

Lecture 14: Computational Issues in Radiosity

The radiosity method has stages that require intensive use of CPU time, and involve approximate methods that can lead to unwanted visual artefacts. The problems that are associated with each stage of the method are summarised as follows:

1. Divide the graphics world into discrete patches
Meshing strategies, meshing errors
2. Compute form factors by the hemicube method
Alias errors
3. Solve the matrix equation for the radiosity of each patch.
Computational strategies
4. Average the radiosity values at the corners of each patch
Interpolation approximations
5. Compute a texture map of each point or render directly
At least this stage is relatively easy

Alias Errors

Computation of the form factors will involve alias errors. These are equivalent to the errors that occur in texture mapping due to discrete sampling of a continuous environment. However, these errors are perhaps the least of the problems we will encounter. Errors in the form factors are a secondary effect, as each patch radiosity will be determined by a large number of other patches. Thus each form factor will make only a small contribution to the result. The alias errors can be reduced, if necessary, by increasing the sampling of the hemi-cube, but this in turn increases the computational demand.

Form Factor Computation

Form factors have a reciprocal relationship:

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

so

$$F_{ji} = \frac{F_{ij}|A_i|}{|A_j|}$$

Thus, providing we can store the form factors, only half the patches need be computed.

However, the number of form factors will be very large, and in early solutions for radiosity it was not possible to pre-compute and save them. To give an idea of the size of the problem, we note that for 60,000 patches, there are 3,600,000,000 form factors. We only need to store half of these (reciprocity), but we will need four bytes for each, hence 7 GB is needed. The memory requirement increases with the square of the number of patches.

As many of the form factors are zero we can save space by using an indexing scheme. An index could be created with one bit per form factor. If a bit is zero this implies that the form factor is zero and not stored. All non zero form factors can be stored in a one dimensional array. Using such a scheme reduces the storage requirement, possibly to one quarter. However, it is easy to conceive of scenes where it is not possible to pre-compute and store the form factors.

Matrix Inversion

Inverting the matrix can be done using the Gauss Seidel method. This is not really a matrix method as such, but rather an iterative scheme based on the form factor equations. Each patch equation has the form:

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

We make successive estimates for B_i , indicated by superscripts: B_i^0, B_i^1, \dots and use the iteration

$$B_i^k = E_i + R_i \sum_j B_j^{k-1} F_{ij}$$

The initial values, B_i^0 , may be set to zero. At the first iteration the emitted light energy is distributed to those patches that are illuminated, in the next cycle, those patches illuminate others and so on. The Gauss Seidel inversion is stable and converges. A sufficient condition to guarantee the convergence is for the matrix to be 'diagonally dominant' and it can be shown that the radiosity matrix has this property.

Progressive refinement

The nature of the Gauss Seidel method allows a partial solution to be rendered as the computation proceeds. Without altering the method, we could render the image after each iteration, allowing the designer to stop the process and make corrections quickly. This may be particularly important if the scene is so large that we need to re-calculate the form factors every time we need them.

If the image is progressively refined in this way, it will start dark and gradually illuminate as the iterations proceed. For this reason it is sometimes thought advantageous to add in a constant ambient term which is reduced as each iterative epoch is completed.

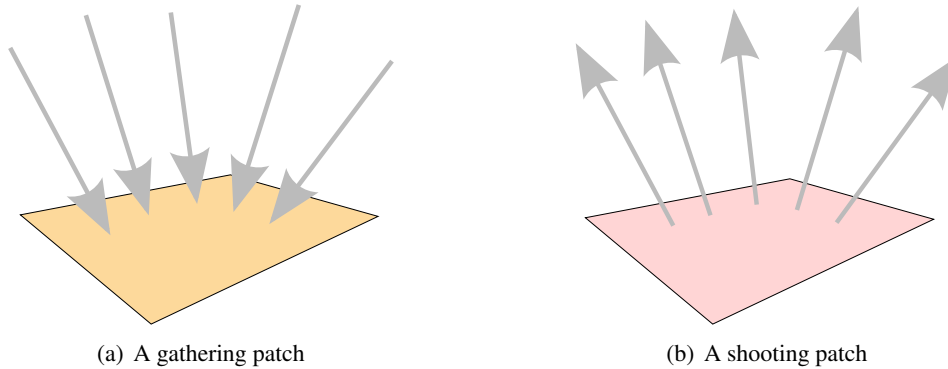


Figure 1: Gathering and shooting

The Gauss Seidel inversion can be modified to make it faster by making use of the fact that it is essentially distributing energy around the scene. By choosing a good order of evaluation we can improve the quality of a partial solution and reduce the number iterations required for convergence.

The evaluation of one B_i value using one row of the matrix in the normal Gauss Seidel way is like a process of *gathering*. The value of B_i for the current iteration is estimated from the B_j values of all the other patches from the previous iteration:

$$B_i^k = E_i + R_i \sum_j B_j^{k-1} F_{ij}$$

Suppose, in one iteration, B_i changes by ΔB_i (i.e. $B_i^k = B_i^{k-1} + \Delta B_i$). Every other patch in the scene will change (providing its form factor is not zero). The magnitude of this change can be written:

$$\Delta B_j = R_j F_{ji} \Delta B_i$$

Which can be simply changed into an iterative updating scheme:

$$B_j^k = B_j^{k-1} + R_j F_{ji} \Delta B_i^{k-1}$$

This is the process of *shooting*, and is equivalent to evaluating the matrix column wise. The use of shooting allows us to choose an evaluation order that ensures fastest convergence. The patches with the largest change ΔB (called the unshot radiosity) are distributed first. In an iteration, the unshot radiosity of a chosen patch is reduced to zero, that of other patches is incrementally increased, then we iterate again. The process starts by shooting the emitting patches, since at the first iteration $\Delta B_i = E_i$.

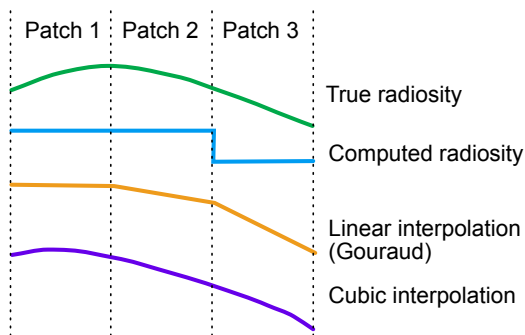


Figure 2: Interpolation Strategies

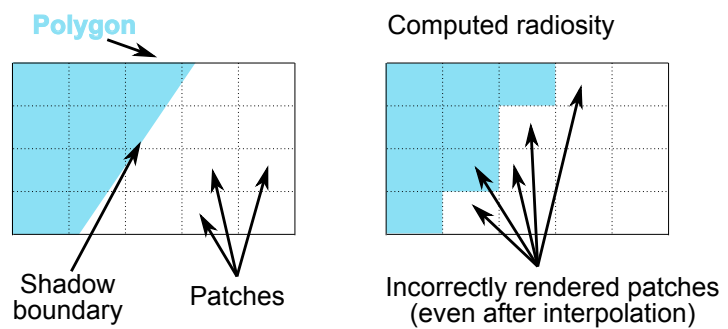


Figure 3: D^0 Meshing Problems

Interpolation Strategies

Visual artefacts do occur with interpolation strategies, but may not be significant for small patches. Figure 2 illustrates the different interpolation strategies and the results they will produce. As noted previously the computed values alone will produce undesirable visual artefacts. However, the use of a cubic interpolation scheme, though more accurate than a purely linear scheme, will probably not produce any significant improvement.

Meshing

Meshing is the process of dividing the scene into patches, and causes the worst problems in terms of visual quality. Meshing artefacts are scene dependent. The most obvious are called D^0 artefacts, and are caused by discontinuities in the radiosity function. These could be due to shadows or to surface albedo (ratio of incident to reflected light) or texture. Discontinuities of this kind cause problems when accompanied by bad patching. Figure 3 illustrates the problem. Here a single polygon is divided into regular square patches. A shadow boundary crosses and the result is a serious visual alias in the computed radiosity. Interpolation will reduce the visual effect of the alias, but will smooth out the hard edge of the shadow.

Discontinuity Meshing (a priori)

The idea behind discontinuity meshing is to compute discontinuities in advance, and align the patches with them. Additionally, the discontinuities are made known to the interpolation procedure so that hard boundaries are not smoothed out. Many of the discontinuities are scene dependent. These in particular will include object boundaries, which we would naturally expect to be patch boundaries as well, and albedo discontinuities, possibly specified in texture maps. Shadows can also be determined, and this can be done using a process similar to ray tracing. Books contain a lot of algorithmic detail on this subject, but we will not cover it here.

Adaptive Meshing (a posteriori)

The idea of adaptive meshing is to recompute the mesh as the radiosity calculation proceeds. Places where there are strong discontinuities in the radiosity can be found by comparing values at adjacent patches. There are essentially two approaches:

1. Put more patches (elements) into areas with high discontinuity or
2. Move the mesh boundary to coincide with the greatest change.

Subdivision of Patches (h refinement)

Using the *h-refinement* method we compute the radiosity at the vertices of the coarse grid of patches, as shown in Figure 4. If the discontinuities exceed a threshold we subdivide the patch into elements, and recompute in those elements. They can be checked for discontinuity and subdivided further.

When a patch is divided into elements, the radiosity of each element is computed using the original radiosity solution for all other patches. The assumptions behind this are that:

1. The radiosity of a patch is equal to the sum of the radiosity of its elements, and,
2. the distribution of radiosities among elements of a patch do not affect the global solution significantly.

These assumptions are a reasonable, and they save re-computing the complete radiosity solution after each subdivision.

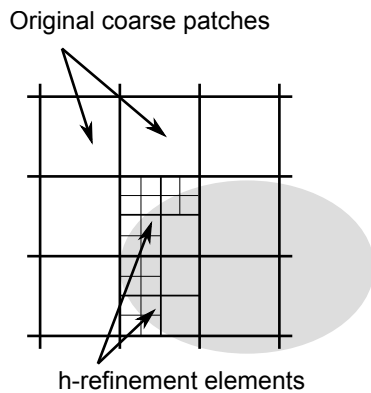


Figure 4: Adaptive Meshing using h-refinement

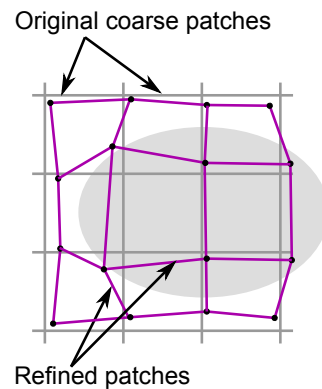


Figure 5: Adaptive Meshing using r-refinement

Patch Refinement (r refinement)

Here we compute the radiosity at the vertices of the coarse grid, and move the patch boundaries closer together if they have high radiosity changes. Unlike the h-refinement solution it is necessary to recompute the entire radiosity solution each refinement. However the method should make more efficient use of patches by shaping them correctly. Hence a smaller number of patches could be used. In Figure 5 we see that the uniform patches tend to get larger, and the ones around the discontinuities tend to get smaller, hence some other forms of re-patching may be required as the algorithm proceeds.

Adding Specularities

We noted that specularities (being viewpoint dependent) cannot be calculated by the standard radiosity method. However, they do form an important aspect of visual realism, and it is desirable to add them where possible. This can be done after the radiosity solution has been obtained, by means of ray tracing. The complete ray tracing solution is not required, just the specular component in the viewpoint direction. Thus only one primary ray per pixel and one secondary ray in the reflected direction are required to achieve this. Further ray tracing can be used to enhance radiosity solutions with effects such as reflections and translucency.