Lecture 13: Radiosity - Principles

Reflectance

Earlier in the course we introduced the reflectance equation for modelling reflected light, $I_{reflected}$, from a surface:

$$I_{reflected} = k_a + k_d\,(\mathbf{n}\cdot\mathbf{l})\,I_{incident} + k_s(\mathbf{r}\cdot\mathbf{v})^q\,I_{incident}$$

Where $I_{incident}$ is the incident light intensity and the constants represent:

- k_a : the amount of ambient light
- k_d : the amount of diffuse reflection
- k_s and q : control the amount of specular reflection

We used this lighting model for calculating shading values for polygons using both Phong and Gouraud shading. We also used the same equation when calculating the illumination at a ray object intersection while ray tracing. In both cases we assumed that there was a small number of point light sources, or if light was distributed then it came from a point source at infinity.

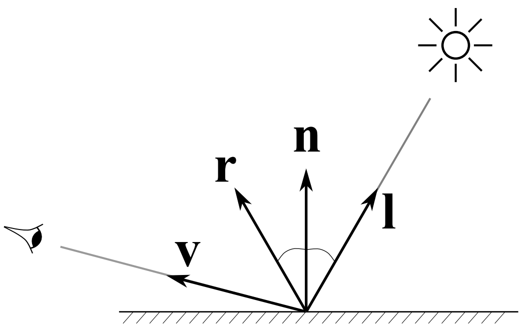


Figure 1: The vectors used in the reflectance equation

However, according to the reflectance equation, every surface in a graphics scene is emitting light. So far, we have only considered the emitted light travelling in the viewing direction. We have neglected the emitted light travelling in other directions which will contribute to the illumination of neighbouring objects. In practice we did not attempt to calculate this, but rather chose a constant k_a to represent the ambient light. We will now attempt to model it more accurately through the use of radiosity.

A better approximation to the reflectance equation is to make the ambient light term a function of the incident light as well:

$$\begin{aligned} I_{reflected} &= k_a\,I_{incident} + k_d\,(\mathbf{n}\cdot\mathbf{l})\,I_{incident} + k_s(\mathbf{r}\cdot\mathbf{v})^q\,I_{incident} \\ &= (k_a + k_d\,(\mathbf{n}\cdot\mathbf{l}) + k_s(\mathbf{r}\cdot\mathbf{v})^q\,)\,I_{incident} \end{aligned}$$

or more simply to write (for a given viewpoint)

$$I_{reflected} = R\,I_{incident}$$

where R is the viewpoint dependant reflectance function.

Radiosity

For any given surface (polygon) of our model we can define the term *Radiosity* as the energy per unit area leaving a surface. It will not be constant over the surface of a polygon. It is the sum of energy emitted by the surface itself plus any reflected energy due to light arriving from other surfaces. A surface might not emit any light energy of its own but if it does, then this light energy is emitted from the entire area of a patch rather than simply from a point.

We assume that the emitted energy is constant over a given surface and, for a small area of the surface, dA, the total energy leaving the patch is the sum of the energy it emits and the light energy it reflects:

$$BdA = E\,dA + R\,I$$

We can divide up the scene into a number of polygons and we can write B_i for the energy leaving the i^{th} patch, E_i for the energy it emits, R_i for its reflectance value and I_i for that light energy that is incident upon it. With this notation, the above equation can be re-written

$$B_i = E_i + R_i I_i \tag{1}$$

We are now treating each polygon of our scene as a distributed light source. The incident energy at any patch is collected from all other patches. In particular, the light energy reaching the i^{th} patch from the j^{th} patch is equal to the energy leaving the j^{th} patch (B_j) multiplied by a constant that links patch i with patch j called the *form factor*. The form factor is written F_{ij} and is described in more detail in the next section. In summary, the light energy reaching the i^{th} patch from the j^{th} patch can be written as $B_j\,F_{ij}$ and the total incident light at patch i is obtained by summing the incident light from all the patches:

$$I_i = \sum_{j=1}^{n} B_j\,F_{ij}$$

where n is the number of patches in the scene and we can assume that $F_{ii} = 0$. We can substitute this expression into Equation 1 to obtain:

$$B_i = E_i + R_i \sum_j B_j\,F_{ij}$$

If we can solve this for every B_i, then we will be able to render each patch directly with a correct light model. The values of B_i in the equation are the actual colour that is used to render the patch, so each B_i, E_i and R_i is a three dimensional vector quantity for an RGB colour image. The form factors are the same for each RGB dimension.

We can re-write the equation for the i^{th} patch slightly to get

$$B_i - \sum_j R_i\,B_j\,F_{ij} = E_i$$

and, putting together the equations for all patches, we can formulate the problem as the following matrix equation:

$$\begin{pmatrix} 1 & -R_1F_{12} & -R_1F_{13} & . & . & -R_1F_{1n} \\ -R_2F_{21} & 1 & -R_2F_{23} & . & . & -R_2F_{2n} \\ -R_3F_{31} & -R_3F_{32} & 1 & . & . & -R_3F_{3n} \\ . & . & . & . & . & . \\ -R_nF_{n1} & -R_nF_{n2} & -R_nF_{n3} & . & . & 1 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ \\ E_n \end{pmatrix}$$

Where n is the number of patches in the scene. The solution is not so easy to find since the form factors also need to be calculated. Moreover, the matrix will be big; 50,000 by 50,000 can be typical.

When considering the computation of the form factors the specular reflection can cause problems. The difficulty is that unlike the diffuse reflection which is uniform, specular reflection is very much direction dependent and involves the relative directions of the viewpoint and each light source. But now, as we have noted, *every* patch is a light source! There will also be problems with specularities since all the light sources are no longer points, so we have to integrate incident light over a specular cone. This all means that computing specularities will be very difficult, so for the moment we will consider only diffuse radiosity.

As previously mentioned, we need to divide our graphics scene into patches for computing the radiosity. If our graphics scene consisted of small polygons we could perhaps use the polygon map as a set of radiosity patches and calculate a single radiosity value for each polygon. However some of the polygons may be too large, for example polygons that might make up a wall, and we will need to subdivide them to make the patches small enough. This is because the emitted light will not be constant across a large polygon, and if the patches that make up the polygon are too large we will see them as subdivisions of the polygon.

In normal circumstances large polygons may have shading differentials, or shadows thrown across them. Since we calculate just one radiosity value for each patch, so the patching pattern may form an unwanted visual artefact become visible. There are two ways to get round this:

1. Make the patches small enough to project to a pixel sized or sub-pixel sized region, or
2. Smooth the results (eg by interpolation similar to Gouraud shading).

The Form Factors

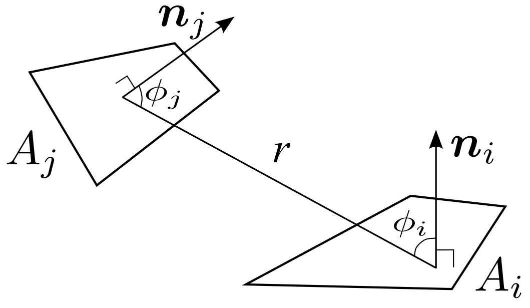


Figure 2: A form factor couples each pair of patches

The form factors link every pair of patches and determine the proportion of radiated energy from one that strikes the other. This is illustrated in Figure 2 and uses the following equation:

$$F_{ij} = \frac{1}{|A_i|} \int_{A_i} \int_{A_j} \frac{\cos\phi_i \;\cos\phi_j}{\pi r^2} dA_j dA_i$$

where $|A_i|$ is the area of patch A_i. The two cosine terms effectively compute the projection of the two patches in the direction normal to the line joining them.

If the patches are in the same plane, then there would be no light transmitted from one to the other. If they are separated and directly facing each other then they are maximally coupled. The $1/r^2$ is the normal inverse square law for the decay of light intensity over distance.

The equation can be simplified if we assume that the size of A_i is small in comparison with r. In this case we can consider the terms in the integral to be constant over A_i. Thus the outer integral evaluates to $|A_i|$ times the constant inner integral, and the equation reduces to:

$$F_{ij} = \int_{A_j} \frac{cos\phi_i \; cos\phi_j}{\pi r^2} dA_j$$

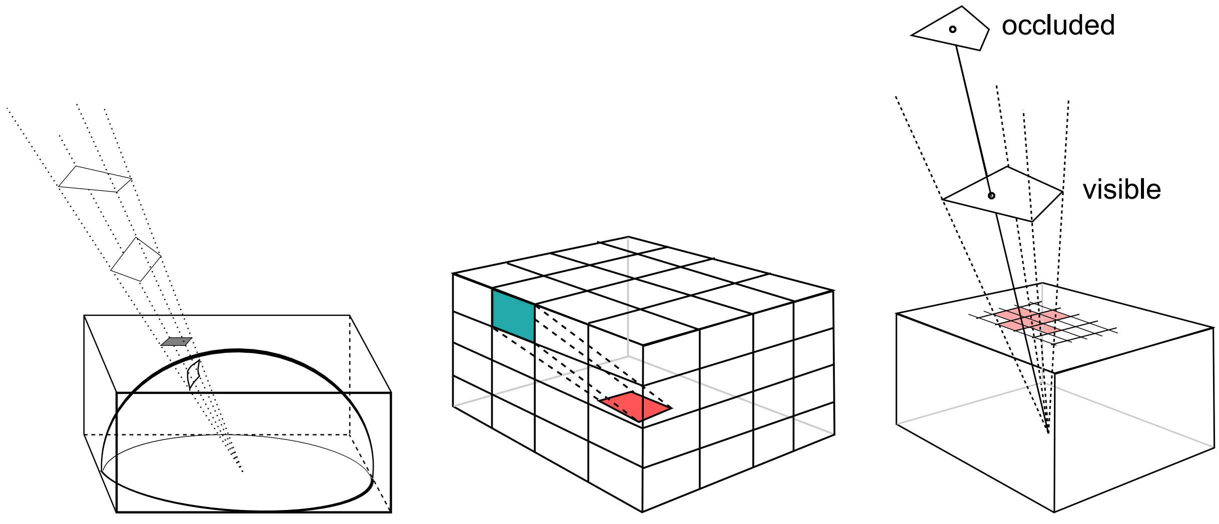
And, of course, we can make the same constancy assumption for patch A_j which leads to the approximate solution:

$$F_{ij} = \frac{cos\phi_i \; cos\phi_j |A_j|}{\pi r^2}$$

The Hemicube method

Although we have simplified the form factor equation, it would still be expensive to evaluate on a patch by patch basis. Accordingly, a fast algorithm was devised which makes the computation of form factors uniform. Using a bounding hemisphere it can be shown that all patches that project onto the same area of the hemisphere have the same form factor. This is illustrated by Figure 3(a), where all four patches have the same form factor. In particular, the algorithm uses the patches on the *hemicube* which is the half unit cube that bounds the hemisphere. The hemicube patch is shaded in the illustration in Figure 3(a).

A hemicube of side 1 unit is placed over the centre of a patch whose form factors are to be computed. Each of the five faces of the hemicube is divided regularly into a set of square patches called ‘hemicube pixels’. An example is given in Figure 3(b) where each face is divided into sixteen hemicube pixels. There is a trade off



(a) The Hemisphere and Hemicube (b) Delta form factors (c) Ray casting for radiosity

Figure 3: Calculating Radiosity with the hemicube

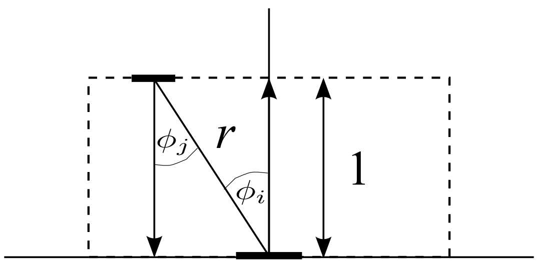


Figure 4: Computing the Delta Form Factors. A hemicube is shown in profile in which the bold line segments show the patch (bottom) and hemicube pixel (top).

here between speed and accuracy: The larger the size of the hemicube pixels, the worse the estimate of the form factors, but the faster the algorithm.

It will be observed that the form factors between the hemicube pixels and patch under consideration, called the *delta form factors*, will be the same whichever patch we are computing. Moreover, they will be simple to compute since the geometry is highly regular.

For example, suppose that the centre of a hemicube pixel shown in Figure 4 is at the coordinate $(x_p, y_p, 1)$. Thus a unit vector from the patch towards the origin can be written as

$$\frac{1}{r}(-x_p, -y_p, -1)^T \quad \text{where} \quad r = \sqrt{x_p^2 + y_p^2 + 1}$$

The unit surface normal to the hemicube pixel is $(0, 0, -1)^T$, and thus taking the dot product of the two vectors we find that $\cos\phi_i = 1/r$. Similar reasoning shows us that $\cos\phi_j = 1/r$. If the area of a hemicube pixel is ΔA, then its form factor is

$$\frac{\cos\phi_i \, \cos\phi_j \, \Delta A}{\pi \, r^2}$$

Thus the delta form factors on the top plane are given by

$$\frac{\Delta A}{\pi \, r^4}$$

By similar reasoning we can deduce that the form factors of the pixels on the vertical sides of the hemicube are given by

$$\frac{z_p \, \Delta A}{\pi \, r^4}$$

Values can therefore easily be computed once and stored for the hemicube pixels, and similarly a simple equation can be derived for the delta form factors of the sides of the hemicube.

We have previously noted that all patches that project onto the same area on the hemicube have the same form factor. Thus if a patch were to project exactly to a hemicube pixel, its form factor would be the same as the delta form factor for that hemicube pixel. If a patch projects to several hemicube pixels, its form factor will be simply the sum of the delta form factors of those hemicube pixels.

We can use this to find an approximate value for the form factors using a ray casting operation which is shown in figure 3(c). For each hemicube pixel we cast a ray through its centre and find the nearest intersection with another patch of the scene. We assume that this is the patch that projects entirely to that hemicube pixel, and all other patches are occluded by it. The smaller the hemicube pixels the more likely this is to be true, and the better the estimate of the form factors. The ray casting can be done using the techniques described in the ray tracing lecture.

We project a ray from the centre of the patch whose form factors we are computing, through the centre of each hemicube pixel, and out into the scene. We find the nearest patch that it intersects with. All the previously elaborated methods can be used to establish coherence and minimise the ray patch intersection calculations. Notice that all we need to determine is which patches are visible at each hemicube pixel. We do not need to generate any secondary rays after the nearest intersection has been found.

We can do do the same computation by the alternative means of polygon rendering. To do this we need to transform the scene. The origin of the transformed scene will be the centre of the patch that we are calculating, and, for the top face of the hemicube, the viewing direction will be through the centre of the face, vertically upwards in figure 3(c). Each patch vertex can then be projected onto the top plane with one matrix multiplication, and the pixels it projects to can be determined by a raster filling algorithm. We need to find the closest patch that projects to a hemicube pixel, and all others can be considered occluded. Essentially we have the same choices to make as we had when removing hidden parts when rendering a scene. We could make use of a z-buffer, and allocate a patch to a pixel only if it is closer than any other previous allocation. Alternatively we can use the painter's algorithm, and sort the patches by distance before projecting them onto the hemicube. The last patch to be allocated to a particular pixel displaces all others. When the allocation process is complete, the form factors with each patch of the scene are found by summing the delta form factors of the hemicube pixels to which they project. If a patch is not allocated to any pixel its form factor is zero, which is generally the case.

In summary, the radiosity method is as follows:

1. Divide the graphics world into discrete patches
2. Compute form factors by the hemicube method
3. Solve the matrix equation for the radiosity of each patch.
4. Average the radiosity values at the corners of each patch,

5a. Compute a texture map of each point on the patch (for walkthroughs), or

5b. Project to the viewing window and render with interpolation shading.

Radiosity Images

Much of the early work on radiosity was carried out at Cornell University, and images and tutorial material can be found on their web site.

http://www.graphics.cornell.edu/online/research/