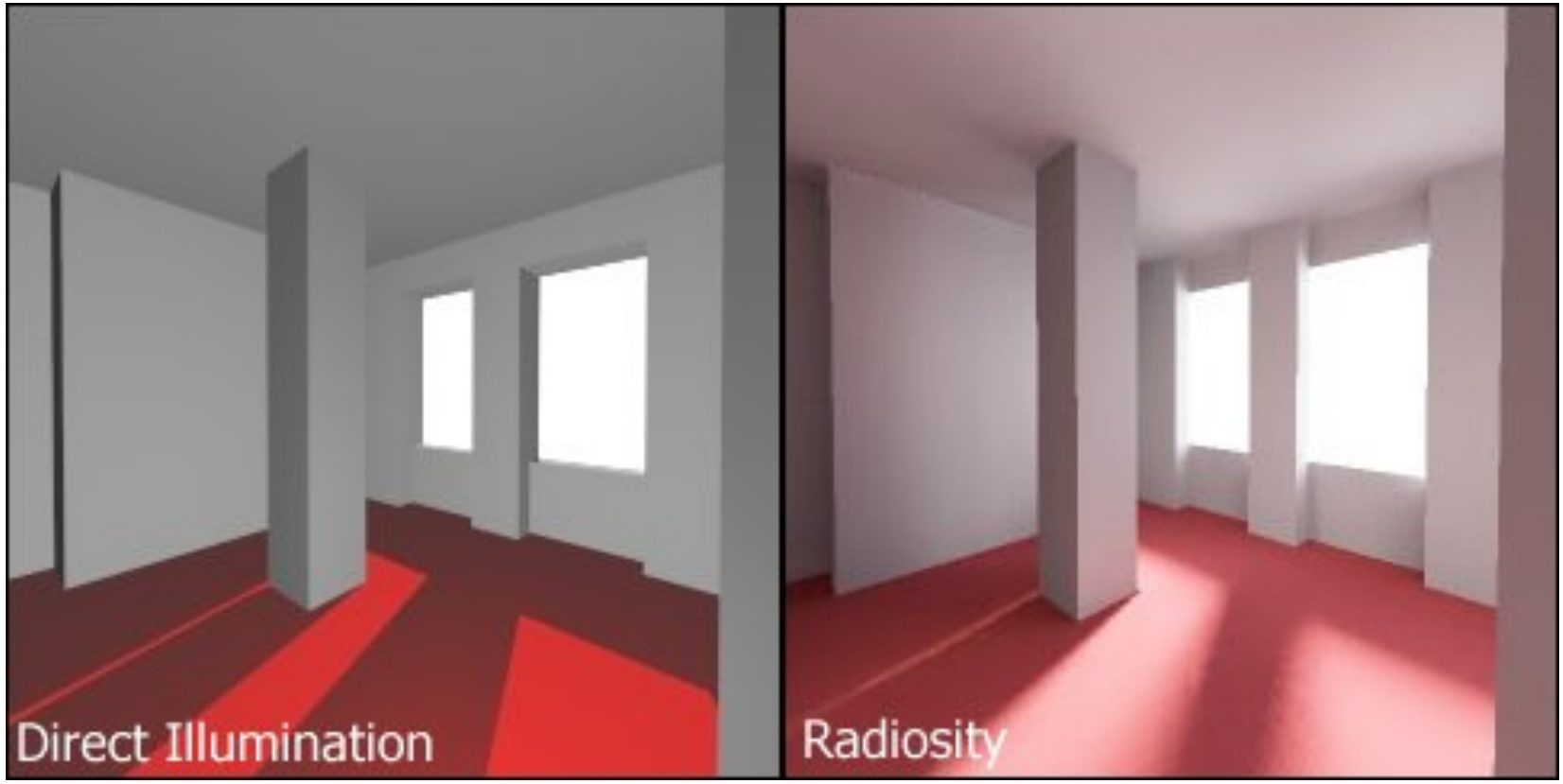
Interactive Computer Graphics: Lecture 17

Radiosity

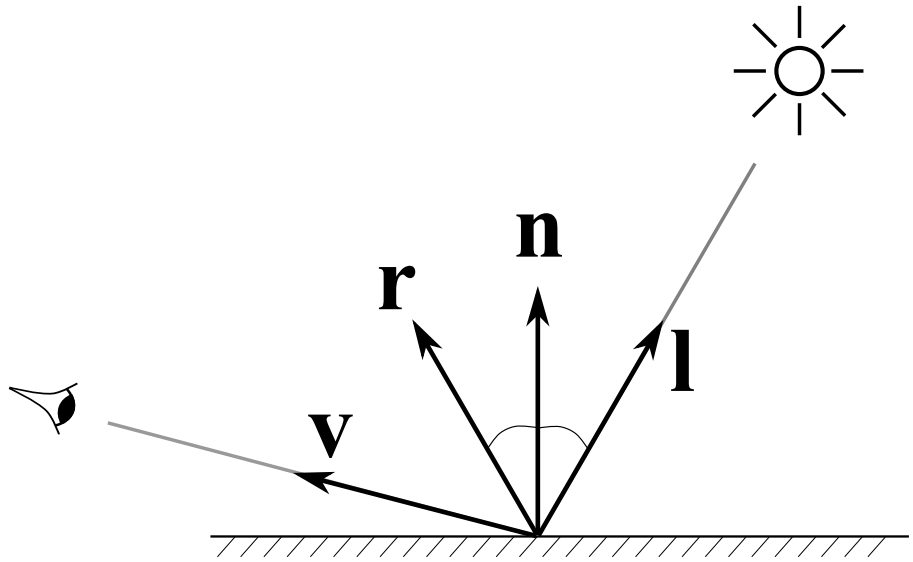

Direct Illumination


Direct Illumination
Radiosity

The reflectance equation

- Recall the reflectance equation: models reflected light from a surface:

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

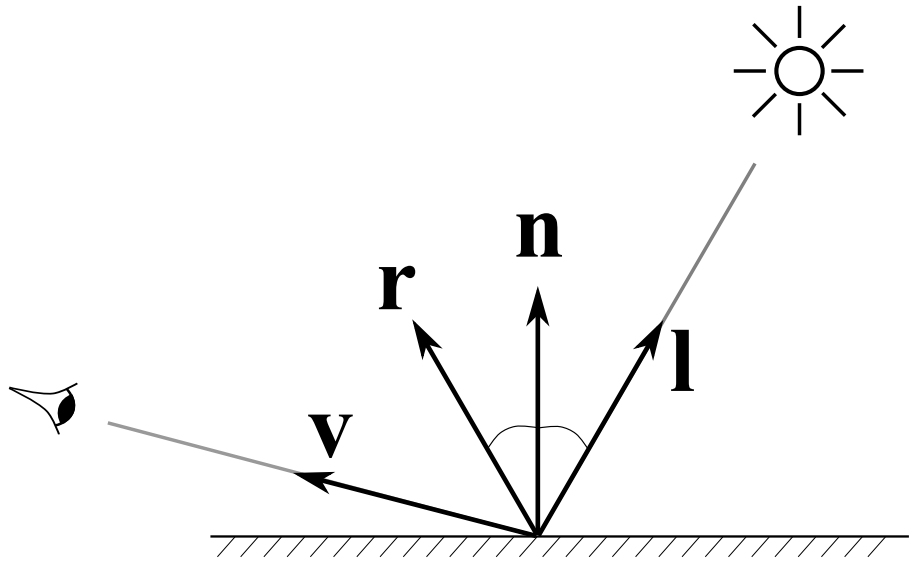


The reflectance equation

- Recall the reflectance equation: models reflected light from a surface:

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

- $I_{incident}$: light intensity



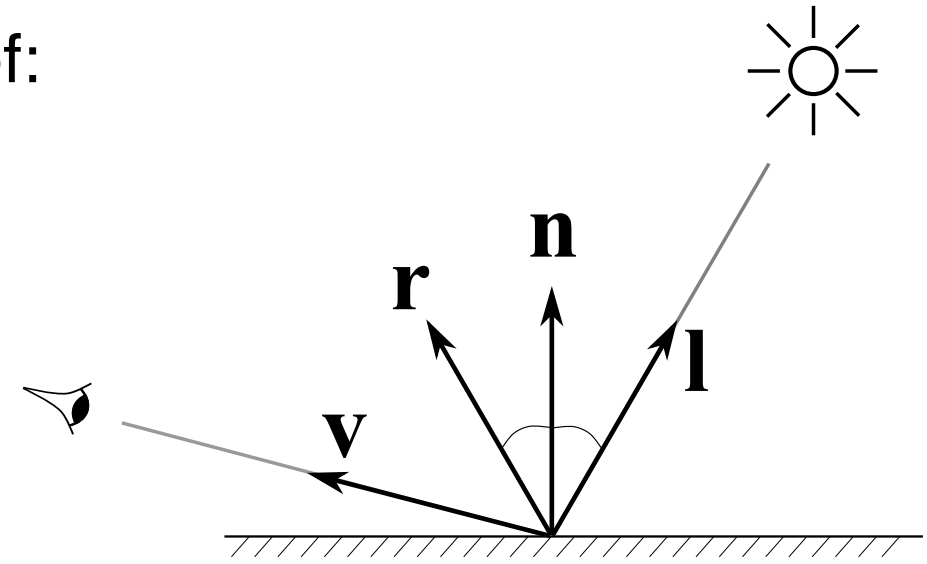
The reflectance equation

- Recall the reflectance equation: models reflected light from a surface:

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

- $I_{incident}$: light intensity

- $k_{\{a,d,s\}}$: control amounts of:
  - k_a: ambient light
  - k_d: diffuse light
  - k_s, q: specular reflection



Lighting model for ray tracing

For ray tracing we assumed that there were a small number of point light sources.

However, according to the reflectance equation, every surface is reflecting light, and so should also be considered a light source.

So rather than use a constant for ambient light, shouldn't we sum the light received from *all* other surfaces in the scene?

Ambient light

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

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

- Or, more simply, to write (for a given viewpoint)

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

- where R is the (viewpoint dependent) reflectance function.

Radiosity

- Radiosity is defined as the energy per unit area leaving a surface.
- It is the sum of
  - the emitted energy per unit area (if any)
  - the reflected energy.
- For a small area of the surface (patch) dA (where the emitted energy can be regarded as constant) we have:

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

- Notice that we now treat light sources as distributed

Radiosity

- Divide the scene into patches $i = 1, ..., n$
- For the i-th patch, let:
  - B_i = total energy leaving the patch
  - E_i = total energy emitted by patch itself
  - R_i = reflectance value
  - I_i = incident light energy arriving at the patch
- With this notation, the above equation can be re-written

$$B_i = E_i + R_i I_i$$

Collecting energy

We can estimate the incident energy for patch i as:

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

where the sum is taken over all surface patches of the scene

The B_j's in the sum represent the energy leaving all the other patches in the scene

F_{ij} is a constant that links surface patch i with patch j and is called the *form factor*

We can assume that F_{ii} = 0

Final formulation

- We can substitute this expression for incident light into the previous equation
- We obtain a discrete approximation for the energy leaving the i-th patch:

$$B_i = E_i + R_i\, I_i \quad \text{becomes} \quad B_i = E_i + R_i \sum_j B_j\, F_{ij}$$

- The form factors F_{ij} take into account
  - Patch areas
  - The angle at which they ‘face’ each other
- They control the amount of energy leaving patch i that reaches patch i

In matrix form

- Re-write the equation for the i-th patch

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

- Joining the equations for all patches, we can formulate the matrix equation:

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

In matrix form

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

- If we can solve this for every B_i then we will be able to render each patch with a correct light model.
- However, this is not so easy to do since
  - the form factors need to be found
  - the full reflectance equation is insolvable
  - the matrix is big - minimum 10000 by 10000

Wavelengths

- The radiosity values are wavelength dependent, hence we will need to compute a radiosity value for *R*, *G* and *B*.
- Each patch will require a separate set of parameters for *R*, *G* and *B*.
- The three radiosity values are the values that the rendered pixels will receive.

Back to the reflectance function

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

Note that the specular term depends on the relative positions of the viewpoint and each light source $\mathbf{v}$.

But now, every patch is a light source!

Specular reflections

- Moreover our light sources are no longer points, so we need to collect the incident light in a specular cone to determine the specular reflection.

- This is computationally infeasible.

- We will return to specularities later, but for the moment we will consider only diffuse radiosity.

Patching Problems

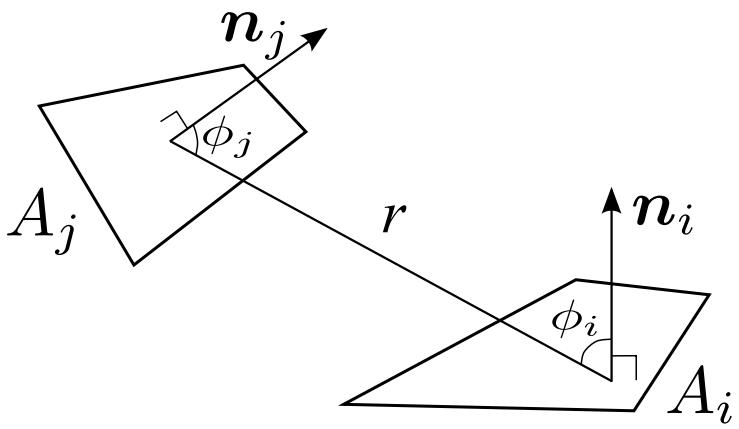
- We need to divide our graphics scene into patches for computing the radiosity.
- For small polygons we can perhaps use the polygon map, but for large polygons we need to subdivide them.
- Since the emitted light will not be constant across a large polygon we will see the subdivisions

Large Polygons

- Each patch will have a different but constant illumination.
- Thus we will see the patch boundaries unless either:
  - Patches project to (sub) pixel size or
  - We smooth the results (eg by interpolation)

Form Factors

- The form factors couple every pair of patches, determining the proportion of radiated energy from one that strikes the other.

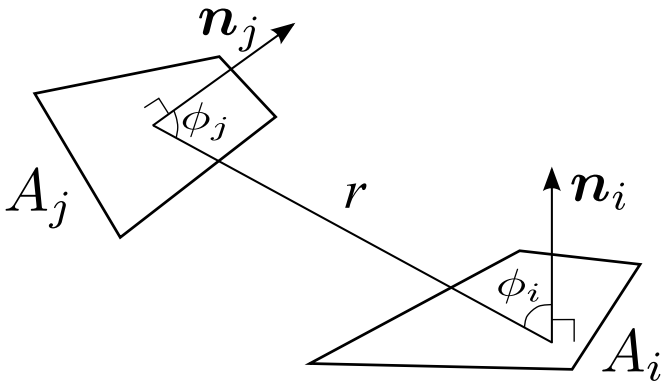


$$F_{ij} = \frac{1}{|A_i|} \int_{A_i} \int_{A_j} \frac{\cos\phi_i \ \cos\phi_j}{\pi r^2} dA_j dA_i$$

Form Factors - the definition

- The $\cos$ terms compute the projection of each patch in the direction to the other.
- If the patches are in the same plane, facing the same way, there is no coupling. If they directly face each other they are maximally coupled.

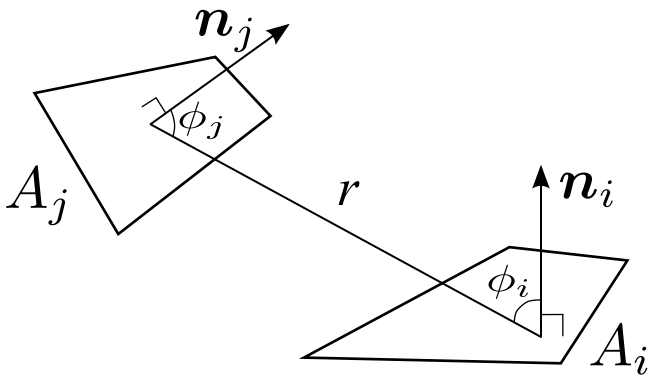
$$F_{ij} = \frac{1}{|A_i|} \int_{A_i} \int_{A_j} \frac{\cos\phi_i \ \cos\phi_j}{\pi r^2} dA_j dA_i$$



Form Factors - simplifying the computation

- The equation can be simplified if we assume that A_i is small compared with r.
- If this is the case, then we can treat the inner integral as constant over the surface of A_i.

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



Simplifying form factors

- With this assumption the outer integral evaluates to $|A_i|$ (i.e. the area of A_i).

- Hence we can write the integral as:

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

Further simplifying

We assumed that the radius is large compared with patch A_i. Should also be reasonable to assume it is large compared to the size of A_j.

Hence the integrand of

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

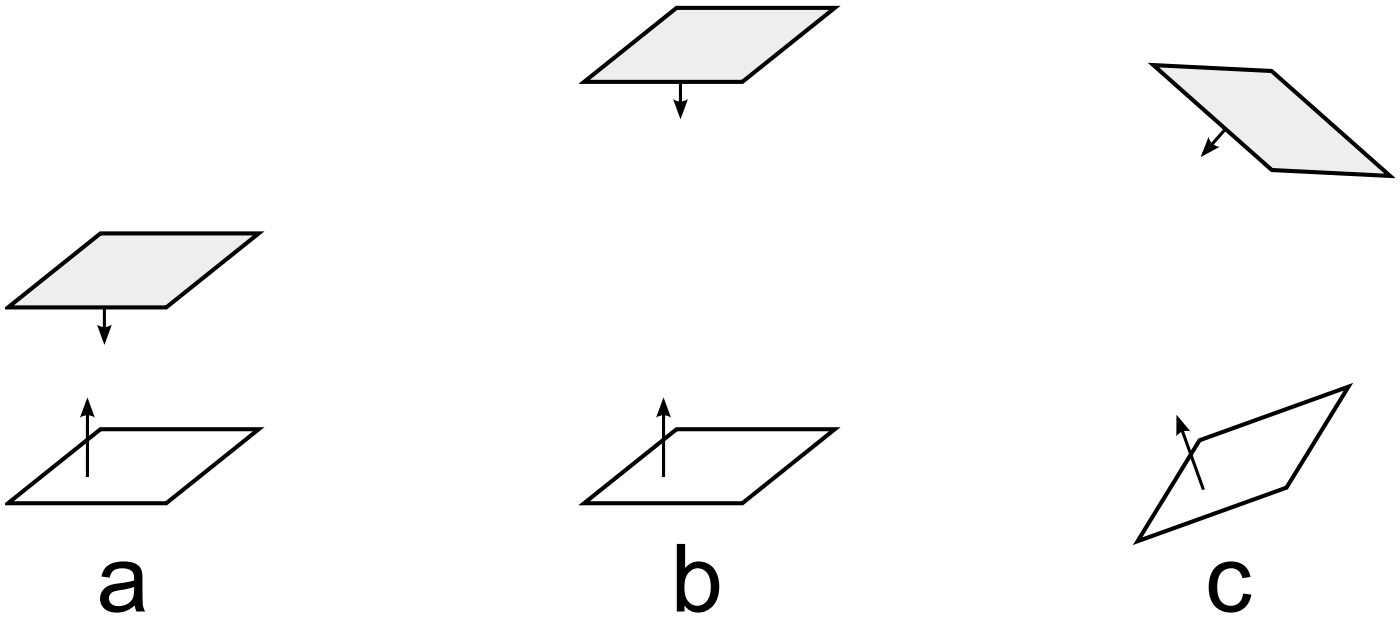
can similarly be considered constant over A_j

So we get the approximation

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

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

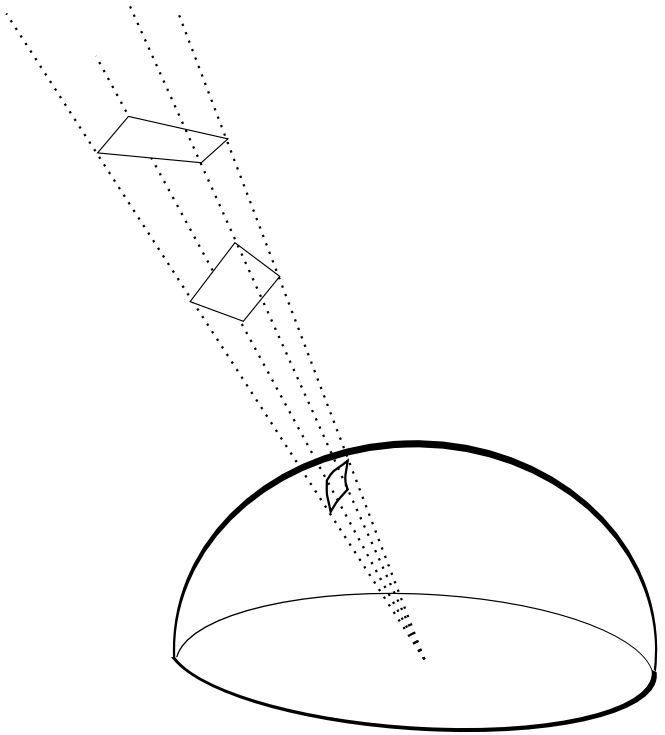
(c) Not really facing each other, even smaller form factor perhaps 0.1

The Hemicube method

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

Direct computation of the approximate form factor equation for every pair of patches will be expensive to compute.

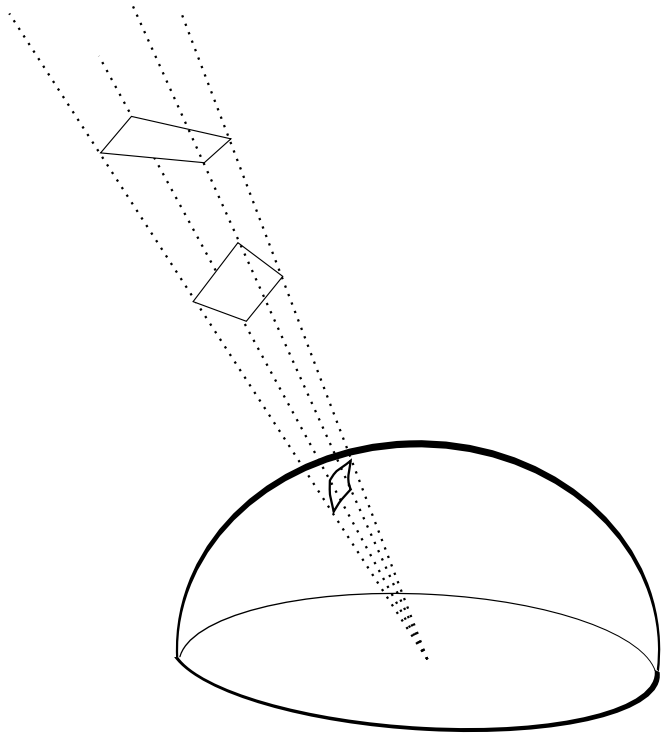
So an approximate computation method based on this observation (the hemi-cube) was developed.



The Hemicube method

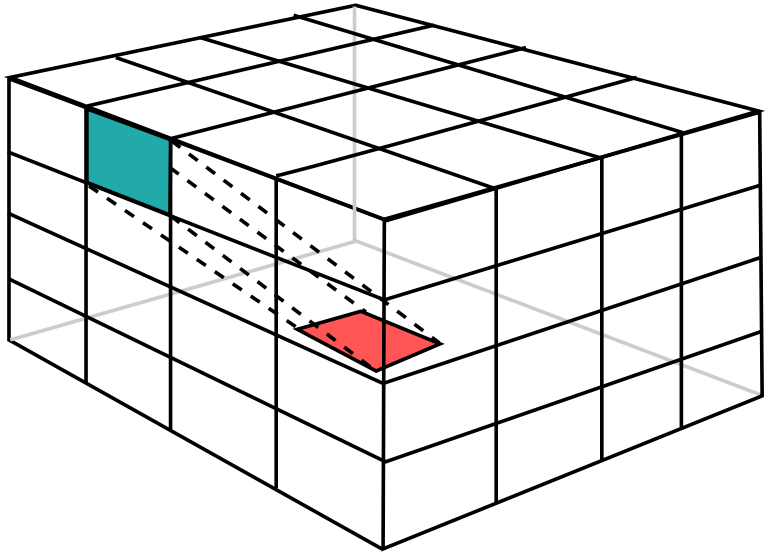
So, all patches that project onto the same area of a surrounding hemicube have approximately the same form factor.

The hemicube is preferred to the hemisphere since computing intersections with planes is computationally less demanding



Delta form factors

The hemicube is divided into small pixel areas and form factors are computed for each.



The resulting form factors can be used for every patch in the scene. We just ‘place’ the same hemicube over each.

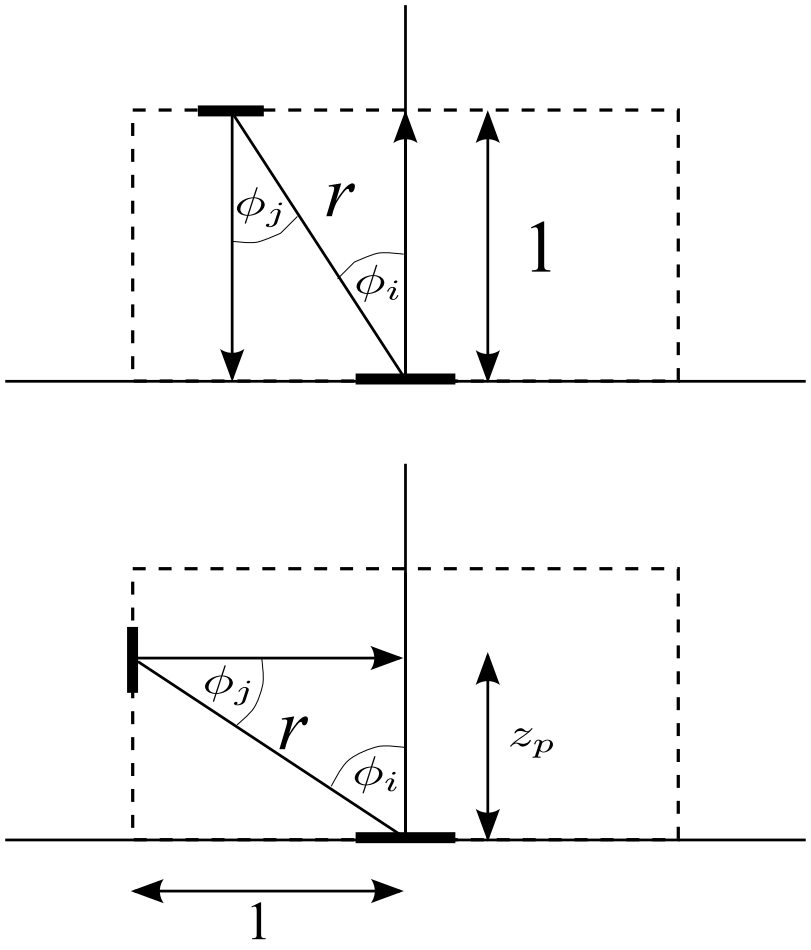
Delta form factors

- If the area of a hemicube pixel is $|A|$, its form factor is:

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

- These delta form factors can be computed and stored in a look up table.
- They can then be applied to every patch without the need for further form factor calculations.

Computing the delta form factors



For a top face we have:

$$\cos\phi_i = \cos\phi_j = \frac{1}{r}$$

so the form factor is $\frac{|A|}{\pi r^4}$

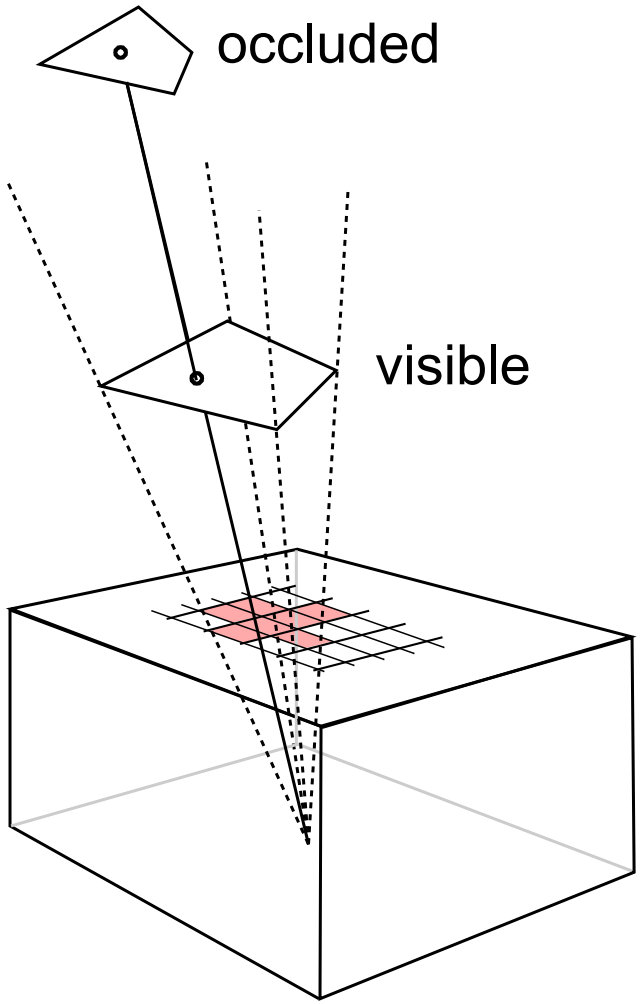
For a side face we have:

$$\cos\phi_i = \frac{1}{r} \qquad \cos\phi_j = \frac{z_p}{r}$$

so the form factor is $\frac{z_p\,|A|}{\pi r^4}$

Projection of patches onto the hemicube

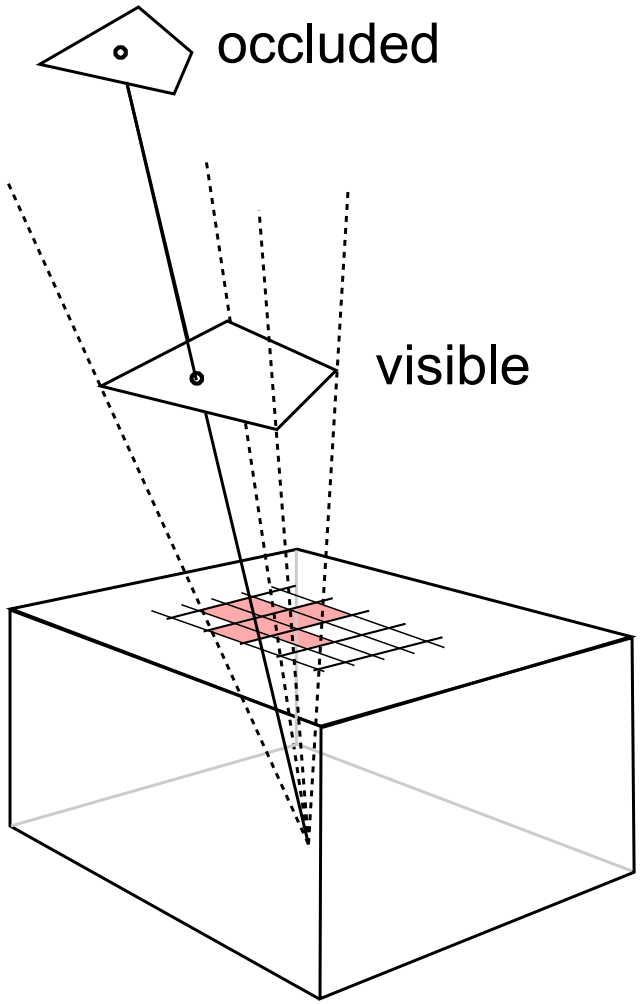
- We now need to know which patch is visible from each hemicube pixel.
- This could be done by ray tracing (casting), or projection.
- Ray tracing neatly solves the occlusion problem.
- Projection would require z-buffering.



Sum the pixels per patch

- Notice that all we need to determine is the nearest visible patch at each hemicube pixel.

- Once this is found we calculate the form factor for each patch by summing the delta form factors of the hemicube pixels to which it projects.



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

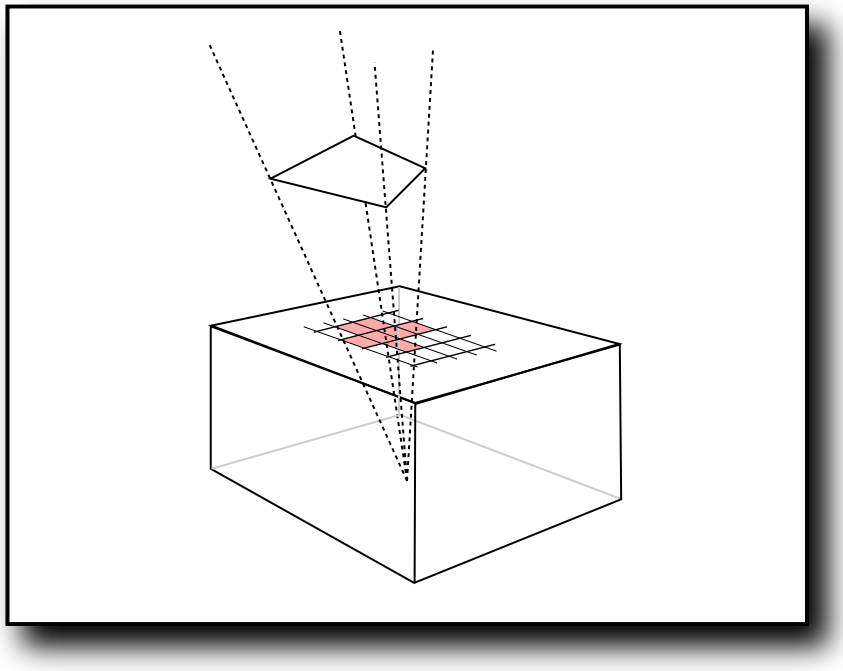
Summary of Radiosity method

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:

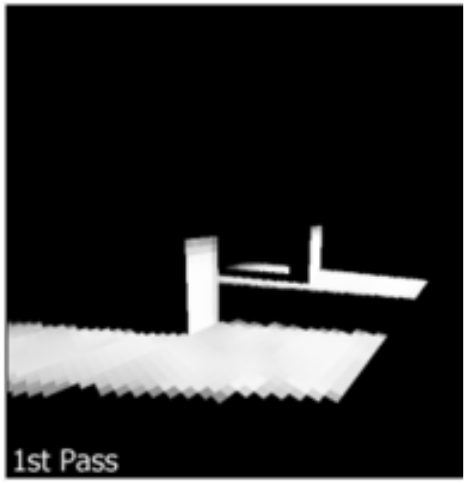
$$B_0 \leftarrow 0.1\,B_1 + 0.05\,B_2$$
$$B_1 \leftarrow 5 + 0.1\,B_0 + 0.15\,B_2$$
$$B_2 \leftarrow 0.02\,B_0 + 0.06\,B_1$$

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

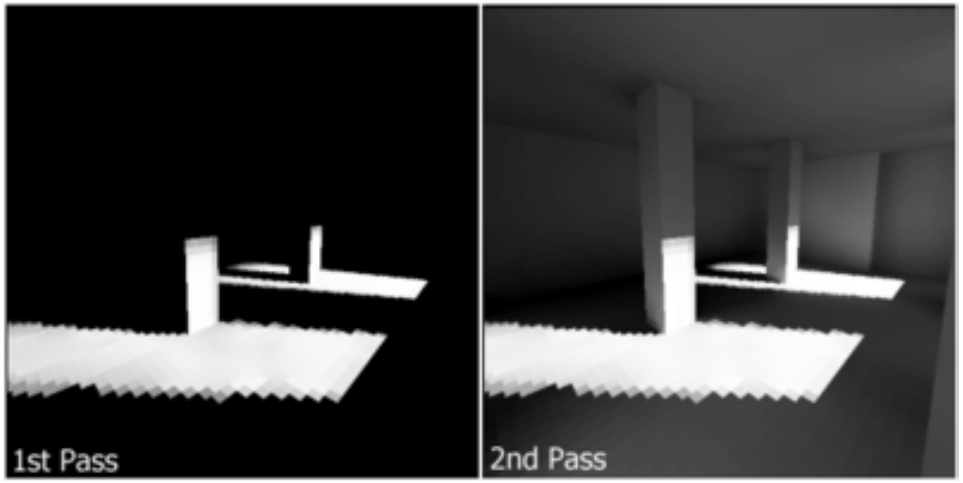
Gauss-Seidel method for solving equations

- 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



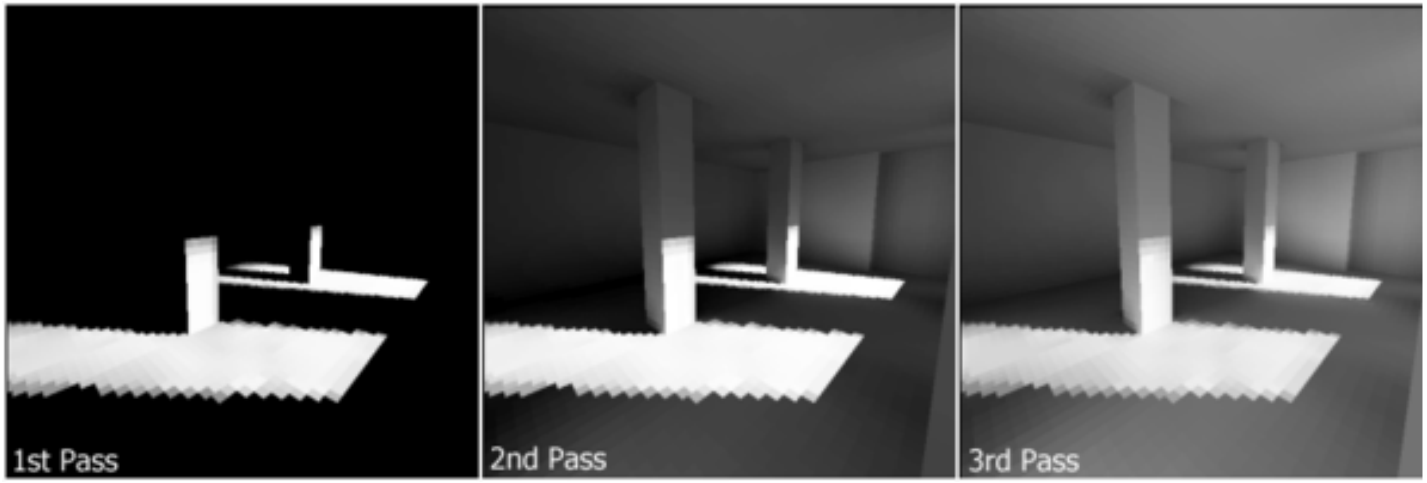
Gauss-Seidel method for solving equations

- 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



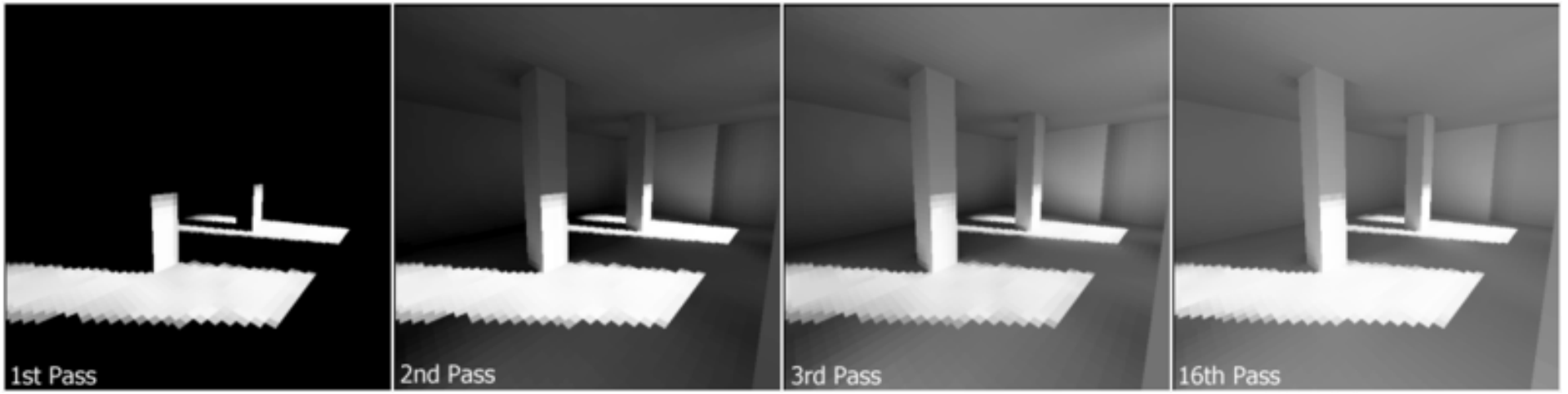
Gauss-Seidel method for solving equations

- 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



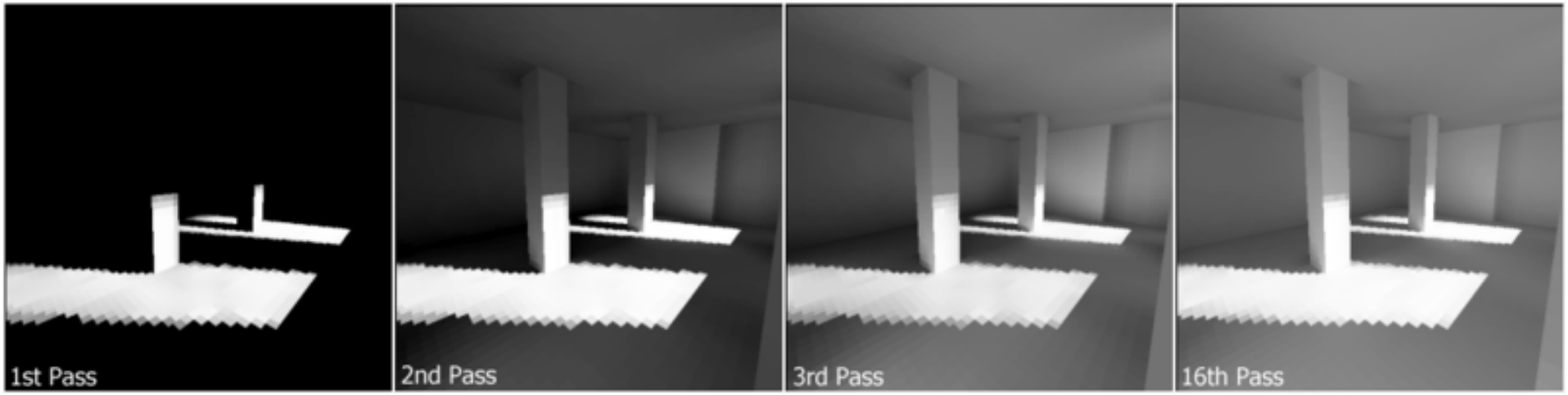
Gauss-Seidel method for solving equations

- 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.

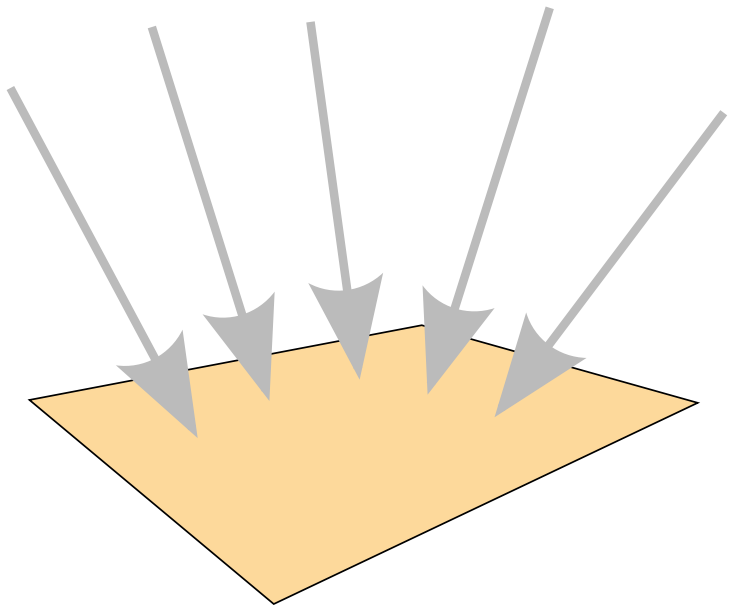
- The method is based on the idea of “shooting and gathering”, and also provides visual enhancement of the partial solution.

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*.

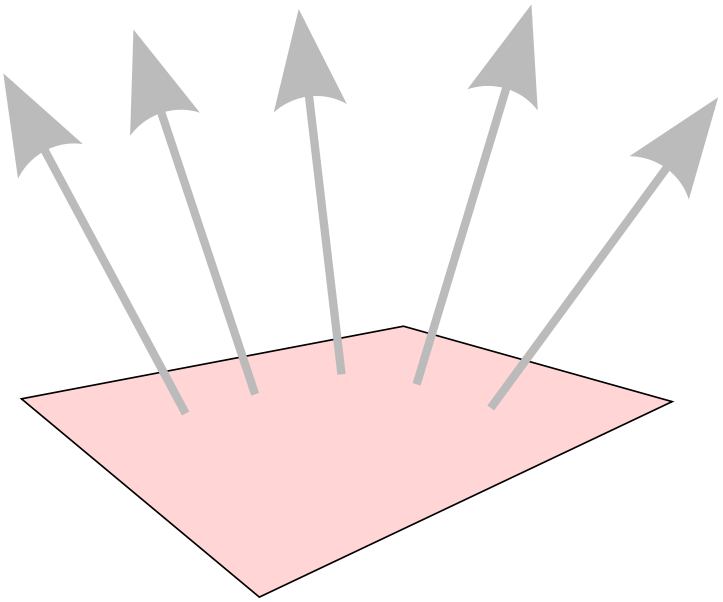


Shooting Patches

- 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

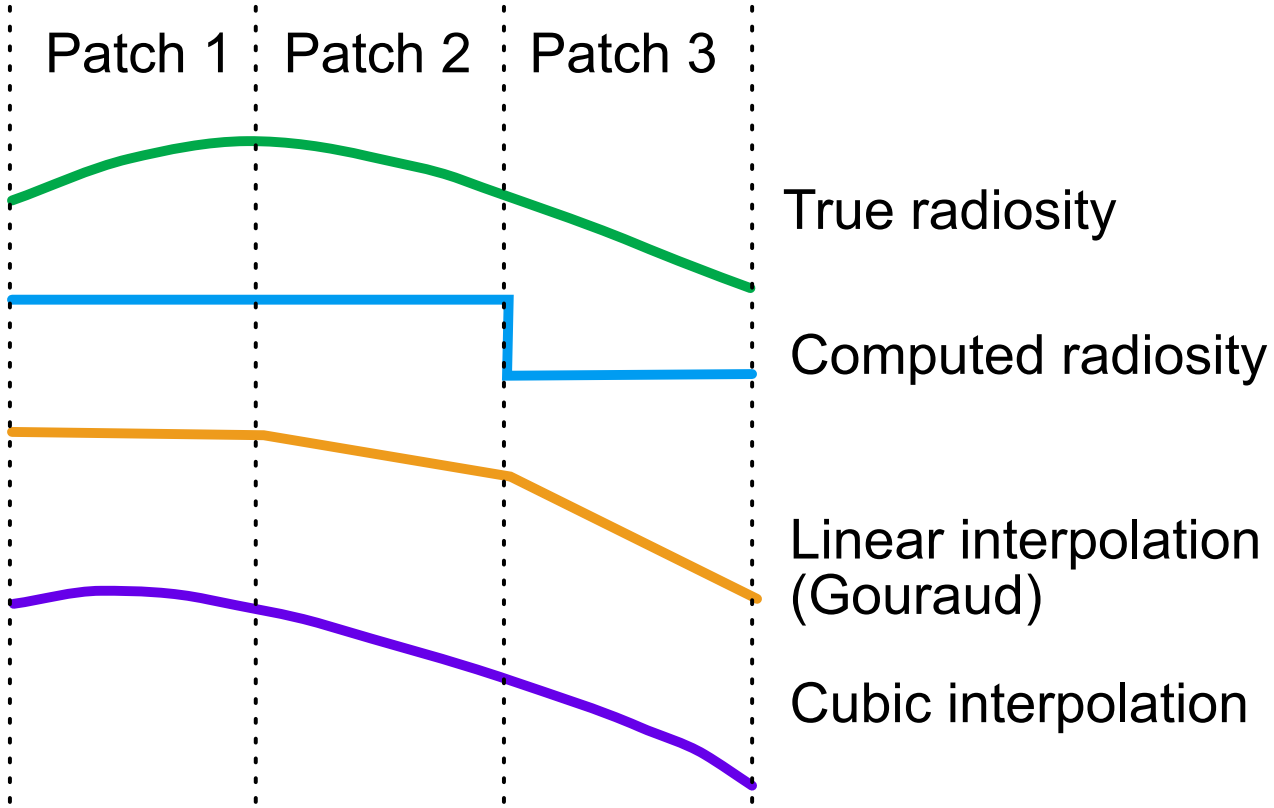
- Choose patch with largest unshot radiosity ΔB_i
- Shoot the radiosity for the chosen patch, i.e. for all other patches update

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

- 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

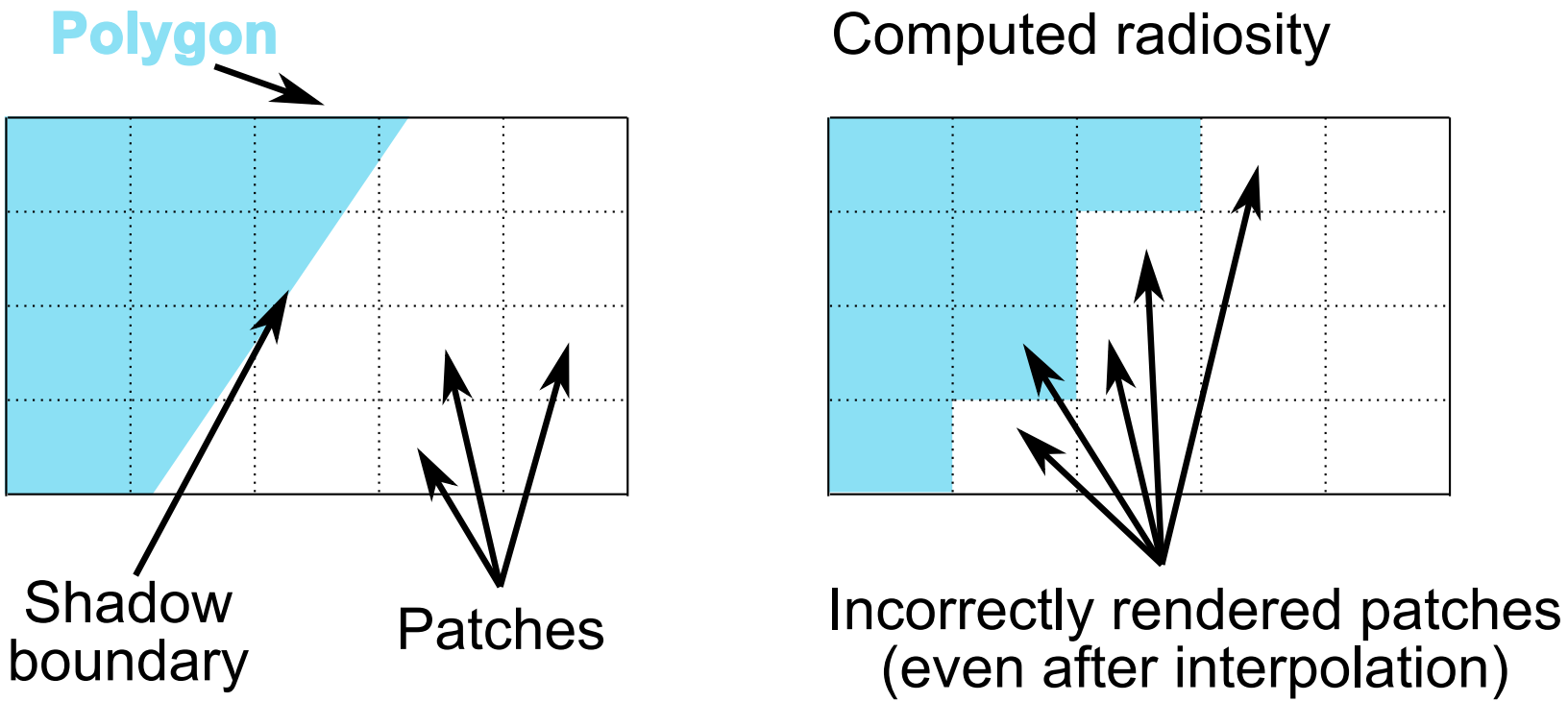


Meshing

- 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