Lecture 2: Radiosity

Fall 2004

Kavita Bala

Computer Science

Cornell University

Information

- Office Hours
  - Wed: 2-3 Upson 5142
- Web-page
- www.cs.cornell.edu/courses/cs665/2004fa/
  - Tentative schedule
  - Homeworks, lecture notes, will be on-line
  - Check for updates and announcements

Classic Ray Tracing

- Image-based
- Gathering approach
  - from the light sources (direct illumination)
  - from the reflected direction (perfect specular)
  - from the refracted direction (perfect specular)
- All other contributions are ignored!
  - Not a complete solution

Whitted RT Assumptions

- Light Source: point light source
  - Hard shadows
  - Single shadow ray direction
- Material: Blinn-Phong model
  - Diffuse with specular peak
- Light Propagation
  - Occluding objects
  - Specular interreflections only
    - trace rays in mirror reflection direction only

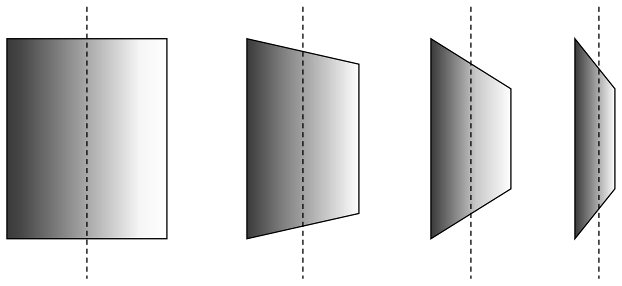
Other approaches

- Classic ray tracing:
  - Only perfect specular and perfect refraction/reflection
  - View-dependent
- Radiosity (1984)
  - Pure diffuse
  - View-independent
- Monte Carlo Ray Tracing (1986)
  - Global Illumination for any environment

Radiosity Advantages

- Physically based approach for diffuse environments
- Can model diffuse interactions, color bleeding, indirect lighting and penumbra (area light sources)
- Boundary element (finite element) problem
- Accounts for very high percentage of total energy transfer

Key Idea #1: diffuse only

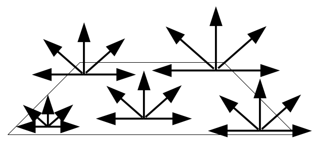


- Radiance independent of direction
- Surface looks the same from any viewpoint
- No specular reflections

Diffuse surfaces

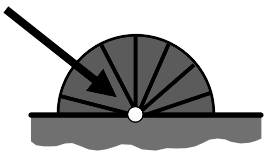
- Diffuse emitter

  L(x→Θ) = constant over Θ



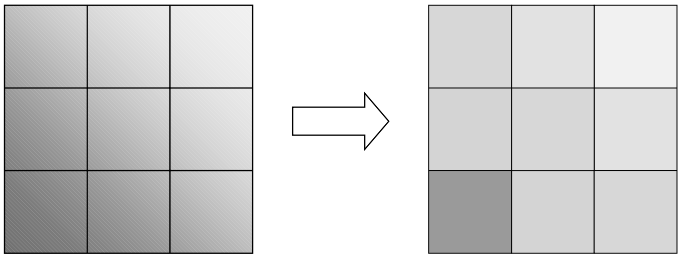
- Diffuse reflector

  Reflectivity constant



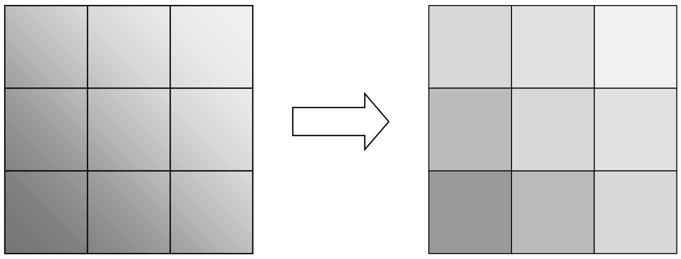
Key Idea #2: “constant”polygons

- Radiosity solution is an approximation, due to discretization of scene into patches



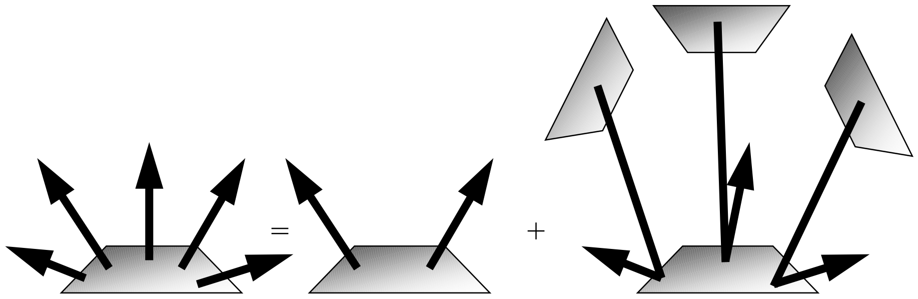
- Subdivide scene into small polygons

Constant radiance approximation



- Radiance is constant over a surface element
  $L(x)$ = constant over x
- surface element i: $L(x) = L_i$

Radiosity Equation



Emitted radiosity = self-emitted radiosity + received & reflected radiosity

$$Radiosity_i = Radiosity_{self,i} + \sum_{j=1}^{N} a_{j\to i} Radiosity_j$$

Radiosity Equation

- Radiosity equation for each polygon i

$$Radiosity_1 = Radiosity_{self,1} + \sum_{j=1}^{N} a_{j\to 1} Radiosity_j$$

$$Radiosity_2 = Radiosity_{self,2} + \sum_{j=1}^{N} a_{j\to 2} Radiosity_j$$

$\dots$

$$Radiosity_N = Radiosity_{self,N} + \sum_{j=1}^{N} a_{j\to N} Radiosity_j$$

- N equations; N unknown variables

Radiosity algorithm

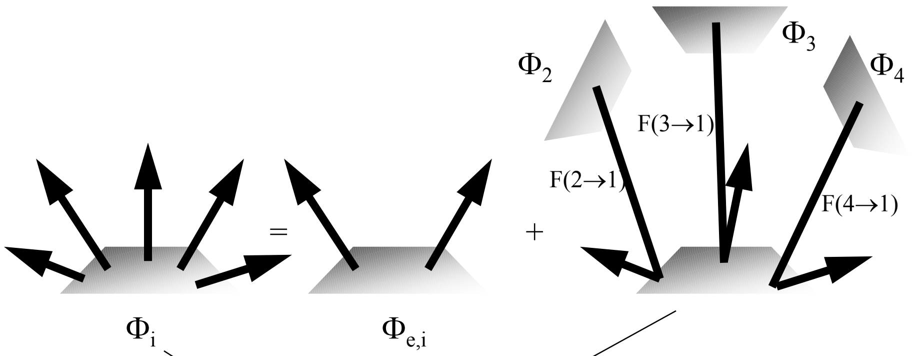
- Subdivide the scene into small polygons
- Compute for each polygon a constant illumination value
- Choose a viewpoint, and display the visible polygons

Radiosity algorithm

- Subdivide the scene in small polygons
- Compute for each polygon a constant illumination value
- Choose a viewpoint, and display the visible polygons
- Choose a new viewpoint ….
- … and another ….
- … and another ….

Radiosity: Typical Image

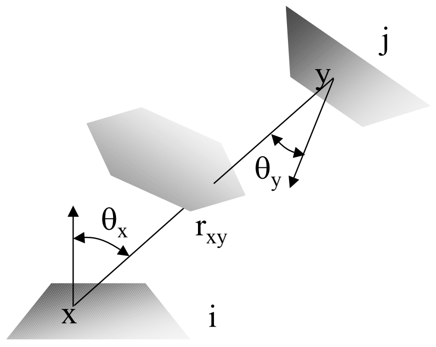
Energy Conservation Equation



$$\Phi_i = \Phi_{e,i} + \rho_i \sum_{j=1}^{N} \Phi_j F(j \to i)$$

Compute Form Factors

$$F(j \to i) = \frac{1}{A_j} \int_{A_i} \int_{A_j} \frac{\cos\theta_x \cdot \cos\theta_y}{\pi \cdot r_{xy}^2} \cdot V(x,y) \cdot dA_y \cdot dA_x$$



Form Factors Invariant

$$F(j \to i) = \frac{1}{A_j} \int_{A_i} \int_{A_j} \frac{\cos\theta_x \cdot \cos\theta_y}{\pi \cdot r_{xy}^2} \cdot V(x,y) \cdot dA_y \cdot dA_x$$

$$F(i \to j) = \frac{1}{A_i} \int_{A_j} \int_{A_i} \frac{\cos\theta_x \cdot \cos\theta_y}{\pi \cdot r_{xy}^2} \cdot V(x,y) \cdot dA_y \cdot dA_x$$

$$F(i \to j) A_i = F(j \to i) A_j$$

Radiosity Equation

- Radiosity for each polygon i

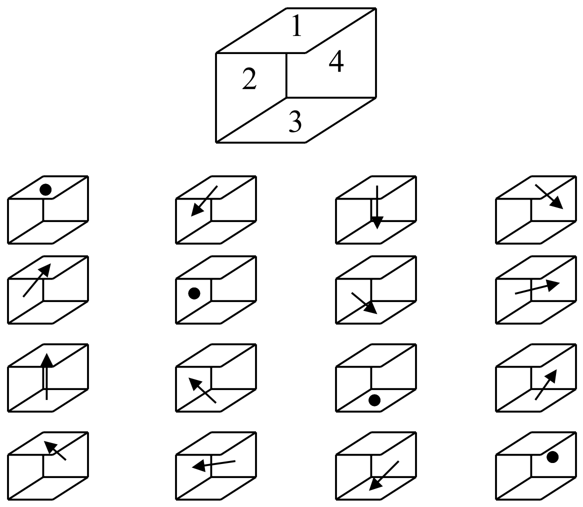
$$\forall i : B_i = B_{e,i} + \rho_i \sum_{j=1}^{N} B_j F(i \to j)$$

- Linear system
  - B_i : radiosity of patch i (unknown)
  - $B_{e,i}$: emission of patch i (known)
  - ρ_I : reflectivity of patch i (known)
  - $F(i \to j)$: form-factor (coefficients of matrix)

Linear System

$$\underbrace{\begin{bmatrix} 1-\rho_1 F_{1\to1} & -\rho_1 F_{1\to2} & \dots & -\rho_1 F_{1\to n} \\ -\rho_2 F_{2\to1} & 1-\rho_2 F_{2\to2} & \dots & -\rho_2 F_{2\to n} \\ \dots & \dots & \dots & \dots \\ -\rho_n F_{n\to1} & -\rho_n F_{n\to2} & \dots & 1-\rho_n F_{n\to n} \end{bmatrix}}_{\text{known}} \underbrace{\begin{bmatrix} B_1 \\ B_2 \\ \dots \\ B_n \end{bmatrix}}_{\text{unknown}} = \underbrace{\begin{bmatrix} B_{e,1} \\ B_{e,2} \\ \dots \\ B_{e,n} \end{bmatrix}}_{\text{known}}$$

Linear System

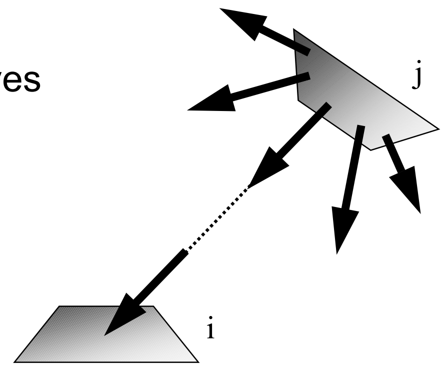


Radiosity Algorithm

- Subdivide scene in polygons
  - mesh that determines final solution
- Compute Form Factors
  - transfer of energy between polygons
- Solve linear system
  - results in power (color) per polygon
- Pick a viewpoint and display
  - loop

Form Factor

- $F_{j \rightarrow i}$ = the fraction of power emitted by j, which is received by i
- Area
  - if i is smaller, it receives less power
- Orientation
  - if i faces j, it receives more power
- Distance
  - if i is further away, it receives less power

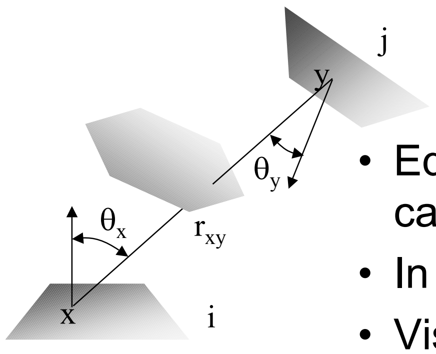


Form Factors - how to compute?

- Closed Form
  - Analytic
- Hemicube
- Monte-Carlo

Form Factor – Analytical

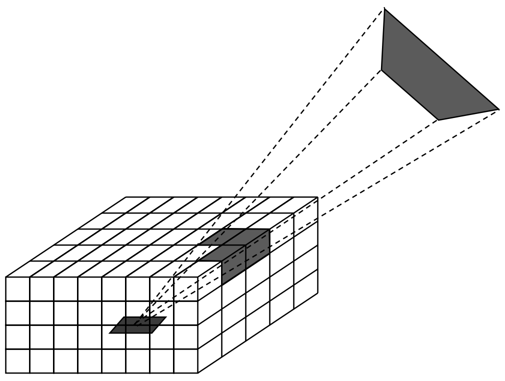
$$F(j \to i) = \frac{1}{A_j} \int_{A_i} \int_{A_j} \frac{\cos\theta_x \cdot \cos\theta_y}{\pi \cdot r_{xy}^2} \cdot V(x,y) \cdot dA_y \cdot dA_x$$



- Equations for special cases (polygons)
- In general hard problem
- Visibility makes it harder

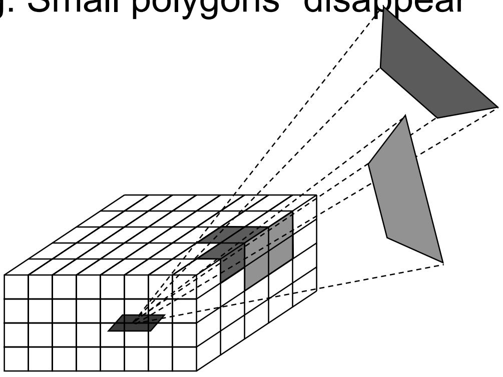
Form Factors - Hemicube

- Project patch on hemicube
- Add hemicube cells to compute form factor



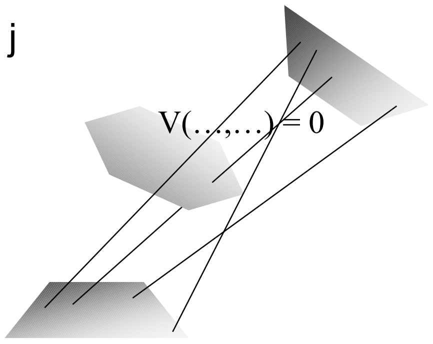
Form Factors - Hemicube

- Depth information per pixel evaluates visibility
- FFs for all polygons in scene
- Hardware rendering (Z-buffer)
- Severe aliasing: Small polygons "disappear"



FF - Monte Carlo

- Generate point on patch i
- Generate point on patch j
- Evaluate integrand
- Compute average



$$\langle F(j \to i) \rangle = \frac{1}{N \cdot A_j} \sum_{k=1}^{N} \frac{\cos\theta_{x_k} \cdot \cos\theta_{y_k}}{p(x_k, y_k) \cdot \pi \cdot r_{x_k y_k}^2} \cdot V(x_k, y_k)$$

Form Factors

- Visibility checks are most expensive operation
- FFs are usually computed when needed
  - computationally expensive
  - memory O(N^2)

Radiosity Algorithm

- Subdivide scene in polygons
  - mesh that determines final solution
- Compute Form Factors
  - transfer of energy between polygons
- Solve linear system
  - results in power (color) per polygon
- Pick a viewpoint and display
  - loop

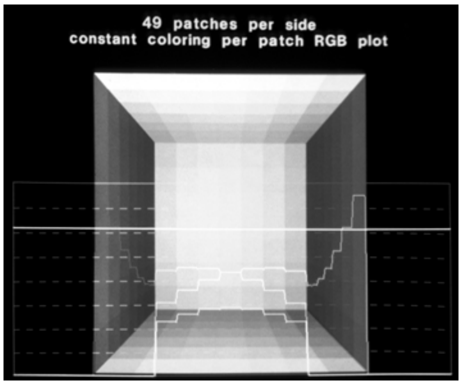
How To Solve Linear System

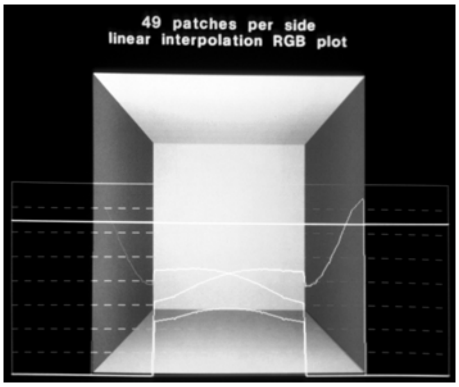
- Matrix Inversion
- Gathering methods
  - Jacobi iteration
  - Gauss-Seidel
- Shooting
  - Southwell iteration
  - Improved Southwell iteration

Matrix Inversion

$$\begin{bmatrix} B_1 \\ B_2 \\ \dots \\ B_n \end{bmatrix} = \begin{bmatrix} 1-\rho_1 F_{1\to 1} & -\rho_1 F_{1\to 2} & \dots & -\rho_1 F_{1\to n} \\ -\rho_2 F_{2\to 1} & 1-\rho_2 F_{2\to 2} & \dots & -\rho_2 F_{2\to n} \\ \dots & \dots & \dots & \dots \\ -\rho_n F_{n\to 1} & -\rho_n F_{n\to 2} & \dots & 1-\rho_n F_{n\to n} \end{bmatrix}^{-1} \begin{bmatrix} B_{e,1} \\ B_{e,2} \\ \dots \\ B_{e,n} \end{bmatrix}$$

$O(n^3)$





Iterative approaches

- Jacobi iteration
- Start with initial guess for energy distribution (light sources)
- Update radiosity/power of all patches based on the previous guess

$$\underbrace{B_i}_{\text{new value}} = B_{e,i} + \rho_i \sum_{j=1}^{N} \underbrace{B_j}_{\text{old values}} F(i \rightarrow j)$$

- Repeat until converged

Jacobi

- For all patches i (i=1...N) : $B_i^{(0)} = B_{e,i}$
- while not converged:
  - for all patches i (i=1...N)

$$B_i^{(g)} = B_{e,i} + \rho_i \sum_{j=1}^{N} B_j^{(g-1)} F(i \to j)$$

update of 1 patch requires evaluation of N FFs

Improved Gathering

- Jacobi iteration only uses values of previous iterations to compute new values
- Gauss-Seidel iteration
  - New values used immediately
  - Slightly better convergence

Gauss-Seidel

- For all patches i (i=1...N) : $B_i^{(0)} = B_{e,i}$
- while not converged:
  - for all patches i (i=1...N)

$$B_i^{(g)} = B_{e,i} + \rho_i \sum_{j=1}^{i-1} B_j^{(g)} F(i \to j) + \rho_i \sum_{j=i}^{N} B_j^{(g-1)} F(i \to j)$$

Example

Progressive Radiosity

- Gathering: $O(n^2)$/iteration
  - Still too slow
- Can use “shooting” as opposed to “gathering” approach
- 1-2% of all emitting and reflecting surfaces can account for very high percentage of energy

Southwell Iteration

- “Shooting” method
- Start with initial guess for light distribution (light sources)
- Select patch and distribute its energy over all polygons

Southwell Iteration (Wrong)

- For all patches i (i=1..N) :
  - $B_i^{(0)} = B_{e,i}$
- while not converged:
  - select shooting patch k with $B_k^{(g-1)} \neq 0$
  - for all patches i (i=1..N)

with n FF evaluations, n patches are updated!

Southwell Iteration

- Keep record of “unshot” radiosity/energy per patch
- Repeat shooting of unshot energy until converged

Southwell Iteration (Correct)

- For all patches i (i=1..N) :
  - $B_i^{(0)} = B_{e,i}$ $\Delta B_i^{(0)} = B_{e,i}$
- while not converged:
  - select shooting patch k with $\Delta B_k^{(g-1)} \neq 0$
  - for all patches i (i=1..N)
  - $\Delta B_k^{(g)} = 0$

with n FF evaluations, n patches are updated!

Progressive Radiosity

- Solution time is fast: O(n) for first results
- Can monotonically approach the complete diffuse radiosity solution

Progressive Refinement

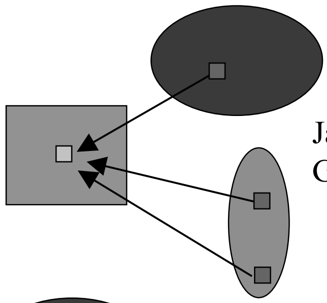
- Southwell selects shooting patches in no particular order
- Progressive refinement radiosity selects patch with largest unshot energy
- First image is generated fairly quickly!

PR + Ambient term

- PR gives an estimate for each radiosity value that is smaller than the real value
- Estimate can be improved by using ambient term
  - Add all unshot energy
  - Distribute total unshot energy equally over all patches
- Solution has improved energy distribution

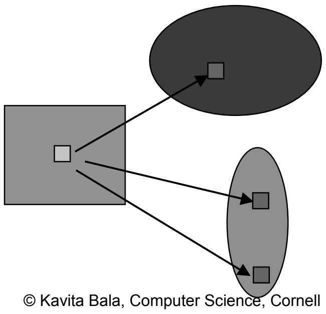
Gathering vs. Shooting

- Gathering



Jacobi
Gauss-Seidel

- Shooting

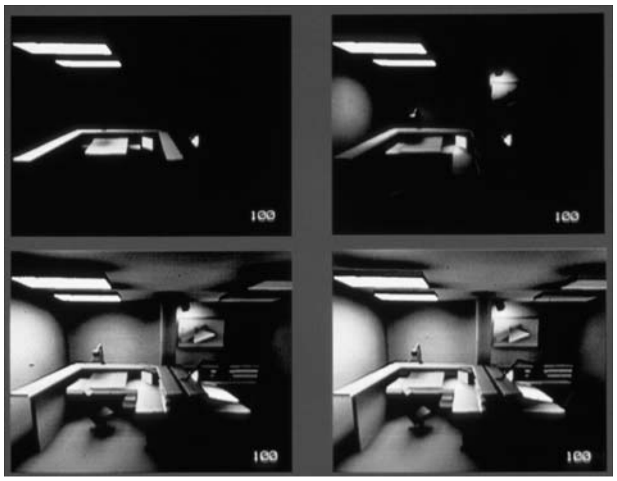


Southwell
Progressive Radiosity

Comparison

Gauss-Seidel

Southwell



Southwell + sorting

Southwell +sorting +ambient

Radiosity Algorithm

- Subdivide scene in polygons
  - mesh that determines final solution
- Compute Form Factors
  - transfer of energy between polygons
- Solve linear system
  - results in power (color) per polygon
- Pick a viewpoint and display
  - Loop over different viewpoints