-factor of patch E. This basic idea leads us to the hemi-cube algorithm. With the hemi-cube algorithm, a row of form-factors is computed in one step by projecting the environment (patches) down onto the hemi-cube algorithm surrounding a single patch (see figure below).

NeXtRad does the hemi-cube projections in five separate passes--one for each side of the hemi-cube. Each one of these passes is almost identical to the view projections done during display (leads to substantial code reuse).

**Computing Form-Factor of Projected Patch**

Now we have to compute the form-factor for the projected patch. This is done by discretizing the hemi-cube (usually 100X100). Then a rasterization is performed to determine which pixels the projected patch covers (also known as scan-conversion). For example, see the figure below:

Each pixel of the hemi-cube can be thought of as a small patch. Then to compute the form-factor of the projected patch, the aggregate sum of the form-factors of the covered pixels is determined. The form-factors for the pixels are called delta form-factors and are precomputed and stored for easy lookup. The accuracy of this process can be increased by increasing the resolution of the hemi-cube. This algorithm can easily be extended to handle occluded surfaces: If two patches project onto the same pixel, the one that is closest gets the pixel (basically, a z-buffer algorithm).

**Derivation of a Delta Form-Factor**

The last step is to determine how to derive the delta form-factors. Fortunately, the geometry is simple enough to determine them analytically. The following example derivation of a form-factor for a pixel on the top face of the cube refers to the figure below:

First, a differential-area to differential-area form-factor is computed (see equation given above):

r = sqrt (x^2 + y^2 + 1)

Cos $Ø_i$ = Cos $Ø_j$

r*cosØ = 1

differential form-factor = (cos Øi * cos Øj)/(Pi * r^2)

substitute 1/r for the cos terms ->

differential form-factor = (cos Øi * cos Øj)/(Pi * r^4)

Next, substitute r with right-hand side given above ->

differential form-factor = 1/(Pi * (x^2 + y^2 + 1)^2)

Now, we must compute the differential area to patch form-factor (i.e. center of the bottom patch is the differential area). Assuming that the differential area to differential area form factors are constant, the integrand stays constant and the integral can be determined by multiplying by the area.

Thus,

**delta form-factor = area (A) / (Pi * (x^2 + y^2 + 1)^2)**

Deriving the delta form-factors for pixels on the sides of the hemi-cube is similar and will not be covered in detail.

## 5. Computing the Radiosities

### Solving The Radiosity Matrix

Once we have computed the form-factors, we are in the position to solve the radiosity matrix (refer to the matrix given above for this discussion). For starters, note that the diagonal values are one--assuming that the patches are convex, $F_{ii}$ equals zero for all i. By definition, the sum of any row of form-factors is equal to one. In the matrix to be solved, each form-factor term is multiplied by a surface reflectivity, which is also less than one. Thus, the summation of the absolute values of all terms in any row exclusive of the main diagonal term is always less than one. Hence, the matrix is strictly diagonally dominant (the sum of the absolute values of each row is less than the diagonal term). Therefore, Gauss-Siedel iteration is guaranteed to converge. In practice it only takes about 5-8 iterations to converge within 1%. Finally, the initial guess for Gauss-Siedel the column of emission values.

**Vertex Radiosities**

Now we have the patch radiosities. To display, the vertex radiosities are needed. These are found by simply averaging the surrounding patch radiosities.

## 6. Displaying the Radiosities

The display stage displays the scene given the vertex radiosities using linear interpolation and a z-buffer algorithm for hidden surface removal. Because the range of the radiosities is not between 0 and 1, all of the final colors are scaled by the maximum to take full advantage of the full range of colors.

Texture mapping also occurs during the display phase. The color of the pixel is determined by multiplying the color of corresponding texture pixel (found with an inverse-mapping) by the radiosity value for that pixel. To make the algorithm tractable, the reflectivities of a textured-patch are determined by averaging together the texture colors. This allows patch radiosities to be computed without having a patch for each texture pixel. When display is done, the actual texture-value is used to compute the color.

Finally, because the solution is view-independent the only recomputation necessary for a change of view is the display phase.