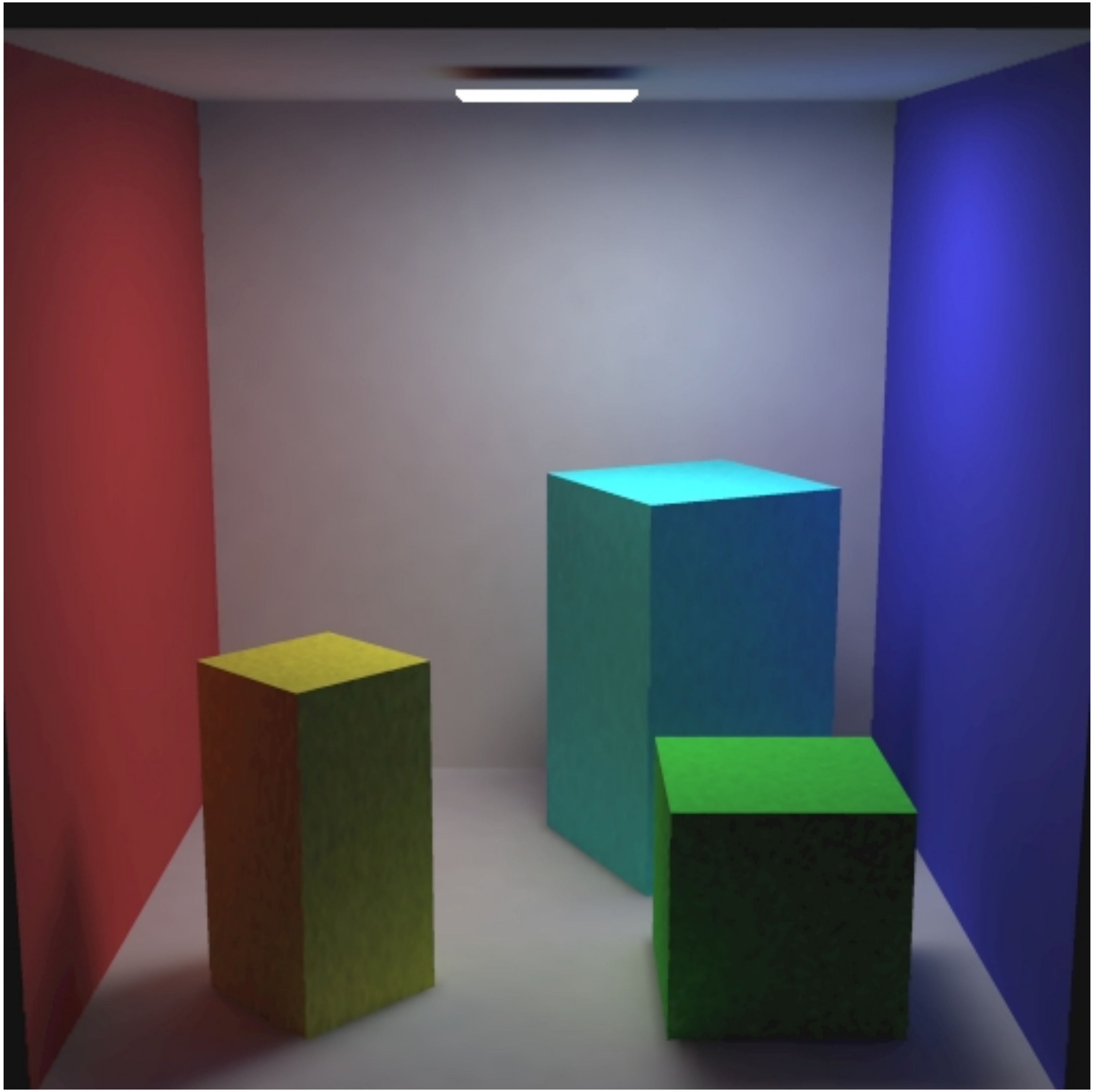
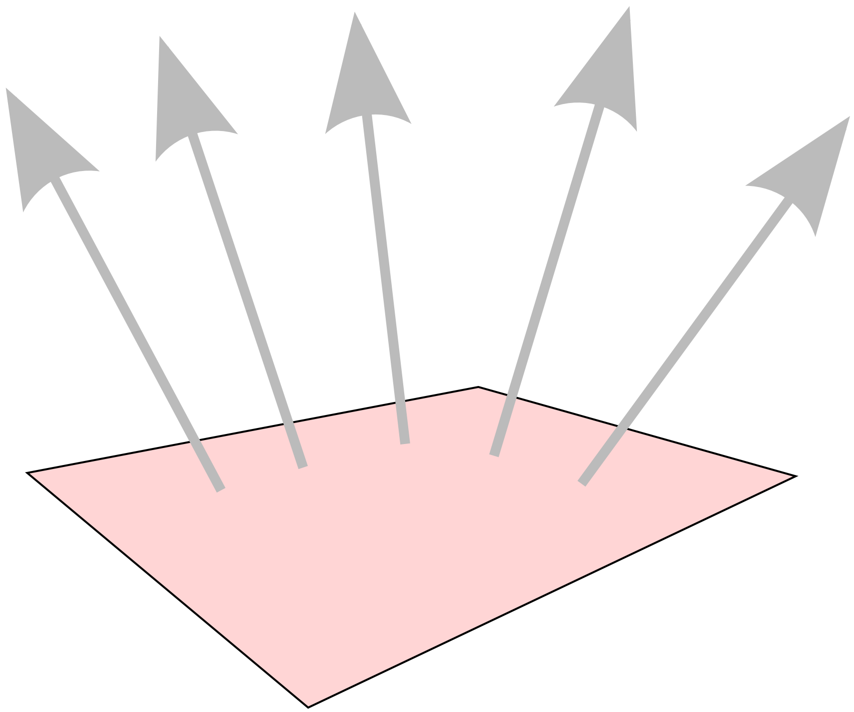
Interactive Computer Graphics: Lecture 14

Computational Issues in Radiosity



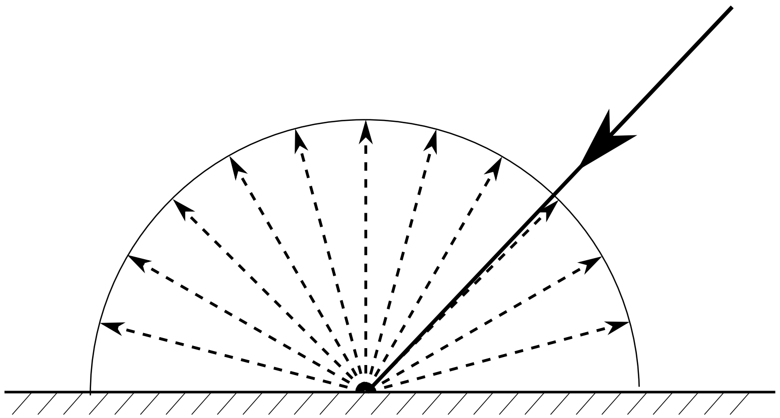
The story so far …

- Every polygon in a graphics scene radiates light.
- The light energy it radiates is called the *radiosity*, denoted by letter B



Lambertian Surfaces

- A Lambertian surface is one that obeys Lambert's Cosine law.
- It is a perfectly matt surface and the reflected energy is the same in all directions.



- We can only calculate Radiosity for Lambertian Surfaces

The Radiosity Equation

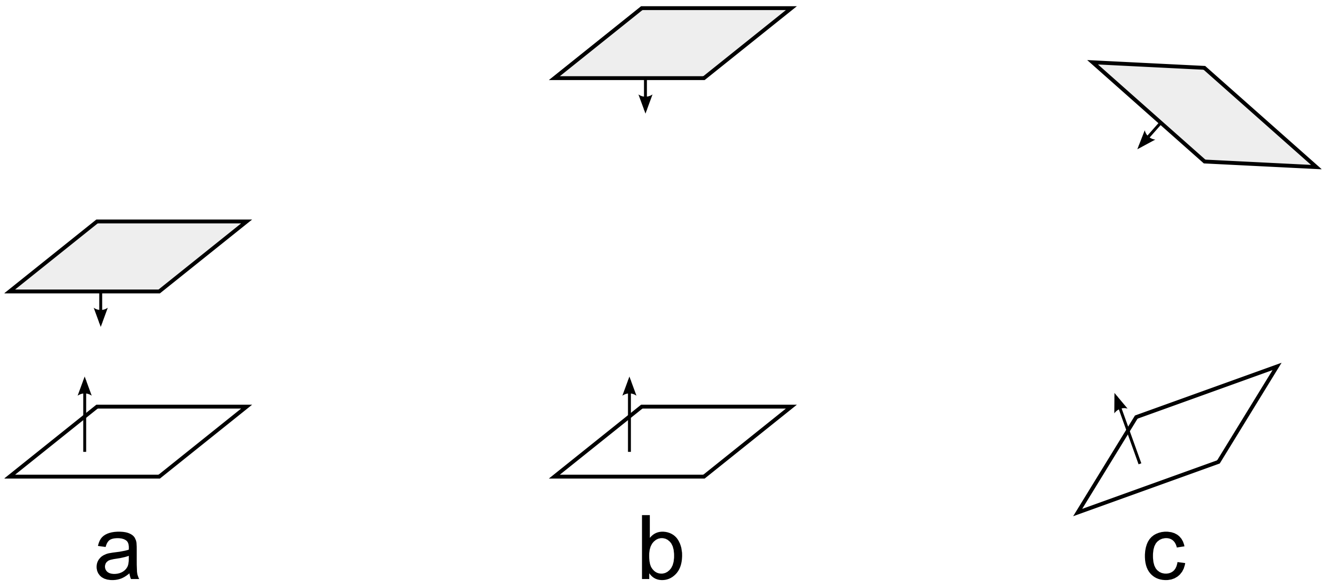
- For patch i:

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

E_i	Light emitted by the patch (usually zero)
$R_i \sum_j B_j F_{ij}$	Reflected light energy that arrived from all other patches
F_{ij}	Proportion of energy leaving patch j that reaches patch i

Form factors

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



(a) Big form factor perhaps 0.5

(b) Further away, smaller form factor, perhaps 0.25

(c) Not really facing each other, even smaller form factor perhaps 0.1

Computing the Form Factors

- Direct Computation:
  - 60,000 polygons (or patches)
  - 3,600,000,000 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 (or project) through the hemicube

The whole solution

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

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

Summary of Radiosity method

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
5. Compute a texture map of each point or render directly

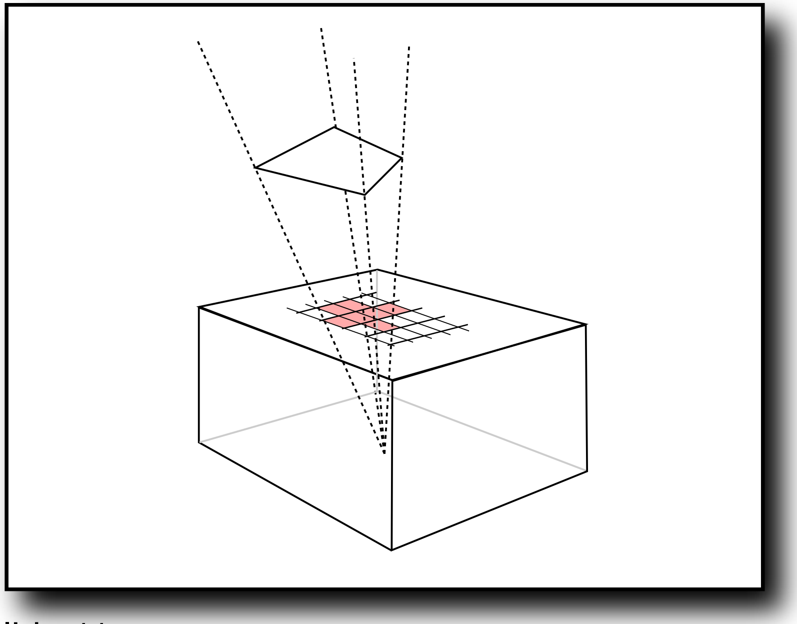
Summary of Radiosity method

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
   At least this stage is relatively easy

Now read on ...

Alias Errors

- Computation of the form factors will involve alias errors.
- 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:

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

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

- So form factors for only half the patches need be computed.

The number of form factors

There will be a large number of form factors:

For 60,000 patches, there are 3,600,000,000 form factors.

We only need store half of these (reciprocity), but we will need four bytes for each, hence 7 GB are needed.

As many of them are zero we can save space by using an indexing scheme (e.g. 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 Seidel method:

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

- Each row of the matrix gives an equation of the form:

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

Inverting the matrix

- The Gauss Seidel method is iterative and uses the equation of each row
- Given:

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

- We use the iteration:

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

- To give successive estimates $B_i^0, B_i^1, \ldots$
- Can set initial values $B_i^0 = 0$

Gauss-Seidel method for solving equations

- Given a scene with three patches, we can write the iterations as *update* equations:

$$B_0 \leftarrow E_0 + R_0(\, F_{01}\, B_1 + F_{02}\, B_2\,)$$
$$B_1 \leftarrow E_1 + R_1(\, F_{10}\, B_0 + F_{12}\, B_2\,)$$
$$B_2 \leftarrow E_2 + R_2(\, F_{20}\, B_0 + F_{21}\, B_1\,)$$

- Assume we know numeric the values for E_0, E_1, E_2, R_0, $R1$, R_2, F_{01}, F_{02}, F_{10}, F_{12}, F_{20}, F_{21}:

$$B_0 \leftarrow 0 + 0.5(\, 0.2\, B_1 + 0.1\, B_2\,)$$
$$B_1 \leftarrow 5 + 0.5(\, 0.2\, B_0 + 0.3\, B_2\,)$$
$$B_2 \leftarrow 0 + 0.2(\, 0.1\, B_0 + 0.3\, B_1\,)$$

Gauss-Seidel method for solving equations

Simplify:

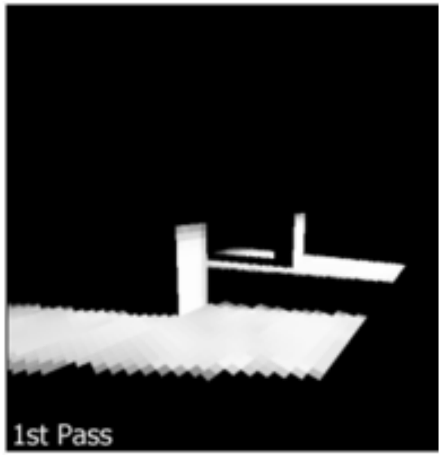
$$B_0 \leftarrow 0.1\,B_1 + 0.05\,B_2$$
$$B_1 \leftarrow 5 + 0.1\,B_0 + 0.15\,B_2$$
$$B_2 \leftarrow 0.02\,B_0 + 0.06\,B_1$$

Step	B_0	B_1	B_2
0	0	0	0
1	0	5	0
2	0.5	5	0.3
3	0.515	5.095	0.31
⋮	⋮	⋮	⋮

The process eventually converges to 0.53, 5.07 and 0.31 in this case

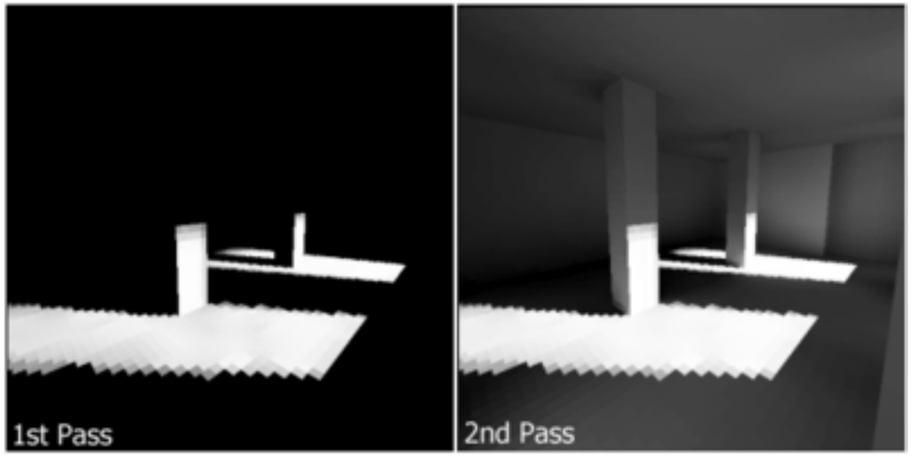
Gauss-Seidel method for solving equations

- The Gauss-Seidel method is stable and converges
- It can be shown that the radiosity matrix is 'diagonally dominant' (a sufficient condition to guarantee convergence).
- 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



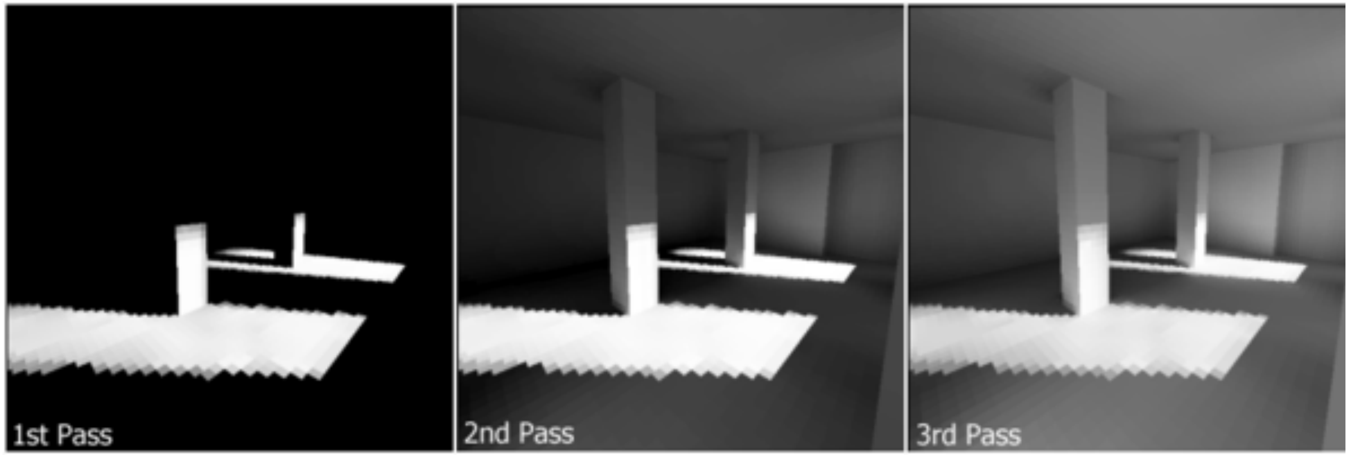
Gauss-Seidel method for solving equations

- The Gauss-Seidel method is stable and converges
- It can be shown that the radiosity matrix is 'diagonally dominant' (a sufficient condition to guarantee convergence).
- 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



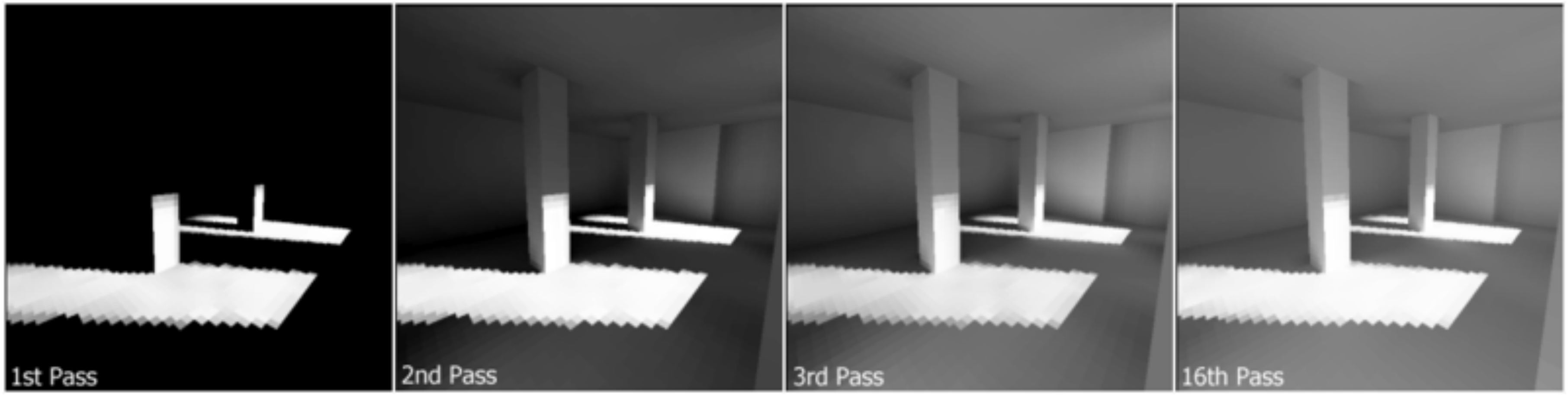
Gauss-Seidel method for solving equations

- The Gauss-Seidel method is stable and converges
- It can be shown that the radiosity matrix is 'diagonally dominant' (a sufficient condition to guarantee convergence).
- 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



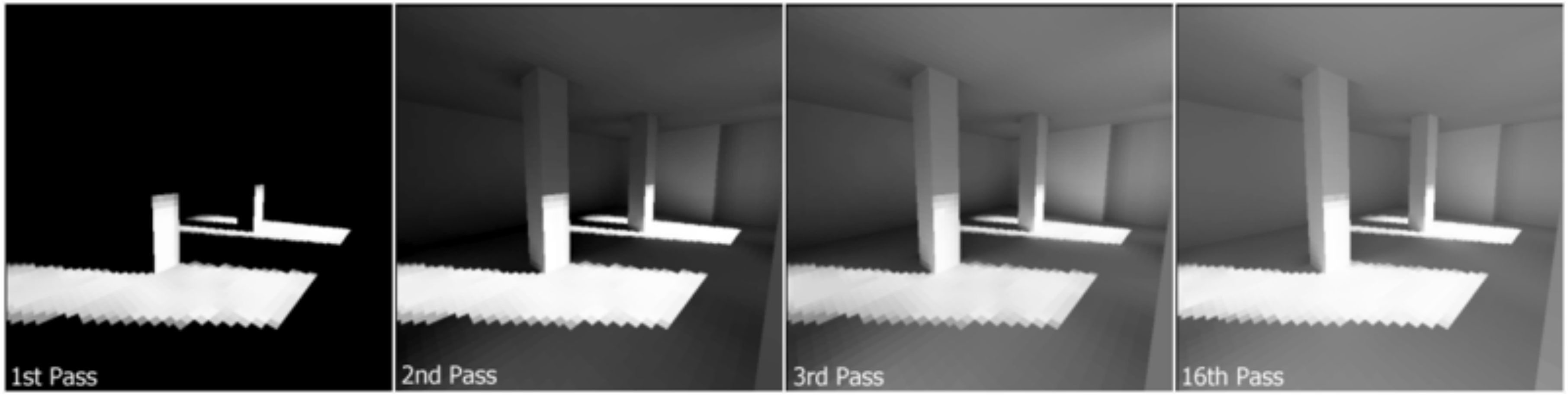
Gauss-Seidel method for solving equations

- The Gauss-Seidel method is stable and converges
- It can be shown that the radiosity matrix is ‘diagonally dominant’ (a sufficient condition to guarantee convergence).
- 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 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.



Inverting the matrix

- 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.

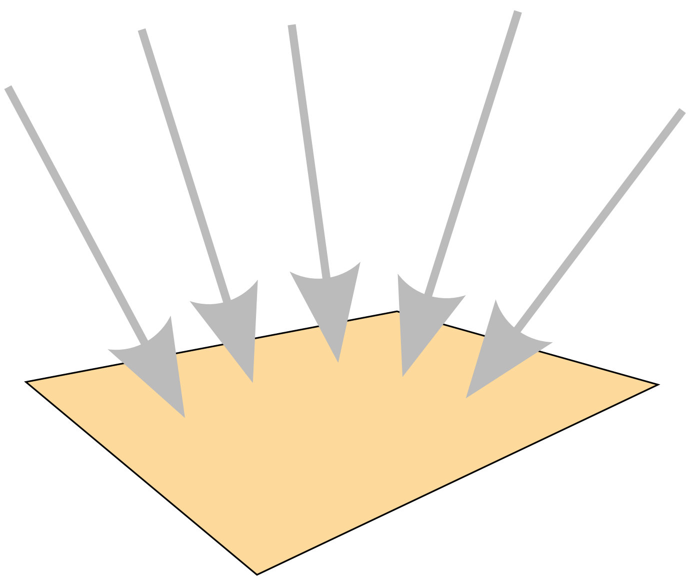
- The method is based on the idea of “shooting and gathering”, and also provides visual enhancement of the partial solution.

Gathering Patches

- Evaluation of one B_i value using one line of the matrix:

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

is the process of *gathering*.

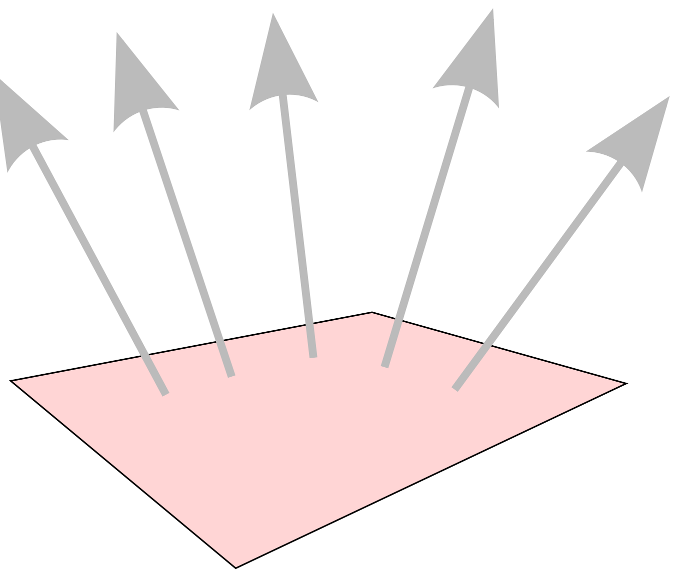


Shooting Patches

- Suppose in an iteration B_i changes by ΔB_i
- The change to every other patch can be 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 idea of gathering and shooting allows us to choose an evaluation order that ensures fastest convergence.
- The patches with the largest change ΔB (called the unshot radiosity) are evaluated first.
- The process starts by initialising all unshot radiosity to zero except emitting patches where $\Delta B_i = E_i$

Processing unshot radiosity

Patch	Unshot radiosity
B_0	ΔB_0
B_1	ΔB_1
B_2	ΔB_2
$\vdots$	$\vdots$
B_N	ΔB_N

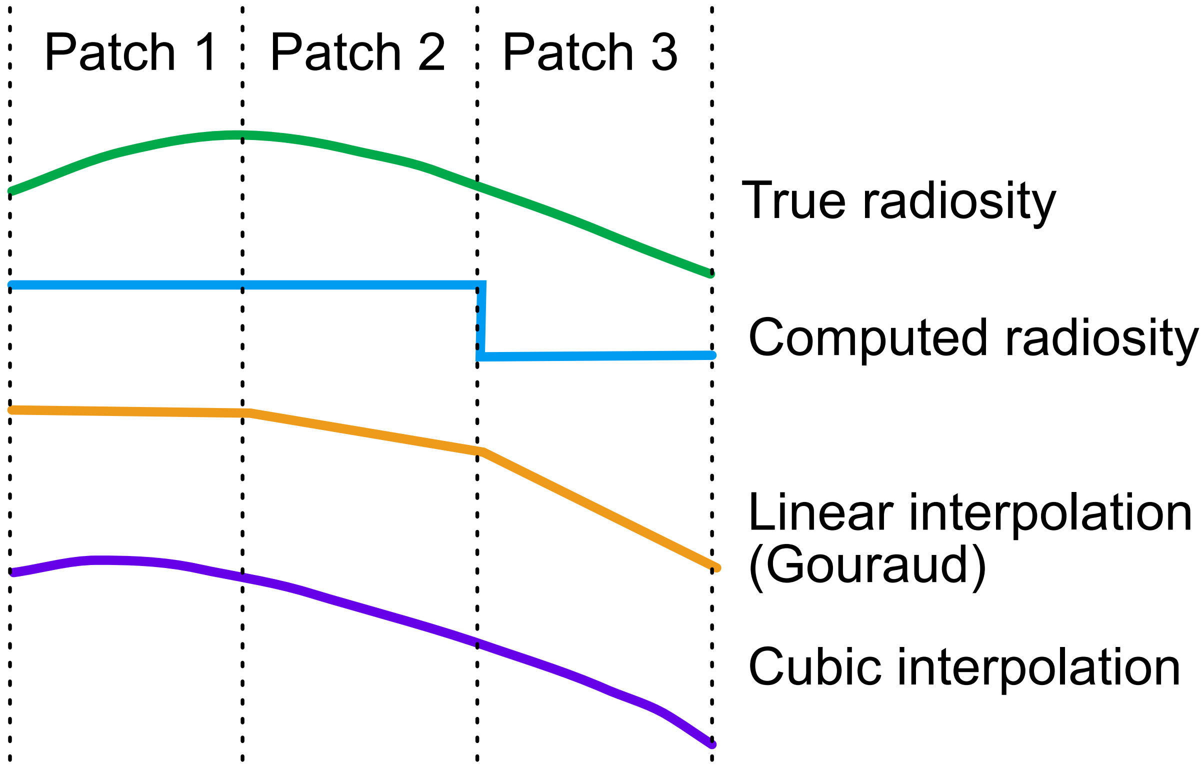
- Choose patch with largest unshot radiosity ΔB_i
- Shoot the radiosity for the chosen patch, i.e. for all other patches update

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

- and add it to their radiosity
- Set $\Delta B_i = 0$ and iterate

Interpolation Strategies

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



Meshing

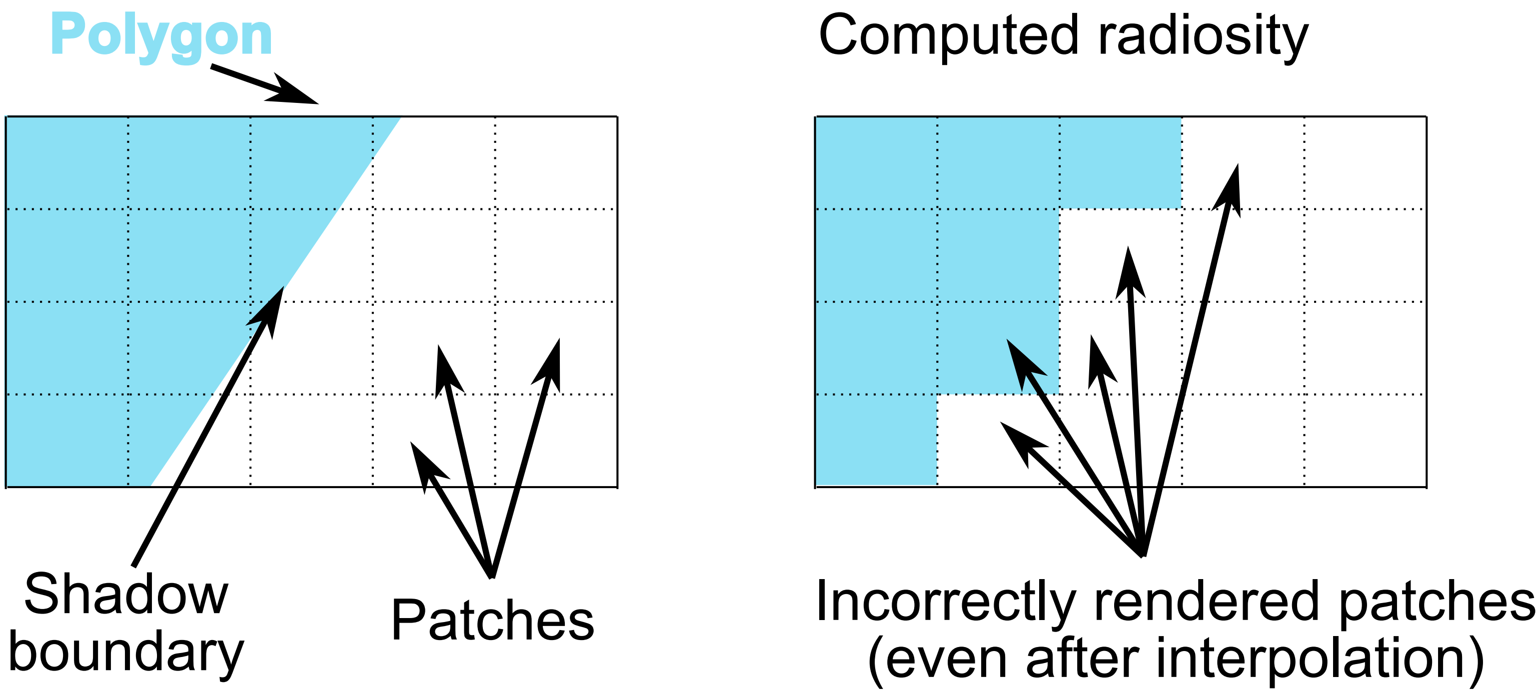
- Meshing is the process of dividing the scene into patches.

- Meshing artifacts are scene dependent.

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

D^0 *artifacts*

- Discontinuities in the radiosity are exacerbated by bad patching



Discontinuity Meshing (a-priori)

- The idea is to compute discontinuities in advance:
  - Object boundaries
  - Albedo/reflectivity discontinuities
  - Shadows (requires pre-processing by ray tracing)
  - etc.

- Place patches in advance so that they align with the discontinuities

- Then calculate radiosity

Adaptive Meshing (a posteriori)

The idea is to re-compute the mesh during the radiosity calculation

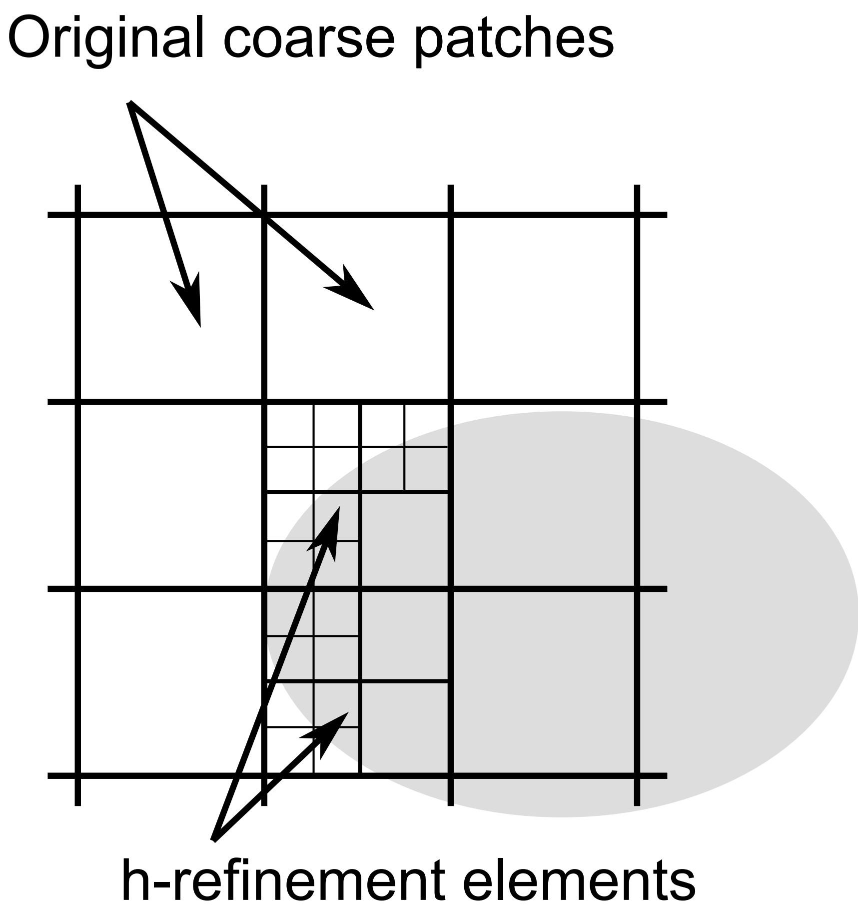
If two adjacent patches have a strong discontinuity in radiosity value, we can

1. Put more patches (elements) into that area, or
2. 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.

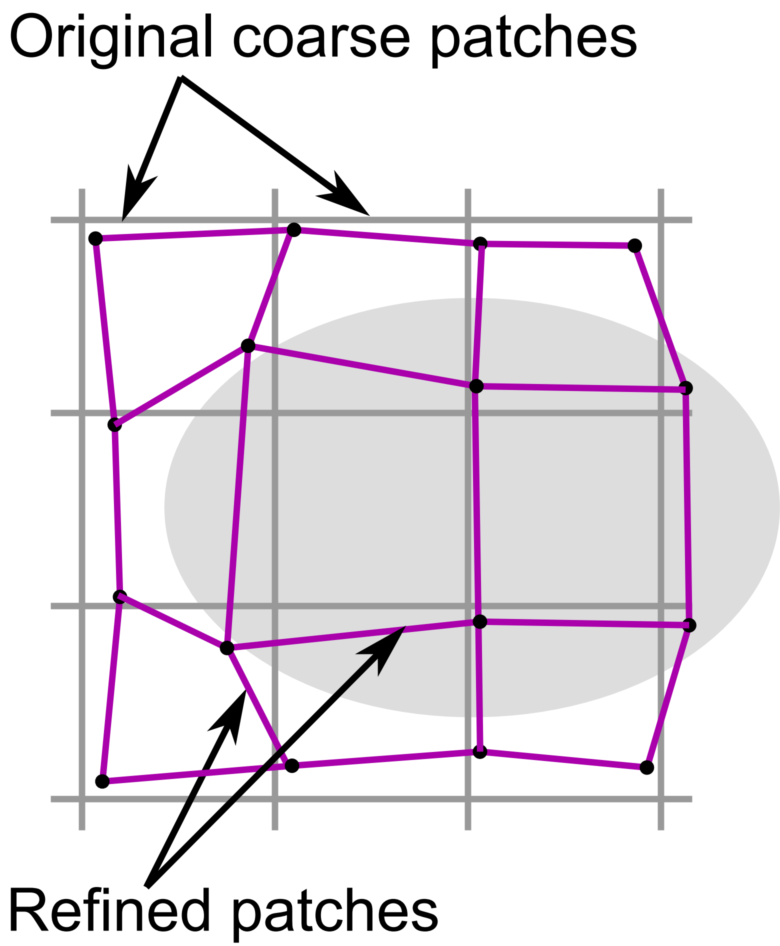
The assumptions are that

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

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



Computational issues of r-refinement

- Unlike the other solution (h-refinement) 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.

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