Lecture 11: Radiosity

Earlier in the course we introduced the reflectance equation for modelling light reflected from surfaces:

$$I_{reflected} = k_a + I_i k_d \boldsymbol{n.s} + I_i K_s (\boldsymbol{r.v})^t$$

Where I is the incident light and the constants represent:

k_a the amount of ambient light
k_d the amount of diffuse reflection
k_s the amount of specular reflection

We used this lighting model for calculating shading values for polygons using both Phong and Gouraud shading. We used the same equations 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.

However, according to the reflectance equation, every surface in a graphics scene is emitting light. We have considered the emitted light travelling in the viewing direction, and neglected the emitted light travelling in other directions. This light 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:

$$I_{reflected} = I_i k_a + I_i k_d \boldsymbol{n.s} + I_i K_s (\boldsymbol{r.v})^t$$

or more simply to write (for a given viewpoint)

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

where R is the viewpoint dependant reflectance function.

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 the emitted energy (if any) and the reflected energy. For a small area of the surface dA (where the emitted energy can be regarded as constant) we have:

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

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 for patch i:

$$I_i = \int B_j F_{ij}\, dA_j$$

where the integral is taken over all patches except i, and F_{ij} is a constant that links patch i and patch j called the form factor. For computer graphics we cannot expect to compute a continuous solution, so we divide all polygons up into patches and replace the integral with a sum:

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

Where the sum is taken over all patches except i (or alternatively we can set $F_{ii} = 0$) If we can solve this for all Bi then we will be able to render each patch directly with a correct light model. We can formulate the problem as a matrix equation:

$$\begin{bmatrix} 1 & -R_1F_{12} & -R_1F_{13} & . & . & -R_1F_{1n} \\ -R_2F_{21} & 1 & -R_2F_{23} & . & . & -R_2F_{2n} \\ -R_3F_{31} & -R_3F_{32} & 1 & . & . & -R_3F_{3n} \\ .\ . & .\ . & .\ . & . & . & .\ . \\ -R_nF_{n1} & -R_nF_{n2} & -R_nF_{n3} & . & . & 1 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ .\ . \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \\ .\ . \\ E_n \end{bmatrix}$$

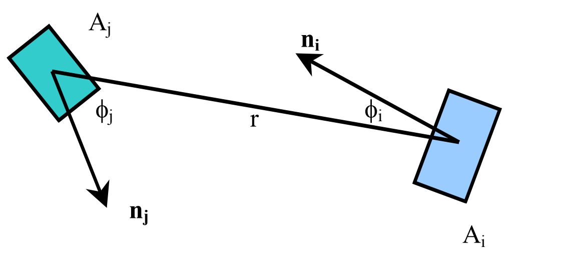
However, the solution is so easy to do since the form factors are not known. Moreover, the matrix will be big 10,000 by 10,000 may be typical. The radiosity values are wavelength dependent, hence we will need to compute a radiosity value for RG and B. The radiosity values are the values that the rendered pixels will receive.

When considering the form factors the specular reflection will be seen to cause problems. The difficulty is that unlike the diffuse reflection which is uniform, the specular reflection is very much direction dependent and involves the vector to the light source ***v***. But now, as we have noted, every patch is a light source! There will be problems with specularities as well since all the light sources are no longer points, so we have to integrate incident light over a specluar cone. All this 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 can perhaps use the polygon map as a set of radiosity patches, but for large polygons, such as might make up a wall, we need to subdivide to make the patches small enough. This is going to cause a problem, since the emitted light will not be constant across a large polygon we will see the subdivisions. This is because in normal circumstances large polygons will show shading differentials, or may have shadows thrown across them. As we will see, we will compute a single value for each patch, so the patching pattern will become visible. There are two ways to et round this:

Make patches small enough to project to (sub) pixel size

or Smooth the results (eg by interpolation like Gouraud shading)



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

Diagram 1: Form Factors

The form factors couple every pair of patches, determining the proportion of radiated energy from one that strikes the other. The general equation is shown in diagram 1. The two cos terms effectively compute the projection of the two patches in the direction normal to the line joining them. (If they were at right angles then there would be no light transmitted from on to the other. If they are facing each other then they are maximally coupled. The $1/r^2$ is the normal inverse square law. The equation can be simplified if we consider Ai to be small. If r is large, compared with the dimensions of Ai, the cos terms and the $1/r^2$ can be considered constant over A_i. Thus the outer integral evaluates to 1, 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 assumption for patch A_j. Thus this integrand is also treated as a constant, to give the approximate solution:

$$Fij = \cos\phi_i \cos\phi_j \, \mathrm{Area}(A_j)/\pi\, r^2$$

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 diagram 2, where all four patches have the same form factor. In particular, the patches on the hemicube are used in the algorithm.

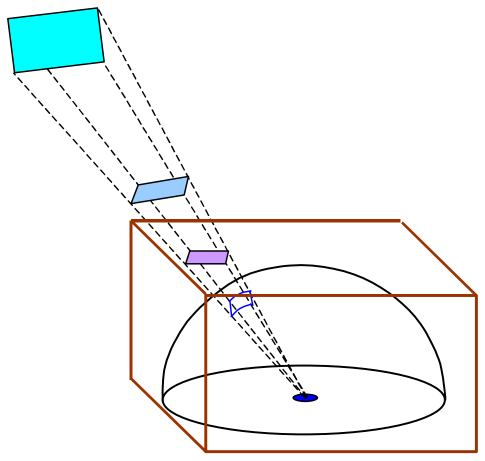


Diagram 2: The HemiSphere and Hemicube

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 "hemicube pixels". There will be a trade off 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, for the top face, where z=1, we will have that

$$\cos \phi_i = \cos \phi_j = 1/r$$

where r is the distance from the centre of the patch to the hemicube pixel. If the area of a hemicube pixel is ΔA, its form factor is:

$$\cos \phi_i \cos \phi_j \, \Delta A / \pi r^2$$

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

$$\Delta A / \pi r^4$$

Values can easily be computed and stored for these pixels.

We have previously noted that all patches that project onto the same area of a surrounding hemicube have the same form factor. Thus all patches that project to a hemicube pixel will have the form factor calculated for that pixel. If a patch projects to several hemicube pixels, its form factor will be simply the sum of the form factors of those hemicube pixels.

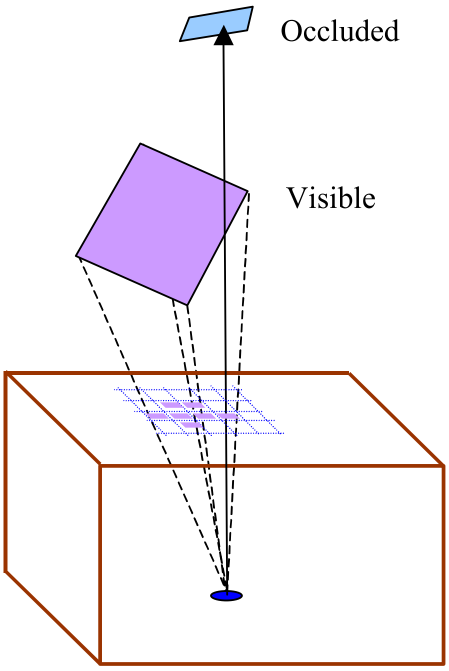


Diagram 3: Finding visible patches

One of the justifications for choosing the hemicube, as opposed to the hemisphere is that the computation of the projection of the patch onto the plane(s) is straightforward. We need to develop a viewing transformation matrix with the origin (viewpoint) at the centre of the patch, and the viewing direction (z) in the normal direction, and the projection plane at z=1. 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. However, if we take this approach we need to solve the occlusion problem. We need to find the closest patch that projects to a pixel. 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 of the patches are found by summing the form factors of the pixels to which they have been allocated. If a patch is not allocated to any pixel its form factor is zero.

An alternative to the process is to use a modified ray tracing algorithm. For each pixel we project a ray back into the scene and find the nearest patch that it intersects with. Again 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.

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/