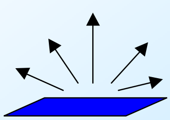
Interactive Computer Graphics

Lecture 12: Radiosity - Computational Issues

The story so far

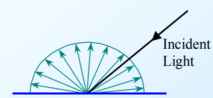
Every polygon in a graphics scene radiates light.

The light energy it radiates per unit area is called the RADIOSITY and denoted by letter B



Lambertian Surfaces

A lambertian surface is one that obeys Lambert's Cosine law. Its reflected energy is the same in all directions.



Perfectly Matt surface
The reflected intensity is the same in all directions

We can only calculate Radiosity for Lambertian Surfaces

The Radiosity Equation

For patch i $B_i = E_i + R_i \Sigma B_j F_{ij}$

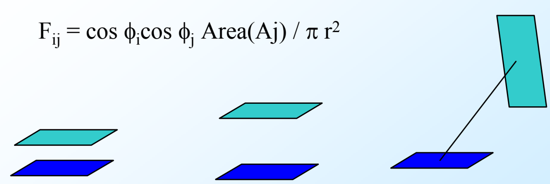
Ei is the light emitted by the patch (usually zero)

$R_i \Sigma B_j F_{ij}$ is the Reflectance*Light energy arriving from all other patches

F_{ij} is the proportion of energy leaving patch j that reaches patch i

Form Factors Fij

$F_{ij} = \cos \phi_i \cos \phi_j \text{ Area(Aj)} / \pi r^2$



Big form factor perhaps 0.5

Further away thus smaller form factor perhaps 0.25

Not facing each other thus even smaller form factor perhaps 0.1

Computing the Form Factors

Direct Computation

- 20,000 polygons (or patches)
- 400,000,000 form factors

Computation takes forever - most of the results will be zero.

Hemicube method

Pre-compute the form factors on a hemicube

For each patch ray trace through the hemicube

The whole solution

All that remains to be done is to solve the matrix equation:

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

Summary of the Radiosity Method

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

Now read on . . .

Alias Errors

Computation of the form factors will involve alias errors.

This is equivalent to errors in texture mapping, due to discrete sampling of a continuous environment.

However, as the alias errors are averaged over a large number of pixels the errors will not be significant.

Form Factor reciprocity

Form factors have a reciprocal relationship:

$$F_{ij} = \cos\phi_i \cos\phi_j \, \text{Area}(Aj) / \pi r^2$$
$$F_{ji} = \cos\phi_i \cos\phi_j \, \text{Area}(Ai) / \pi r^2$$
$$F_{ji} = F_{ij} * \text{Area}(Ai) / \text{Area}(Aj)$$

Thus only half the patches need be computed.

The number of form factors

There will be a large number of form factors:

for 20,000 patches, there are 400,000,000 form factors. We only need store half of these (reciprocity), but we will need four bytes for each, hence 800Mbytes are needed.

As many of them are zero we can save space by using an indexing scheme. (eg use one bit per form factor, bit = 0 implies form factor zero and not stored)

Inverting the matrix

Inverting the matrix can be done by the Gauss Siedel method:

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

Each row of the matrix provides an equation of the form

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

Inverting the matrix

Gauss Siedel formulates an iterative method using the equation of each row

Given:

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

We use the iteration:

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

The initial values B_i^0 may be set to zero

Gauss-Siedel method for solving equations

Given a scene with three patches:

$B_0 \leftarrow E_0 + R_0 (F_{01} B_1 + F_{02} B_2)$

$B_1 \leftarrow E_1 + R_1 (F_{10} B_0 + F_{12} B_2)$

$B_2 \leftarrow E_2 + R_2 (F_{20} B_0 + F_{21} B_1)$

and suppose we have numeric values

$B'_0 \leftarrow 0 + 0.5 (0.2 B_1 + 0.1 B_2) = 0.1 B_1 + 0.05 B_2$

$B'_1 \leftarrow 5 + 0.5 (0.2 B_0 + 0.3 B_2) = 5 + 0.1 B_0 + 0.15 B_2$

$B'_2 \leftarrow 0 + 0.2 (0.1 B_0 + 0.3 B_1) = 0.02 B_0 + 0.06 B_1$

Gauss-Siedel example - continued

$B_0 \leftarrow 0.1 B_1 + 0.05 B_2$

$B_1 \leftarrow 5 + 0.1 B_0 + 0.15 B_2$

$B_2 \leftarrow 0.02 B_0 + 0.06 B_1$

Substitute first estimate $B_0=0$; $B_1=0$; $B_2=0$ in RHS

Compute next estimate $B_0=0$; $B_1=5$; $B_2=0$

Substitute estimate $B_0=0$; $B_1=5$; $B_2=0$ in RHS

Compute next estimate $B_0=0.5$; $B_1=5$; $B_2=0.3$

Gauss-Siedel example - concluded

$B_0 = 0.1 B_1 + 0.05 B_2$

$B_1 = 5 + 0.1 B_0 + 0.15 B_2$

$B_2 = 0.02 B_0 + 0.06 B_1$

Substitute estimate $B_0=0.5$; $B_1=5$; $B_2=0.3$ in RHS

Compute next estimate $B_0=0.515$; $B_1=5.095$; $B_2=0.31$

Process eventually converges in this case

Inverting the Matrix

The Gauss Siedel inversion is stable and converges fast since the Ei terms are constant and correct at every iteration, and all Bi values are positive

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 image will start dark and progressively illuminate as the iteration proceeds

Progressive Refinement

The nature of the Gauss Siedel 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.

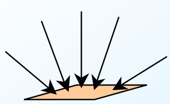
This may be particularly important if the scene is so large that we need to re-calculate the form factors every time we need them.

Inverting the matrix

The Gauss Siedel inversion can be modified to make it faster by making use of the fact that it is essentially distributing energy around the scene.

The method is based on the idea of "shooting and gathering", and also provides visual enhancement of the partial solution.

Gathering Patches



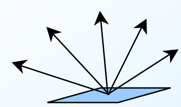
Gathering Patch

Evaluation of one B value using one line of the matrix:

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

is the process of gathering.

Shooting Patches



Shooting Patch

Suppose in an iteration Bi changes by ΔBi. The change to every other patch is found using:

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

This is the process of shooting, and is evaluating the matrix column wise.

Evaluation Order

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 radiosiy) are evaluated first.

The process starts by shooting the emitting patches, since at the first iteration ΔBi = Ei

Processing Unshot Radiosity

Patch	Unshot Radiosity
B_0	ΔB_0
B_1	ΔB_1
B_2	ΔB_2
B_n	ΔB_n

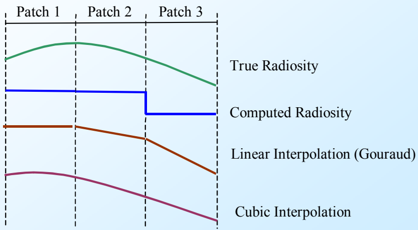
Choose patch with largest unshot radiosity ΔBi

Shoot the radiosity, ie for all other patches calculate $R_j F_{ji} \Delta B_i$ and add to the unshot radiosity

Add ΔBi to Bi, Set ΔBi = 0 and iterate

Interpolation Strategies

Visual artefacts do occur with interpolation strategies, but may not be significant for small patches



Meshing

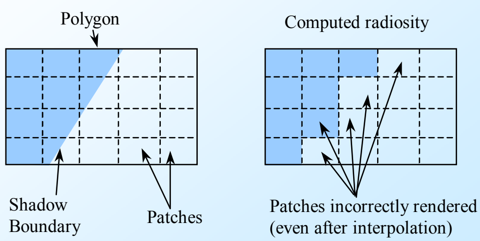
Meshing is the process of dividing the scene into patches.

Meshing artifacts are scene dependent.

The most obvious are called D^0 artifacts, caused by discontinuities in the radiosity function

D^o Artefacts

Discontinuities in the radiosity are exacerbated by bad patching



Discontinuity Meshing (a priori)

The idea is to compute discontinuities in advance:

eg

- Object Boundaries
- Albedo discontinuities (in texture)
- Shadows (requires pre-processing by ray tracing)
- etc

Adaptive Meshing (a posteriori)

The idea is to re-compute the mesh as part of the radiosity calculation:

eg If two adjacent patches have a strong discontinuity in radiosity value, we:

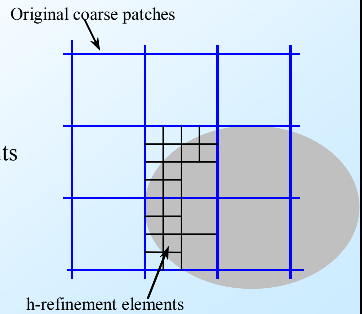
(i) put more patches (elements) into that area, or

(ii) move the mesh boundary to coincide with the greatest change

Subdivision of Patches (h refinement)

Compute the radiosity at the vertices of the coarse grid.

Subdivide into elements if the discontinuities exceed a threshold



Computational issues of h-refinement

When a patch is divided into elements each element radiosity is computed using the original radiosity solution for all other patches.

The assumptions are that

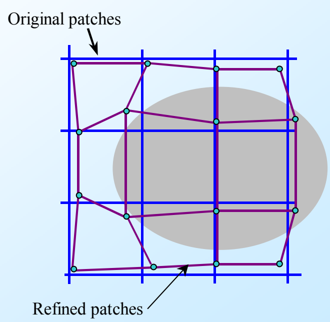
(i) the radiosity of a patch is equal to the sum of the radiosity of its elements, and,

(ii) the distribution of radiosities among elements of a patch do not affect the global solution significantly

Patch Refinement (r refinement)

Compute the radiosity at the vertices of the coarse grid.

Move the patch boundaries closer together if they have high radiosity changes



Patch refinement

Unlike the other solution it would be necessary to re-compute 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.

Adding Specularities

We noted that specularities (being viewpoint dependent) cannot be calculated by the standard radiosity method.

However, they could be added later by ray tracing.

The complete ray tracing solution is not required, just the specular component in the viewpoint direction