
Hong Qin
Department of Computer Science
Stony Brook University (State University of New York)
Stony Brook, New York 11794-2424
Tel: (631)632-8450; Fax: (631)632-8334
qin@cs.stonybrook.edu, qin@cs.sunysb.edu
http://www.cs.stonybrook.edu/~qin
Global Illumination

- **Global Illumination**
  - A point is illuminated by more than light from local lights
  - It is illuminated by all the emitters and reflectors in the global scene
    - **Ray Tracing**
    - **Radiosity**
Ray Tracing
Ray Tracing Fundamentals

- Represent *specular* global lighting
- Trace light backward (usually) from the eye, through the pixel, and into the scene
- Recursively bounce off objects in the scene, accumulating a color for that pixel
- Final output is single image of the scene
Recursive Ray Tracing

- Cast a ray from the viewer’s eye through each pixel
- Compute intersection of this ray with objects from scene
- Closest intersecting object determines color
Recursive Ray Tracing

• For each ray cast from the eyepoint
  – If surface is struck
    • Cast ray to each light source (shadow ray)
    • Cast reflected ray (feeler ray)
    • Cast transmitted ray (feeler ray)
    • Perform Phong lighting on all incoming light
  – Note that, diffuse component of Phong lighting is not pushed through the system
Recursive Ray Tracing

- Computing all shadow and feeler rays is slow
  - Stop after fixed number of iterations
  - Stop when energy contributed is below threshold

- Most work is spent testing ray/plane intersections
  - Use bounding boxes to reduce comparisons
  - Use bounding volumes to group objects
  - Parallel computation (on shared-memory machines)
Recursive Ray Tracing

• **Just a sampling method**
  - We’d like to cast infinite rays and combine illumination results to generate pixel values
  - Instead, we use pixel locations to guide ray casting

• **Problems?**
Problems With Ray Tracing

- **Aliasing**
  - Supersampling
  - Stochastic sampling

- Works best on specular surfaces (not diffuse)

- For perfectly specular surfaces
  - Ray tracing \( \equiv \) rendering equation (subject to aliasing)
Ray Tracing - Pros

• Simple idea and nice results
• Inter-object interaction possible
  – Shadows
  – Reflections
  – Refractions (light through glass, etc.)
• Based on real-world lighting
Ray Tracing - Cons

- Takes a long time
- Computation speed-ups are often highly scene-dependent
- Lighting effects tend to be abnormally sharp, without soft edges, unless more advanced techniques are used
- Hard to put into hardware
Supersampling - I

- Problem: each pixel of the display represents one single ray
  - Aliasing
  - Unnaturally sharp images
- Solution: send multiple rays through each “pixel” and average the returned colors together
Supersampling - II

• **Direct supersampling**
  – Split each pixel into a grid and send rays through each grid point

• **Adaptive supersampling**
  – Split each pixel only if it’s significantly different from its neighbors

• **Jittering**
  – Send rays through randomly selected points within the pixel
Soft Shadow

- Basic shadow generation was an on/off choice per point
- “Real” shadows do not usually have sharp edges
- Instead of using a point light, use an object with area
- Shoot jittered shadow rays toward the light and count only those that hit it
Soft Shadow Example

Hard shadow

Soft shadow
Radiosity

- Ray tracing models specular reflection and refractive transparency, but still uses an ambient term to account for other lighting effects.
- Radiosity is the rate at which energy is emitted or reflected by a surface.
- By conserving light energy in a volume, these radiosity effects can be traced.
Radiosity – Basic Concept

- Radiosity of a surface: rate at which energy leaves a surface
  - emitted by surface and reflected from other surfaces
- Represent diffuse global lighting
- Create a closed energy system where every polygon emits and/or bounces some light at every other polygon
- Calculate how light energy spreads through the system
- Solve a linear system for radiosity of each “surface”
  - Dependent on emissive property of surface
  - Dependent on relation to other surfaces (form factors)
- Final output is a polygon mesh with pre-calculated colors for each vertex
Radiosity
Radiosity

- **Break environment up into a finite number** $n$ **of discrete patches**
  - Patches are opaque Lambertian surfaces of finite size
  - Patches emit and reflect light uniformly over their entire surface
Radiosity

• Model light transfer between patches as a system of linear equations
• Solving this system gives the intensity at each patch
• Solve for R, G, B intensities and get color at each patch
• Render patches as colored polygons in OpenGL
Radiosity

- All surfaces are assumed perfectly diffuse
  - What does that mean about property of lighting in scene?
  - Light is reflected equally in all directions
  - Same lighting independent of viewing angle/ location
  - Only a subset of the Rendering Equation

Diffuse-diffuse surface lighting effects possible
The “Rendering Equation”

- Jim Kajiya (current head of Microsoft Research) developed this in 1986

\[ I(x, x') = g(x, x') [ \varepsilon(x, x') + \int_{S} \rho(x, x', x'') I(x', x'') dx'' ] \]

- \( I(x, x') \) is the total intensity from point \( x' \) to \( x \)
- \( g(x, x') = 0 \) when \( x/x' \) are occluded and \( 1/d^2 \) otherwise (\( d = \) distance between \( x \) and \( x' \))
- \( \varepsilon(x, x') \) is the intensity emitted by \( x' \) to \( x \)
- \( \rho(x, x', x'') \) is the intensity of light reflected from \( x'' \) to \( x \) through \( x' \)
- \( S \) is all points on all surfaces
Radiosity Equation

- Then for each surface $i$:

$$B_i = E_i + \rho_i \sum B_j F_{ji} \left(\frac{A_j}{A_i}\right)$$

where

- $B_i, B_j$ = radiosity of patch $i, j$
- $A_i, A_j$ = area of patch $i, j$
- $E_i$ = energy/area/time emitted by $i$
- $\rho_i$ = reflectivity of patch $i$
- $F_{ji}$ = Form factor from $j$ to $i$
Form Factors

- **Form factor**: fraction of energy leaving the entirety of patch $i$ that arrives at patch $j$, accounting for:
  - The shape of both patches
  - The relative orientation of both patches
  - Occlusion by other patches
Form Factors

- Compute n-by-n matrix of form factors to store radiosity relationships between each light patch and every other light patch.

\[ dF_{di,dj} = \frac{\cos \theta_i \cos \theta_j}{\pi r^2} H_{ij} dA_j \]
Form Factor – Another Example

- **Spherical projections to model form factor**
  - Project polygon $A_j$ on unit hemisphere centered at (and tangent to) $A_i$
    - Contributes $\cos \theta_j / r^2$
  - Project this projection to base of hemisphere
    - Contributes $\cos \theta_i$
  - Divide this area by area of circle base
    - Contributes $\pi(1^2)$

\[ dF_{di,dj} = \frac{\cos \theta_i \cos \theta_j}{\pi r^2} H_{ij} dA_j \]
Form Factor – Another Model

• Hemicube allows faster computations
  – Analytic solution of hemisphere is expensive
  – Use rectangular approximation, Hemicube
  – Cosine terms for top and sides are simplified
  – Dimension of 50 – 200 squares is good
Form Factors Properties

• In diffuse environments, form factors obey a simple reciprocity relationship:

\[ A_i F_{ij} = A_i F_{ji} \]

• Which simplifies our equation:

\[ B_i = E_i + \rho_i \sum B_j F_{ij} \]

• Rearranging to:

\[ B_i - \rho_i \sum B_j F_{ij} = E_i \]
Radiosity Equation

- So...light exchange between all patches becomes a matrix:

\[
\begin{bmatrix}
1 - \rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\
-\rho_2 F_{21} & 1 - \rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
-\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 - \rho_n F_{nn}
\end{bmatrix}
\begin{bmatrix}
B_1 \\
B_2 \\
\vdots \\
B_n
\end{bmatrix}
= 
\begin{bmatrix}
E_1 \\
E_2 \\
\vdots \\
E_n
\end{bmatrix}
\]

- What do the various terms mean?
Solving Radiosity Equation
Goal

• Find efficient ways to solve the radiosity equation
  – Jacobi Iteration
  – Gauss-Seidel
  – Southwell or Shooting
  – Progressive Radiosity
Radiosity

- **Q:** How many form factors must be computed?
  - **A:** $O(n^2)$

- **Q:** What primarily limits the accuracy of the solution?
  - **A:** The number of patches
Radiosity

• Now “just” need to solve the matrix!
  – Matrix is “diagonally dominant”
  – Thus Gauss-Siedel must converge

• End result: radiosities for all patches

• Solve RGB radiosities separately, color each patch, and render!

• Caveat: actually, color vertices, not patches
Radiosity Equation

\[
\begin{bmatrix}
1 - \rho_1 F_{1,1} & \cdots & \cdots & - \rho_1 F_{1,n} \\
- \rho_2 F_{2,1} & 1 - \rho_2 F_{2,2} & \cdots & - \rho_2 F_{2,n} \\
\vdots & \ddots & \ddots & \vdots \\
- \rho_{n-1} F_{n-1,1} & \cdots & \cdots & - \rho_{n-1} F_{n-1,n} \\
- \rho_n F_{n,1} & \cdots & \cdots & 1 - \rho_n F_{n,n}
\end{bmatrix}
\begin{bmatrix}
B_1 \\
B_2 \\
\vdots \\
B_n
\end{bmatrix}
= 
\begin{bmatrix}
E_1 \\
E_2 \\
\vdots \\
E_n
\end{bmatrix}
\]

- We also need to compute the form factors, \( F_{ij} \)
- Problem is the size of matrices
  \((N*N \text{ for } N \text{ elements, } N \text{ usually } > 50000)\)
Solving for All Patches

• Putting into matrix form
  \[ b = e - RFb \]
  \[ b = [I - RF]^{-1} e \]

• Use matrix algebra to solve for \( B_i \)'s
Solving for All Patches

- One patch defined by:

\[ B_i = \varepsilon_i + \rho_i \sum_{1 \leq j \leq n} B_j F_{j,i} \frac{A_j}{A_i} \]

- Symmetry: \( A_i F_{i,j} = A_j F_{j,i} \)

- Therefore:

\[ B_i - \rho_i \sum_{1 \leq j \leq n} B_j F_{i,j} = \varepsilon_i \]
Solving for All Patches

- Difficult to perform Gaussian Illumination and solve for \( b \) (size of \( F \) is large but sparse – why?)

- Instead, iterate: \( b^{k+1} = e - RFb^k \)
  
  - Multiplication of sparse matrix is \( O(n) \), not \( O(n^2) \)
  
  - Stop when \( b^{k+1} = b^k \)
Solving for All Patches

- **Alternative solution**
  - We know: \( \frac{1}{1-x} = \sum_{i=0}^{\infty} x^i \)
  - Therefore: \( [I - RF]^{-1} = \sum_{i=0}^{\infty} (RF)^i \)
  - And solution for \( b \) is:
    - \( b = \sum_{i=0}^{\infty} (RF)^i e \)
    - \( b = e + (RF)e + (RF)^2 e + (RF)^3 e + \cdots \)
Convergence

- Gauss-Seidel known to converge for diagonally dominant matrices

\[
\begin{bmatrix}
1 - \rho_1 F_{1,1} & \cdot & \cdot & \cdot & - \rho_1 F_{1,n} \\
- \rho_2 F_{2,1} & 1 - \rho_2 F_{2,2} & \cdot & \cdot & - \rho_2 F_{2,n} \\
\cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot \\
- \rho_{n-1} F_{n-1,1} & \cdot & \cdot & \cdot & - \rho_{n-1} F_{n-1,n} \\
- \rho_n F_{n,1} & \cdot & \cdot & \cdot & 1 - \rho_n F_{n,n}
\end{bmatrix}
\begin{bmatrix}
B_1 \\
B_2 \\
\cdot \\
\cdot \\
B_n
\end{bmatrix}
= 
\begin{bmatrix}
E_1 \\
E_2 \\
\cdot \\
\cdot \\
E_n
\end{bmatrix}
\]
Solve by Direct Methods?

- Not feasible to use something like Gaussian elimination because of size of matrix
- We don’t even want to store the matrix
- Use iterative methods
Radiosity

- Where we go from here:
  - Evaluating form factors
  - *Progressive radiosity*: viewing an approximate solution early
  - *Hierarchical radiosity*: increasing patch resolution on an as-needed basis
Iterative Approach

- Define a residual \( r = E - KB \)
- Iterate, computing \( B \), to reduce residual
  \[
  r^{(0)} = E - KB^{(0)}
  \]
- Every iteration, compute new \( B \) and \( r \)
  \[
  r^{(k)} = E - KB^{(k)}
  \]
- Initial Condition
  \[
  B^{(0)} = E
  \]
Method 1: Jacobi Iteration

- Update each element \( B_i^{(k)} \) to the next iteration using the solution vector from the previous iteration.
  
- In other words, compute complete set of \( B \) and use that for next iteration.
Details

- The $i$-th matrix row is

$$\sum_{j=1}^{n} K_{ij} B_j = E_i$$

- Solve for $B_{ii}$

$$K_{ii} B_i = E_i - \sum_{j \neq i} K_{ij} B_j$$
Details

• Recall that

\[ r^{(k)} = E - KB^{(k)} \]

• So

\[ r^{(k)} = E_i - \sum_{j=1}^{n} K_{ij} B_j^{(k)} \]

• or

\[ r^{(k)} = E_i - \sum_{j \neq i} K_{ij} B_j^{(k)} - K_{ii} B_i^{(k)} \]

• and

\[ E_i - \sum_{j \neq i} K_{ij} B_j^{(k)} = r^{(k)} + K_{ii} B_i^{(k)} \]
Substitute

\[ E_i - \sum_{j \neq i} K_{ij} B_j^{(k)} = r^{(k)} + K_{ii} B_i^{(k)} \]

into

\[ K_{ii} B_i = E_i - \sum_{j \neq i} K_{ij} B_j \]

to get

\[ K_{ii} B_i^{(k+1)} = r^{(k)} + K_{ii} B_i^{(k)} \]
or

\[ B_i^{(k+1)} = \frac{r^{(k)}}{K_{ii}} + B_i^{(k)} \]
Jacobi Iteration

• If we compute residual $r$ each iteration, we can compute updated $B$

$$B^{(k+1)}_i = \frac{r^{(k)}}{K_{ii}} + B^{(k)}_i$$

• Works ... but converges slowly

$$r^{(k)} = E_i - \sum_{j=1}^{n} K_{ij}B^{(k)}_j$$
Method 2: Gauss-Seidel

- At each step use the most current values in $B$

\[ K_{ii}B_i^{(k+1)} = E_i - \sum_{j=1}^{i-1} K_{ij}B_j^{(k+1)} - \sum_{j=i+1}^{n} K_{ij}B_j^{(k)} \]

- Analogous formulation to get

\[ B_{i}^{(k+1)} = \frac{r^{(k)}}{K_{ii}} + B_{i}^{(k)} \]

- Now must update residuals at each step
Algorithm

Set all $B_i$ to the $E_i$ values

While (not converged) {
    For (i = 1 to n)
        Compute new $B_i$
}

A full iteration takes $O(n^2)$ – residual update costs $O(n)$ at each step
Method 3: Gathering

• A physical analogy is to think of a node or element as **gathering** light from all of the other elements to arrive at a new estimate.

• Each element $j$ contributes some radiosity to the radiosity of element $i$ as follows:

$$\Delta B_i = \rho_i B_j F_{ij}$$
Gathering variant: Southwell

• Very similar, but instead of proceeding in order from \( I \) to \( n \), choose the row with the highest residual and update it.....

• ...that is, gather to the element which received the least light from what it should
Southwell Algorithm

• For $i$, such that $r_i = \text{Max}(r)$, compute

$$B_{i}^{(k+1)} = E_{i} - \sum_{j\neq i} \frac{K_{ij} B_{j}^{(k)}}{K_{ii}}$$

• Note that, now the variable $k$ is a step and not a complete iteration
Complexity

• In order to keep each step $O(n)$, you need to incrementally update the residuals.
Computing Residual

- Define the difference in radiosity at each step as \( \Delta B^{(p)} \)

- Then

\[
B^{(p+1)} = B^{(p)} + \Delta B^{(p)}
\]

so the residual can be computed as

\[
r^{(p+1)} = E - K(B^{(p)} + \Delta B^{(p)}) = r^{(p)} - K\Delta B^{(p)}
\]
Only One $B$ Changes

- All of the changes in the $B$ vector are 0, except for the one that was just updated at step $I$, so

$$r_j^{(p+1)} = r_j^{(p)} - K_{ji} \Delta B_i, \forall j$$
Initial Conditions

- Set $B^{(0)}$ to all be zero, and $r^{(0)}$ to be $E$
- So at the first step, the element being the brightest emitter would have its radiosity set to the value of that emitter and its residual set to 0
- This leads to the interpretation of . . .
Shooting

- The residual can be interpreted as the amount of energy left to be reflected (or emitted)

- At each step, one of the residuals (the one for row \( i \)) contributes — *shoots* — to all of the other residuals
Progressive Radiosity
(Similar to Southwell)

- Shoot from the element having the most energy
- Compute the form factors as you shoot
- Update all of the radiosities
- Display the results every iteration
Initially

For all $i$ {
$B_i = E_i$;
$\Delta B_i = E_i$;
}

while (not converged) {
    Select \( i \), such that \( \Delta B_i A_i \) is greatest;

    Project all other elements onto Hemicube at \( i \) to compute form factors;

    For every element \( j \) {
        \[
        \Delta Rad = \Delta B_i * \rho_j F_{ji};
        \]
        \[
        \Delta B_j = \Delta Rad;
        \]
        \[
        B_j = \Delta Rad;
        \]
    }

    \( \Delta B_i = 0; \)

    Display image;
}

Advantages

- You see progresses
- You don’t store a $O(n^2)$ matrix of form factors
- When the process starts out, all of the unshot energy is at lights
- As the process unfolds, the energy is spread around and the residuals become more even
Ambient Term

- An estimate of the average form factors can be made from their areas

\[ F_{*j} \approx \frac{A_j}{\sum_{j=1}^{n} A_j} \]

- We can also compute the area-weighted average of reflectivities

\[ \bar{\rho} = \frac{\sum \rho_i A_i}{\sum A_i} \]
Ambient Term

- Just to make the images look better (less dark) at the beginning, Cohen, et. al. use an ambient term.
- It’s related to the reflected illumination not yet accounted for (or in other words the energy yet unshot).
Ambient Estimate

- Ambient term is total of the area-weighted unshot energy times the total reflectivity

\[ B_{ambient} = R_{total} \sum_{j=1}^{n} (\Delta B_j F^*_j) \]

- Each element displays its own fraction

\[ B_{display}^i = B_i^i + \rho_i B_{ambient} \]
Reflection

- The energy will be reflected over and over, so the total reflection can be expressed as

\[ R_{total} = 1 + \bar{\rho} + \bar{\rho}^2 + \bar{\rho}^3 + \ldots = \frac{1}{1 - \bar{\rho}} \]
• 30,000 patches divided into 50,000 elements.
• Solution run for only 2000 patches
• View-dependent post-process, computing radiosity at visible vertices, 190 hours
Displayed Image after 1, 2, 24, and 100 Steps
Magritte Studio Image
Radiosity - Cons

- Form factors need to be re-computed if anything moves
- Large computational and storage costs
- Non-diffuse light not represented
  - Mirrors and shiny objects hard to include
- Lighting effects tend to be “blurry”, not sharp without good subdivision
- Not applicable to procedurally defined surfaces
Radiosity - Pros

• **Viewpoint independence means fast real-time display after initial calculation**

• **Inter-object interaction possible**
  – Soft shadows
  – Indirect lighting
  – Color bleeding

• **Accurate simulation of energy transfer**
View-dependent vs View-independent

- Ray-tracing models specular reflection well, but diffuse reflection is approximated
- Radiosity models diffuse reflection accurately, but specular reflection is ignored
- Advanced algorithms combine the two
Radiosity

• Radiosity is expensive to compute
• Some parts of illuminated world can change
  – Emitted light
  – Viewpoint
• Other things cannot
  – Light angles
  – Object positions and occlusions
  – Computing form factors is expensive
• Specular reflection information is not modeled
Summary

• **Now we know**
  – How to formulate the radiosity problem
  – How to solve equations
  – How to approximate form factors
References