Rendering

Sung-eui Yoon
KAIST
September 4, 2016

- This chapter was updated at Sep., 2016.
-
- This is downloaded from `http://sglab.kaist.ac.kr/~sungeui/render`.

Chapter 6

Radiosity

In the last chapter, we discussed ray tracing techniques. While ray tracing techniques can support various rendering effects such as shadow and transparency, their performance was identified too slow to be used for interactive graphics applications. Some of issues of ray tracing is that we generate many rays whenever we change viewpoints. Furthermore, processing those rays take high computation time, and they tend to have random access patterns on underlying data structures (e.g., meshes and bounding volume hierarchy), resulting in high cache misses and lower computational performance.

On the other hand, radiosity emerges as an alternative rendering method that works for special cases with high performance [GTGB84]. While radiosity is not designed for handling various rendering effects, it has been widely used to complement other rendering techniques, since radiosity shows high rendering performance of specific material types such as diffuse materials. In other words, radiosiy as well as ray tracing are two common building blocks of designing other advanced rendering techniques, and we thus study this technique in this chapter.

6.1 Two Assumptions

Radiosity has two main assumptions (Fig. 6.1):

- **Diffuse material.** We assume that the material type we handle for radiosiy is diffuse or close to the diffuse materials. The ideal diffuse material reflects incoming light into all the possible outgoing directions with the equal amount of light energy, i.e., the same radiance, which is one of radiometric quantity discussed in Sec. 7. Thanks to this diffuse material assumption, any surface looks the same and has the same amount of illumination level given the view point. This in turn simplifies many computations.

- **Constant radiance per each surface element.** Take a look at a particular surface (e.g., a wall or a desk in your room). The illumination level typically varies smoothly depending on the configuration between a point in the surface and position of light sources. To support this phenomenon, radiosity treats that each surface is decomposed into surface elements such as triangles. It then assumes for simplicity that each surface element has a single value related to the illumination level, especially, radiosity value (Ch. 7). Simply speaking, radiosity is the total incoming (or outgoing) energy arriving in a unit area in a surface.

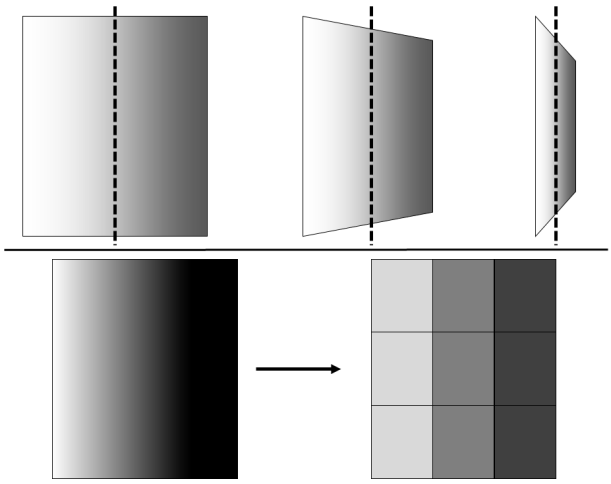

Figure 6.1: Radiosity has the diffuse material assumption (top) and constant illumination per surface element (bottom).

We will see how these assumptions lead to a simple solution to the rendering problem.

Relationship with finite element method (FEM). As you will see, radiosity can generate realistic rendering results with an interactive performance, while dealing only with diffuse materials and light sources. This was excellent results, when radiosity was proposed back at 1984. Furthermore, approaches and solution for radiosity were novel at the graphics community at that time. Nonetheless, those techniques were originally introduced for simulating heat transfers and have been well established as Finite Element Methods (FEM). FEM was realizing its potential benefits around 1960s and 70s, and was applied even to a totally different problem, physically based rendering. This is a very encouraging story to us. By studying and adopting recently developing techniques into our own problem, we can design very creative techniques in our own field!

6.2 Radiosity Equation

An input scene to radiosity is commonly composed of triangles. We first subdivide the scene into smaller triangles such that our assumption of the constant radiance per each subdivided triangle is valid. Suppose that there are n different surface elements. We use B_i to denote radiosity of a patch i. Some of such patches can be light sources and thus emit some energy. Since we also assume the light sources to be diffuse emitters, we also use radiosity for such self-emitting patches, and their emitting energy is denoted by $B_{e,i}$.

Intuitively speaking, the radiosity of the patch i is the sum of the self-emitting energy from

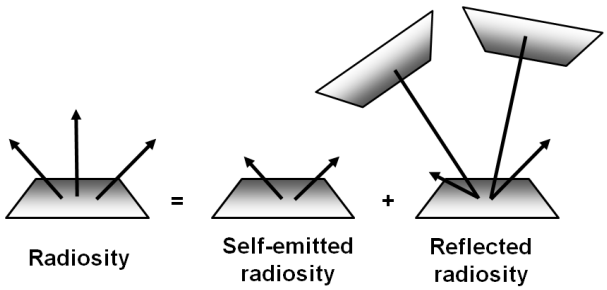

Figure 6.2: The radiosity of a patch is computed by the sum of the self-emitted radiosity from itself and the radiosity reflected and received from other patches.

the patch itself, $B_{e,i}$, and the energy reflected from the patch i by receiving energy from all the other patches (Fig. 6.2. We can then model the interaction between the patch i and different patches as the following:

$$B_i = B_{e,i} + \rho_i \sum_j B_j F(i \rightarrow j), \tag{6.1}$$

where j is another index to access all the surface elements in the scene, $F(i \rightarrow j)$ is a form factor that describes how much the energy from the patch i arrives at another patch j, and ρ_i is a reflectivity of the patch i.

$B_{e,i}$ and ρ_i are input parameters to the equation and given by a scene designer. The form factor is a term that we can compute depending on the geometric configuration between two patches i and j. The form factor can be understood by the area integration of the rendering equation, which is more general than the radiosity equation. This is discussed in Sec. 8.2. As a result, the unknown terms of the equation is the radiosity B_i of n different patches. Our goal is then to compute such unknown terms. We discuss them in the next section, followed by the overall algorithm of the radiosity rendering method.

6.3 Radiosity Algorithm

Given the radiosity equation (Eq. 6.1), the unknown term is the radiosity, B_i, per each patch, resulting in n different unknown radiosity values for n patches. Since we can setup n different equation for each patch based on the radiosity equation, overall we have n different equations and unknowns. When we represent such n different equations, we have the following matrix representation:

$$\begin{bmatrix} 1-\rho_1 F(1\to 1) & -\rho_1 F(1\to 2) & \dots & -\rho_1 F(1\to n) \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F(n\to 1) & -\rho_n F(n\to 2) & \dots & 1-\rho_n F(n\to n) \end{bmatrix} \begin{bmatrix} B_1 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} B_{e,1} \\ \vdots \\ B_{e,n} \end{bmatrix} \tag{6.2}$$

The above matrix has the form of $AX = B$, where $X = [B_1 \dots B_n]^T$ is a 1 by n matrix containing unknowns.

To compute the unknown X, we can apply many matrix inversion algorithms including Gaussian elimination that has $O(n^3)$ time complexity [PFTV93]. This approach, however, can be very expensive to be used for interactive applications, since the number of surface elements can be hundreds of thousands in practice.

Instead of using exact approaches of computing the linear equations, we can use other numerical approaches such as Jacobi and Gauss-Seidel iteration methods. Jacobi iteration works as the following:

- **Initial values.** Start with initial guesses on radiosity values to surface patches. For example, we can use the direct illumination results using Phong illumination considering the light sources as the initial values for surface patches.
- **Update step.** We plug those values, i.e., old values, into the right term of the radiosity equation (Eq. 6.1), and get new values on B_i. We perform this procedure to all the other patches.
- **Repeat until converge.** We repeat the update step until radiosity values converge.

The Jacobi iteration method has been studied well in numerical analysis, and its properties related to convergence have been well known [PFTV93].

One numerical iteration simulates one bounce of the light energy from a patch to another patch.

Effects of numerical iteration. Instead, we discuss how it works in the context of rendering. While performing the update step of the Jacobi iteration, we compute a new radiosity value for each patch from old values. In this process, we compute the new radiosity value received and reflected from other patches. Intuitively, the update step supports one bounce of the light energy from a patch to another patch.

Fig. 6.3 visualizes how radiosity values change as we have different number of update steps, i.e., passes. While only surface elements that are directly visible from the light source are lit in the first pass, other surface elements get brighter as we perform multiple update steps and thus multiple bounces. In a way, this also visualizes how the incoming light energy is distributed across the scene. In the end, we see only the converged result, which is the equilibrium state of the light and material interaction described in the radiosity equation.

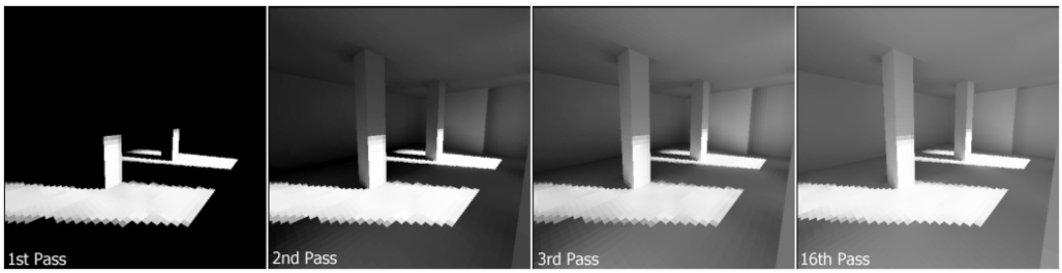

Figure 6.3: This shows a sequence of images computed by different updates, i.e., light bounces, during the radiosity iteration process. This is the courtesy of the wikipedia.

Overall algorithm. In summary, we subdivide triangles of the input scene into smaller surface elements, i.e., patches. We then compute radiosity values per each patch by solving the linear equations given by the radiosity equation. For static models, we perform this process only a single time. At runtime, when a viewer changes a view point, we then project those triangles whose color. This projection process is efficiently performed by using the rasterization process in GPUs.

Radiosity is commonly accelerated by adopting the rasterization method

So far, we did not consider view points given by users while computing radiosity values. This is unnecessary, because we do not need to consider view-dependent information for radiosity computation process; note that radiosity algorithm assumes the diffuse materials and emitters and thus we get the same radiance value for any view directions. This is one of the main features of the radiosity algorithm, leading to its strength and weakness of the method.

Drawbacks of the basic radiosity method. The main benefit of the basic radiosity method is that we can re-use the pre-computed radiosity values, even though the user changes the viewpoint. Nonetheless, it has also drawbacks. First of all, the radiosity assumes different materials and emitters, while various scenes have other materials such as glossy materials. Also, when we have dynamic models, we cannot re-use pre-computed radiosity values and thus re-compute them. Some of these issues are addressed in Ch. 12.

The basic radiosity method does not support glossy materials.

6.4 Light Path Expressions

The radiosity method does support light reflections between diffuse materials, but does not support interactions between glossy materials. Can we represent such light paths that the radiosity method supports?

Regular expressions are used to denote different types of light paths.

Heckbert proposed to use the regular expression to characterize light paths [Hec90]. This approach considers light paths starting from the eye, noted E, to the light, denoted, L. Diffuse, specular, and glossy materials are denoted as D, S, and G, respectively. We also adopt various

operations of regular expressions such as | (or), * (zero or more), and + (one or more).

The light paths that radiosity method are then characterized by LD^*E. On the other hand, the classic ray tracing method (Ch. 5) supports $L(DS^*)E$, since it generates secondary rays when a ray hits specular or refractive objects.