Radiosity estimates via finite elements

D.A. Forsyth
after slides by John Hart

In a world of diffuse surfaces ...

- Recall
  - radiosity is radiated power per unit area, independent of direction
  - we obtained:

$$B(\mathbf{x}) = E(\mathbf{x}) + \rho(\mathbf{x}) \int_S \frac{\cos\theta_i \cos\theta_s}{\pi r^2} Vis(\mathbf{x}, \mathbf{u}) B(\mathbf{u}) dA_s$$

  - which we wrote as:

$$B(\mathbf{x}) - E(\mathbf{x}) - \rho(\mathbf{x}) \int K(\mathbf{x}, \mathbf{u}) B(\mathbf{u}) dA_{\mathbf{u}} = 0$$

Radiosity estimate via finite elements

- Divide domain into patches
- Radiosity will be constant on each patch
  - patch basis function, or element

$$\phi_i(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ in patch } i \\ 0 & \text{otherwise} \end{cases}$$

- Now write
  - B_i for radiosity at patch i
  - E_i for exitance at patch i
  - Substitute into eqn:

$$B(\mathbf{x}) - E(\mathbf{x}) - \rho(\mathbf{x}) \int K(\mathbf{x}, \mathbf{u}) B(\mathbf{u}) dA_{\mathbf{u}} = 0$$

Becomes

$$\left(\sum_i B_i \phi_i(\mathbf{x}) \right) - \left(\sum_i E_i \phi_i(\mathbf{x}) \right) - \left(\rho(\mathbf{x}) \int K(\mathbf{x}, \mathbf{u}) \left(\sum_i B_i \phi_i(\mathbf{u}) \right) dA_{\mathbf{u}} \right) = R(\mathbf{x})$$

$\uparrow$

This should be “like zero”

Obtaining an estimate: Finite elements

- But in what sense is it zero?
  - Galerkin method

$$\int R(\mathbf{x})\phi_k(\mathbf{x})dA_x = 0 \forall k$$

- Apply to:

$$\left(\sum_i B_i\phi_i(\mathbf{x})\right) - \left(\sum_i E_i\phi_i(\mathbf{x})\right) - \left(\rho(\mathbf{x})\int K(\mathbf{x},\mathbf{u})\left(\sum_i B_i\phi_i(\mathbf{u})\right)dA_{\mathbf{u}}\right) = R(\mathbf{x})$$

- And get

$$B_k A_k - E_k A_k - \sum_j \left(\int_{\text{patch } k} \rho(\mathbf{x}) \int_{\text{patch } j} K(\mathbf{x},\mathbf{u})d\mathbf{u}d\mathbf{x}\right) B_j = 0$$

Finite Element Radiosity Equation

- Start with:

$$B_k A_k = E_k A_k + \sum_j \left(\int_{\text{patch } k} \rho(\mathbf{x}) \int_{\text{patch } j} K(\mathbf{x}, \mathbf{u}) d\mathbf{u} d\mathbf{x} \right) B_j$$

- Divide through by A_k, assume constant albedo patches, get

$$B_k = E_k + \sum_j \rho_k F_{jk} B_j$$

- Where geometric effects are concentrated in the form factor

$$F_{jk} = \frac{1}{A_k} \int_{\text{patch } k} \int_{\text{patch } j} K(\mathbf{x}, \mathbf{u}) d\mathbf{u} d\mathbf{x}$$

Finite Element Radiosity

- This is a linear system

$$B_k = E_k + \sum_j \rho_k F_{jk} B_j$$

- fold in albedo, write

$$B_k = E_k + \sum_j \Gamma_{kj} B_j$$

- or in terms of matrices and vectors

$$\mathbf{B} = \mathbf{E} + \Gamma \mathbf{B}$$

- BUT YOU SHOULD NEVER DO:
  - B might have 10^6 elements or more!

$$\mathbf{B} = (\mathcal{I} - \Gamma)^{-1} \mathbf{E}$$

Form factors

- Recall: $$F_{jk} = \frac{1}{A_k} \int_{\text{patch } k} \int_{\text{patch } j} K(\mathbf{x}, \mathbf{u}) d\mathbf{u} d\mathbf{x}$$
- if patches are all flat, then: $$F_{ii} = 0$$
- if i can't see j at all, then: $$F_{ij} = 0$$
- reciprocity: $$A_k F_{jk} = A_j F_{kj}$$

Form Factors

- Power leaving patch k:

$$B_k A_k$$

- Power leaving patch k for patch j:

$$\int_{\text{patch } k} \int_{\text{patch } j} K(\mathbf{x}, \mathbf{u}) B_k \, d\mathbf{u} \, d\mathbf{x}$$

- Interpretation:
  - Fjk is percentage of power leaving k that arrives at j

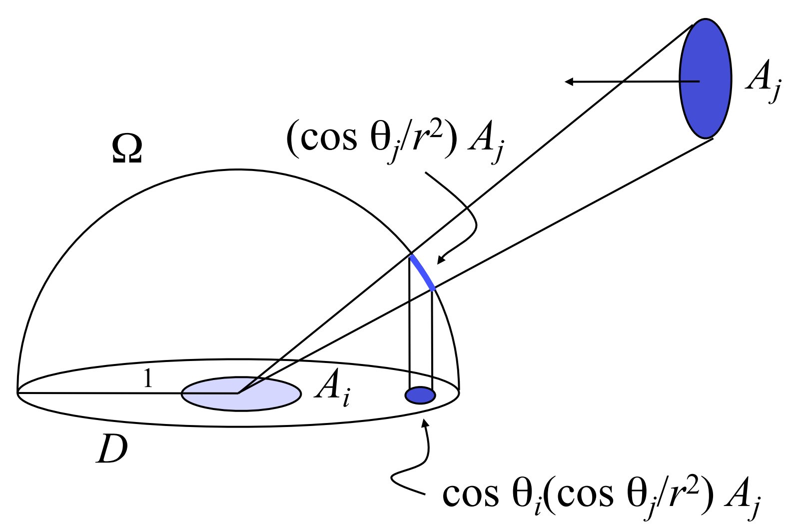
$$F_{jk} = \frac{1}{A_k} \int_{\text{patch } k} \int_{\text{patch } j} K(\mathbf{x}, \mathbf{u}) \, d\mathbf{u} \, d\mathbf{x}$$

  - this gives:

$$\sum_j F_{jk} = 1$$

Computing form factors

- Nusselt's analogy

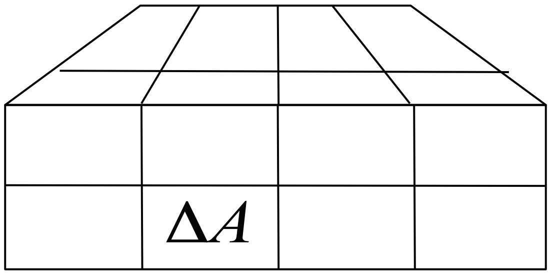


$$F_{ij} = \mathrm{proj}_D(\mathrm{proj}_\Omega(A_j))/\mathrm{Area}(D)$$

The Hemicube

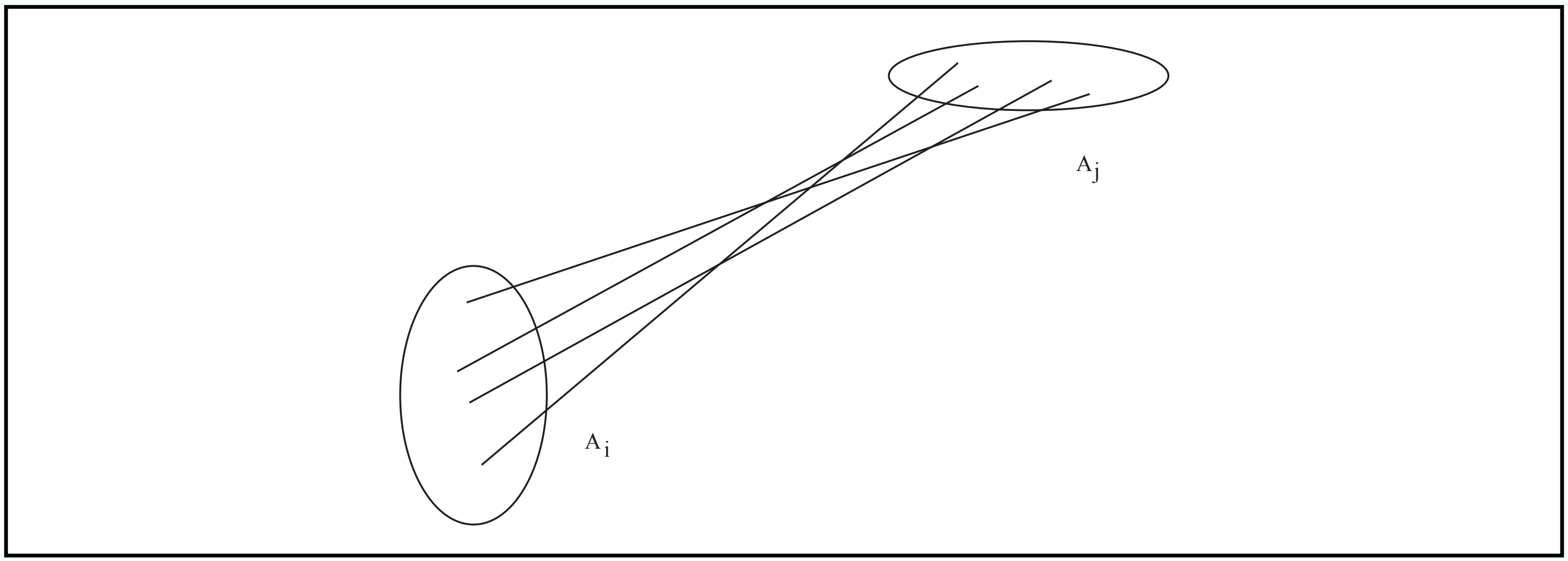
- Render onto faces of cube on receiver

$$\Delta F_{dAiAj} = \frac{\cos\phi_i \cos\phi_j}{\pi r^2} \Delta A$$



Random samples

- with N uniform samples on patches j and k



$$A_j A_k F_{jk} \approx \frac{1}{N} \sum \frac{\cos\theta_i \cos\theta_j Vis(i,j)}{\pi r^2}$$

Finite Element Radiosity

- This is a linear system

$$B_k = E_k + \sum_j \rho_k F_{jk} B_j$$

- fold in albedo, write

$$B_k = E_k + \sum_j \Gamma_{kj} B_j$$

- or in terms of matrices and vectors

$$\mathbf{B} = \mathbf{E} + \Gamma \mathbf{B}$$

- BUT YOU SHOULD NEVER DO:
  - B might have 10^6 elements or more!

$$\mathbf{B} = (\mathcal{I} - \Gamma)^{-1} \mathbf{E}$$

Solving the radiosity system: Gathering

- Neumann series (again!)

$$\mathbf{B} = \mathbf{E} + \Gamma\mathbf{E} + \Gamma^2\mathbf{E} + \Gamma^3\mathbf{E} + ...$$

- Easy iteration

$$\mathbf{B}^{(0)} = \mathbf{E}$$

$$\mathbf{B}^{(n+1)} = \mathbf{E} + \Gamma\mathbf{B}^{(n)}$$

Not a good idea in this form, because we must evaluate the whole of Gamma for EACH iteration;
Gamma might be millions by millions

Gathering with iterative methods

- Linear system   Ax=b

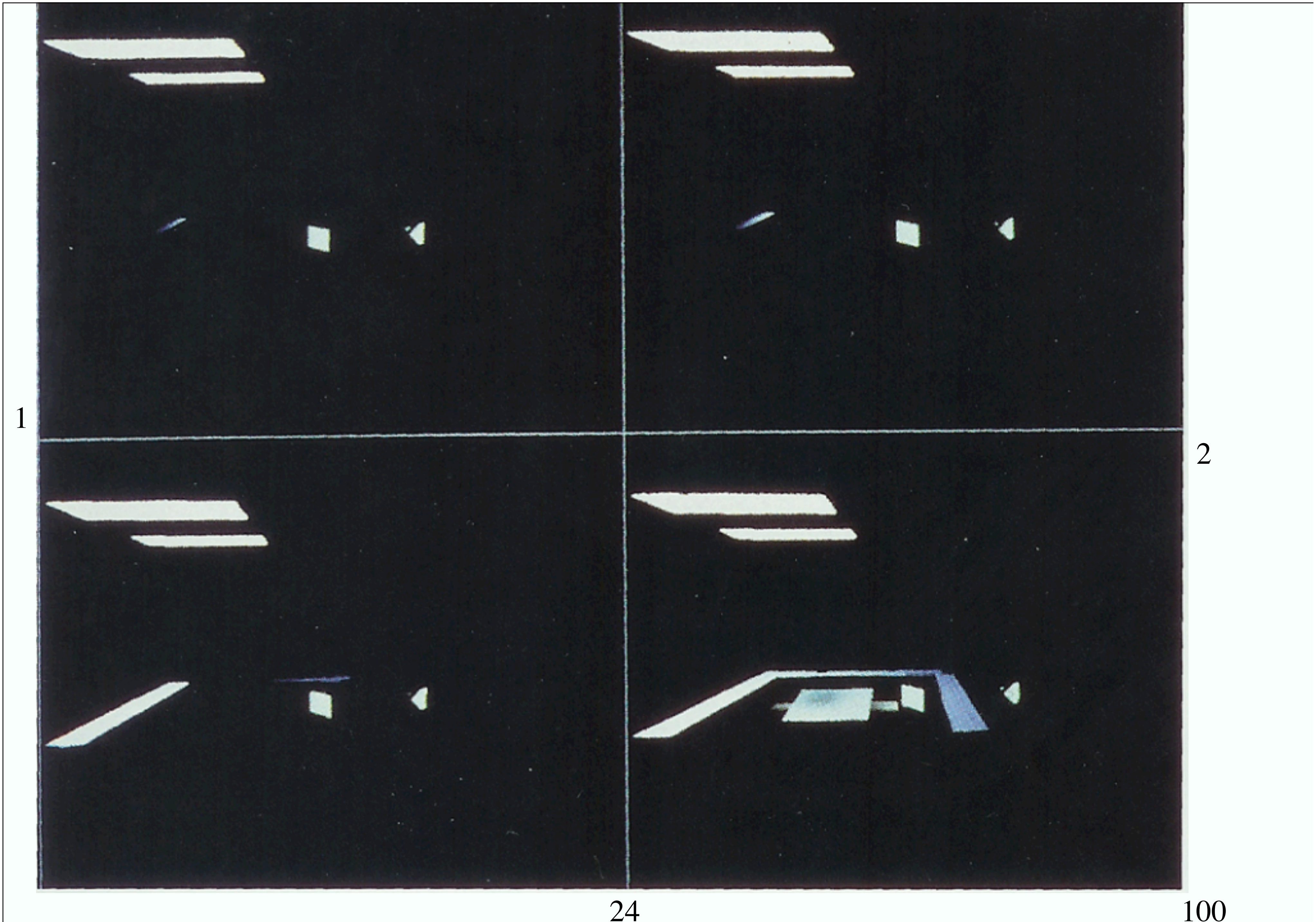
$$\sum_j a_{ij} x_j = b_i$$

- Jacobi iteration
  - reestimate each x

$$x_j^{(n+1)} = \frac{1}{a_{jj}} \left(b_i - \sum_{l \neq j} a_{il} x_l^{(n)} \right)$$

- Gauss-Seidel
  - reuse new estimates

$$x_j^{(n+1)} = \frac{1}{a_{jj}} \left(b_i - \sum_{l<j} a_{il} x_l^{(n+1)} - \sum_{l>j} a_{il} x_l^{(n)} \right)$$



From Cohen, SIGGRAPH 88

Southwell iteration: Progressive radiosity

- Gauss-Seidel, Jacobi, Neumann require us to evaluate whole kernel at each iteration
  - this is vilely expensive 10^6x 10^6 matrix?
  - it's also irrational
    - in G-S, Jacobi, for one pass through the variables,
      - we gather at each patch, from each patch
        - but some patches are not significant sources
    - we should like to gather only from bright patches
      - or rather, patches should "shoot"
- This is Southwell iteration

Southwell iteration: update x

- Define a residual: $$R = (b - Ax)$$
  - whose elements are $$r_i^{(n)} = b_i - \sum_j a_{ij} x_j^{(n)}$$
- now choose the largest r_i
  - and adjust the corresponding x component to make it zero

$$r_i^{(n+1)} = 0$$

$$x_l^{(n+1)} = \left\{ \begin{array}{cc} x_l^{(n)} & \text{if } l \neq i \\ \frac{1}{a_{ii}} \left(r_i^{(n)} + a_{ii} x_i^{(n)} \right) & \text{if } l = i \end{array} \right\}$$

Southwell iteration: update r

- Update the residual by adding old x col, subtracting new

$$\boxed{r_l^{(n+1)} = r_l^{(n)} + a_{li}(x_i^{(n)} - x_i^{(n+1)})}$$

- but this takes an easy form

$$\boxed{r_l^{(n+1)} = r_l^{(n)} - \frac{a_{li}}{a_{ii}} r_i^{(n)}}$$

- Notice we can update variables in order of large residual, using only one col of kernel to do so
  - this converges (non-trivial) rather fast (non-trivial)
  - to get a solution, we need evaluate only a small proportion of the kernel (non-trivial)

Applying Southwell iteration to radiosity

- Our linear system is:

$$(\mathcal{I} - \Gamma)\mathbf{B} = \mathbf{E}$$

- And so we can write the residual as:

$$\mathbf{r}^{(n)} = \mathbf{E} - \mathbf{B}^{(n)} + \Gamma\mathbf{B}^{(n)}$$

- Interpretation:
  - update B at i'th entry
  - at every other entry, we add energy shot from this update to that location
  - therefore residual is energy received, but not yet shot
    - which is zero, eventually

Applying Southwell iteration to radiosity

- Introduce a new variable:

$$\mathbf{N}^{(n)} = \mathbf{B}^{(n)} + \mathbf{r}^{(n)}$$

- Notice
  - when iteration converges, N=B
  - N is: current estimate of radiosity+unshot radiosity
    - so N is a better rendering estimate than B
- N is easy to update
  - need only a column of matrix
  - use equations on following page
  - small r=small N-B

Applying Southwell iteration to radiosity

$$\Delta B = \frac{r_i^{(n)}}{(1 - \Gamma_{ii})}$$

$$B_j^{(n+1)} = \begin{cases} B_j^{(n)} + \Delta B & \text{if } j = i \\ B_j^{(n)} & \text{if } j \neq i \end{cases}$$

$$r_j^{(n+1)} = \begin{cases} 0 & \text{if } j = 1 \\ r_j^{(n)} - \Gamma_{ji} \Delta B & \text{otherwise} \end{cases}$$

$$N_j^{(n+1)} = \begin{cases} B_j^{(n)} + \Delta B & \text{if } j = 1 \\ B_j^{(n)} + r_j^{(n)} - \Gamma_{ji} \Delta B & \text{otherwise} \end{cases}$$

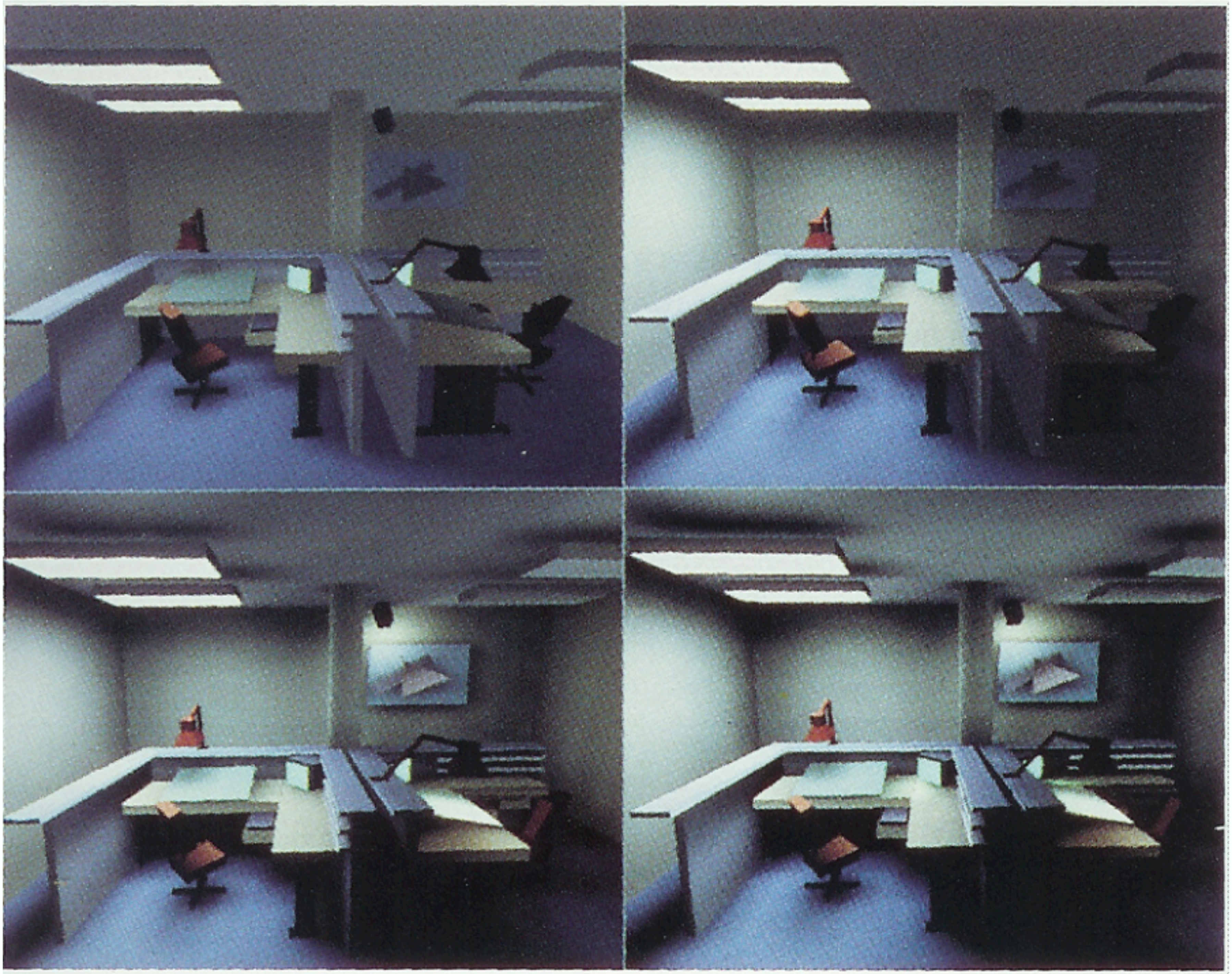
Applying Southwell iteration to radiosity

$$\Delta B = \frac{N_i^{(n)} - B_i^{(n)}}{(1 - \Gamma_{ii})}$$

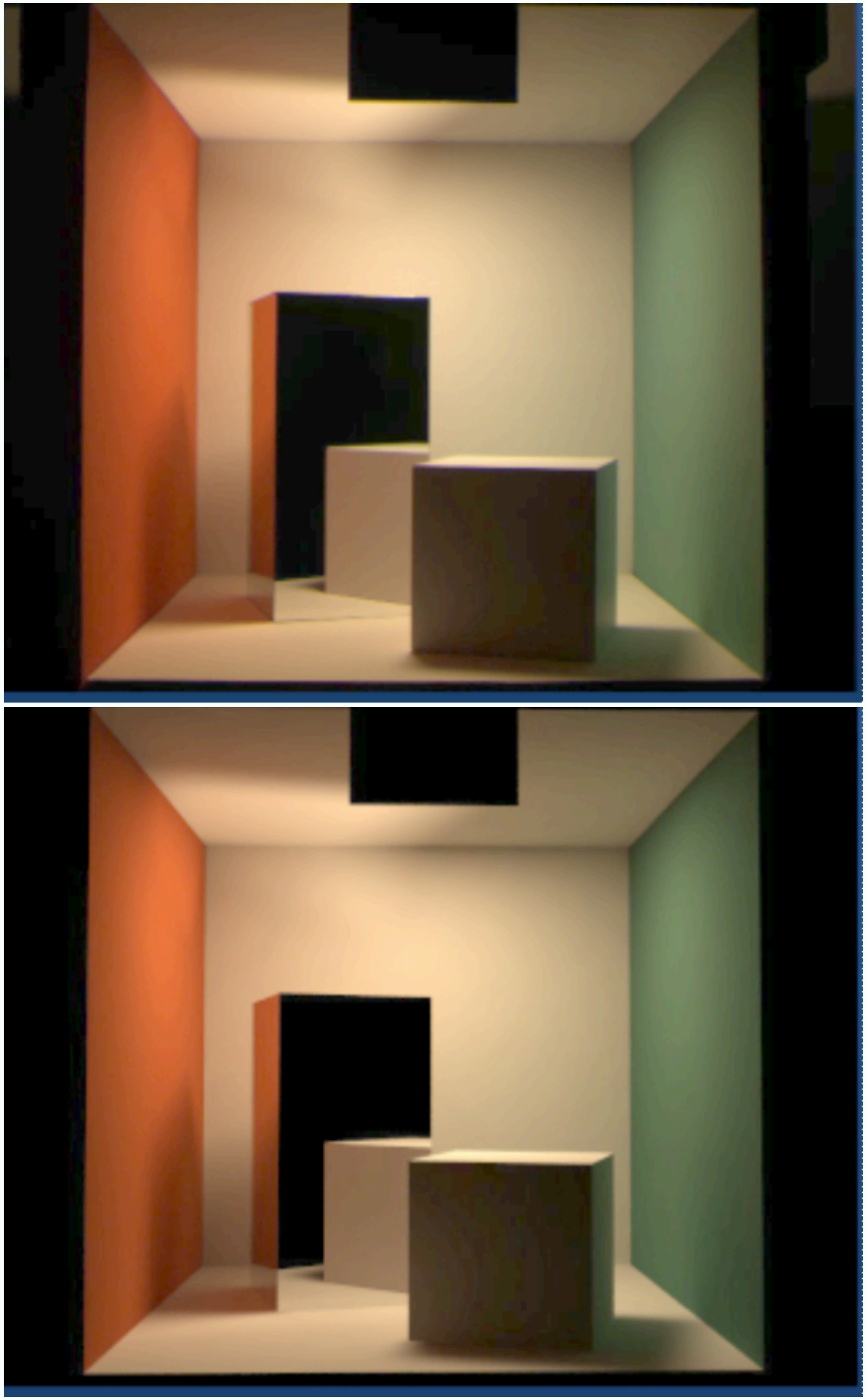
$$B_j^{(n+1)} = \begin{cases} B_j^{(n)} + \Delta B & \text{if } j = i \\ B_j^{(n)} & \text{if } j \neq i \end{cases}$$

$$N_j^{(n+1)} = \begin{cases} B_j^{(n)} + \Delta B & \text{if } j = 1 \\ N_j^{(n)} - \Gamma_{ji} \Delta B & \text{otherwise} \end{cases}$$

And check N-B rather than r to choose i!



From Cohen, SIGGRAPH 88



Cornell Program of Computer Graphics

Hierachical radiosity

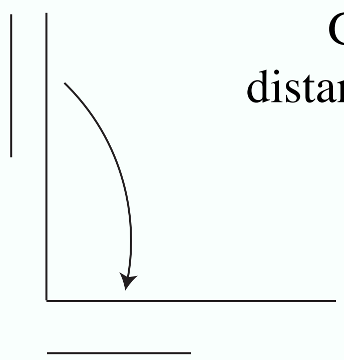
- Radiosity similar to n-body problems
  - gathering can be grouped
- Recall iteration

$$\mathbf{B}^{(0)} = \mathbf{E}$$

$$\mathbf{B}^{(n+1)} = \mathbf{E} + \Gamma \mathbf{B}^{(n)}$$

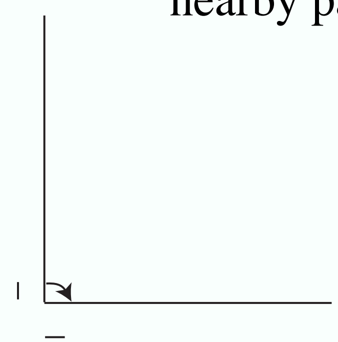
- Can we make matrix multiplication more efficient?
  - Gamma “gathers” old radiosity solution to each patch
  - But distant patches contribute a near constant value
    - so when we gather from distant patches, we should use a big receiver

Alternative meshes



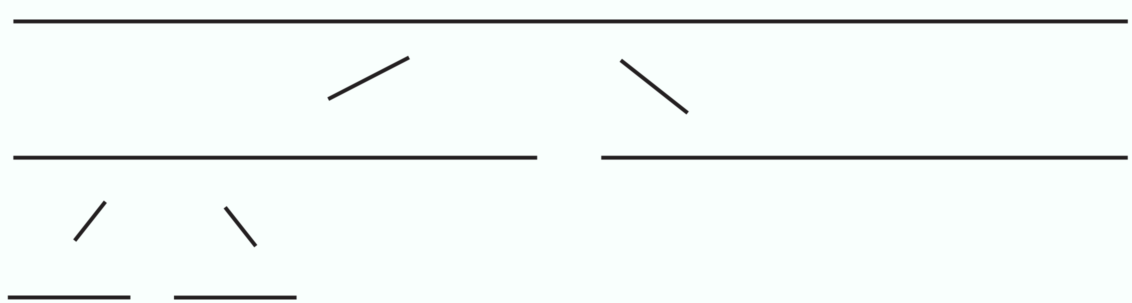
Gathering from
distant patch in a corner

Gathering from
nearby patch in a corner



A mesh hierarchy

- Represent patch with big AND small elements
  - big elements gather from distant
  - small elements gather from nearby
  - how do we know element is small enough
    - check size
    - check FF
    - check radiosity*FF
- Rendering
  - we need to know the radiosity at a point
    - walk the point down hierarchy
    - radiosity is radiosity of smallest element containing point



A mesh hierarchy

- Recall
  - radiosity is power /unit area
- Procedure
  - build initial mesh
  - until (no fixing)
    - until (converged)
      - compute a term in neumann series by
        - elements gather radiosity
        - distribute across the hierarchy
    - check whether mesh is fine enough

This is radiosity we have gathered, but haven't accounted for yet

This is the radiosity of the element

```
struct Quadnode {
    float              B_g;       /* gathering radiosity */
    float              B_s;       /* shooting radiosity */
    float              E;         /* emission */
    float              area;
    float              ρ;
    struct Quadnode**  children;  /* pointer to list of four children */
    struct Linknode*   L;         /* first gathering link of node */
};

struct Linknode {
    struct Quadnode*   q;         /* gathering node */
    struct Quadnode*   p;         /* shooting node */
    float              F_qp;      /* form factor from q to p */
    struct Linknode*   next;      /* next gathering link of node q */
};
```

Figure 7.7: **Quadnode** *and* **Linknode** *data structures*

Root code for solving; assume all surfaces are polygons

```
HierarchicalRad(float BFε)
{
    Quadnode *p, *q;
    Link *L;
    int Done = FALSE;
    for ( all surfaces p ) p→Bs = p→E;
    for ( each pair of surfaces p, q )
        Refine( p, q, BFε );
    while ( not Done ) {
        Done = TRUE;
        SolveSystem();     /* as in Figure 7.9 */
        for ( all links L )
            /* RefineLink returns FALSE if any subdivision occurs */
            if( RefineLink( L, BFε ) == FALSE )
                Done = FALSE;
    }
}
```

Make the mesh hierarchy

Solve using mesh hierarchy

If there is evidence this hierarchy is not fine enough somewhere, refine and go again

```
Refine(Quadnode *p, Quadnode *q, float F_ε )
{
    Quadnode which, r;
    if ( Oracle1( p, q, F_ε ) )
        Link( p, q );
    else {
        which = Subdiv( p, q );
        if( which == q )
            for( each child node r of q ) Refine( p, r, F_ε );
        else if ( which == p )
            for( each child node r of p ) Refine( r, q, F_ε );
        else
            Link( p, q );
    }
}
```

Check which side should be split
for example, split larger area

Compute the form factor for p, q by casting
random rays (as above) then put it in the
appropriate spot in datastructures

Figure 7.8: `Refine` *pseudocode.*

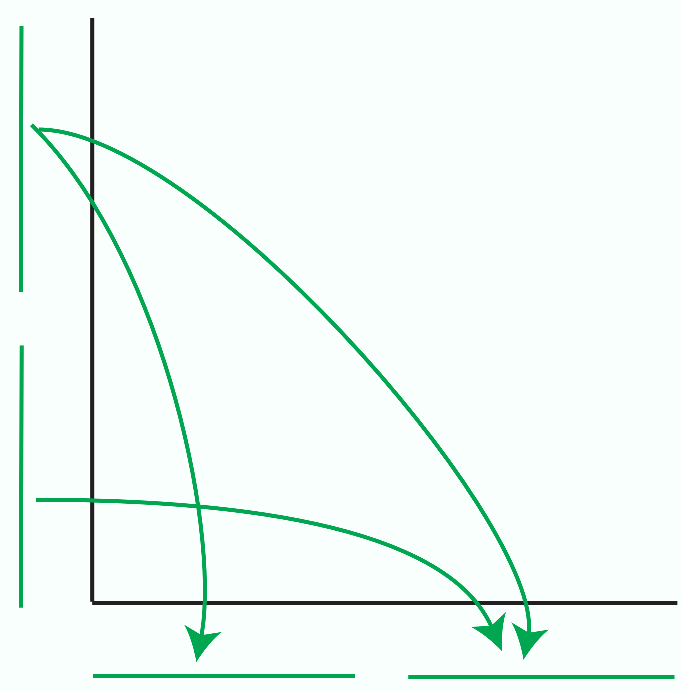
```
SolveSystem()
{
    Until Converged {
        for ( all surfaces p) GatherRad( p );
        for ( all surfaces p) PushPullRad( p, 0.0 ); }
}
```

Gather radiosity across link

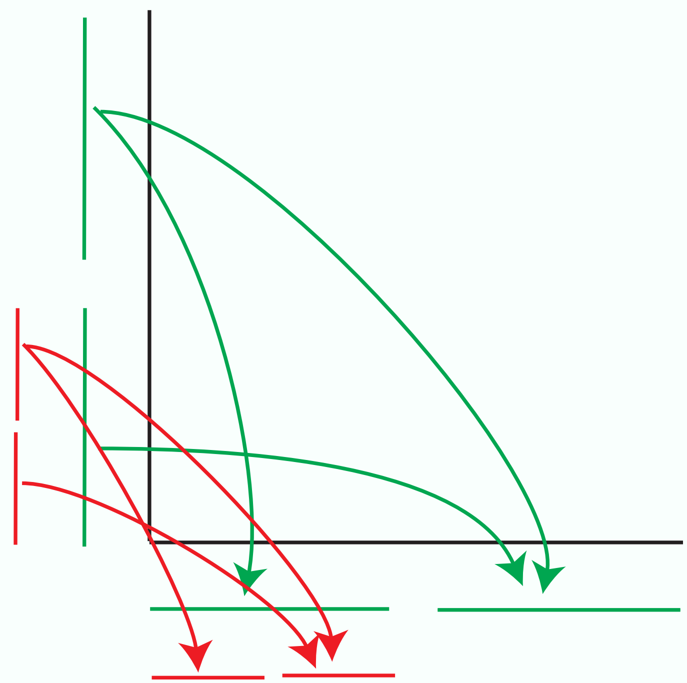
Adjust values in hierarchy so they're consistent

Figure 7.9: `SolveSystem` *pseudocode.*

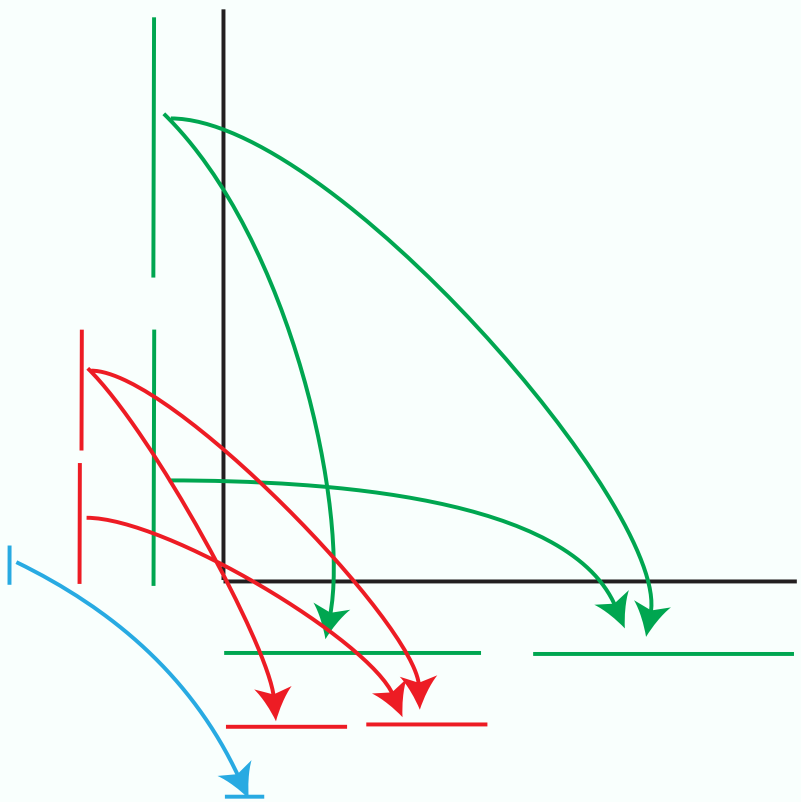
Gathering radiosity



Gathering radiosity



Gathering radiosity



```
GatherRad( Quadnode *p )
{
    Quadnode *q; Link *L;

    p→B_g = 0;
    for ( each gathering link L of p ) /* gather energy across link */
        p→B_g += p→ρ * L→F_pq * L→q→B_s ;
    for each child node r of p
        GatherRad( r );
}
```

Notice that we gather from B_s into B_g

Figure 7.10: `GatherRad` *pseudocode.*

PushPullRad(Quadnode $*p$, float B_{down})

{

float B_{up}, B_{tmp};

if ($p \rightarrow children == NULL$) /* p is a leaf */

$B_{up} = p \rightarrow E + p \rightarrow B_g + B_{down}$;

else

{

$B_{up} = 0$;

for (each child node r of p)

{

$B_{tmp} =$ **PushPullRad**$(r, p \rightarrow B_g + B_{down})$;

$B_{up} \mathrel{+}= B_{tmp} * \frac{r \rightarrow area}{p \rightarrow area}$

}

}

$p \rightarrow B_s = B_{up}$;

return B_{up};

}

Radiosity is power/unit area
so parent adds to children,
children add area weighted sum to parent

Figure 7.11: **PushPullRad** *pseudocode.*

```
float Oracle1( Quadnode *p, Quadnode *q, float Fε )
{
    if ( p→area < A_ε and q→area < A_ε )
        return( FALSE );
    if ( EstimateFormFactor( p, q ) < Fε )
        return( FALSE );
    else
        return( TRUE );
}
```

Figure 7.12: `Oracle1` *pseudocode.*

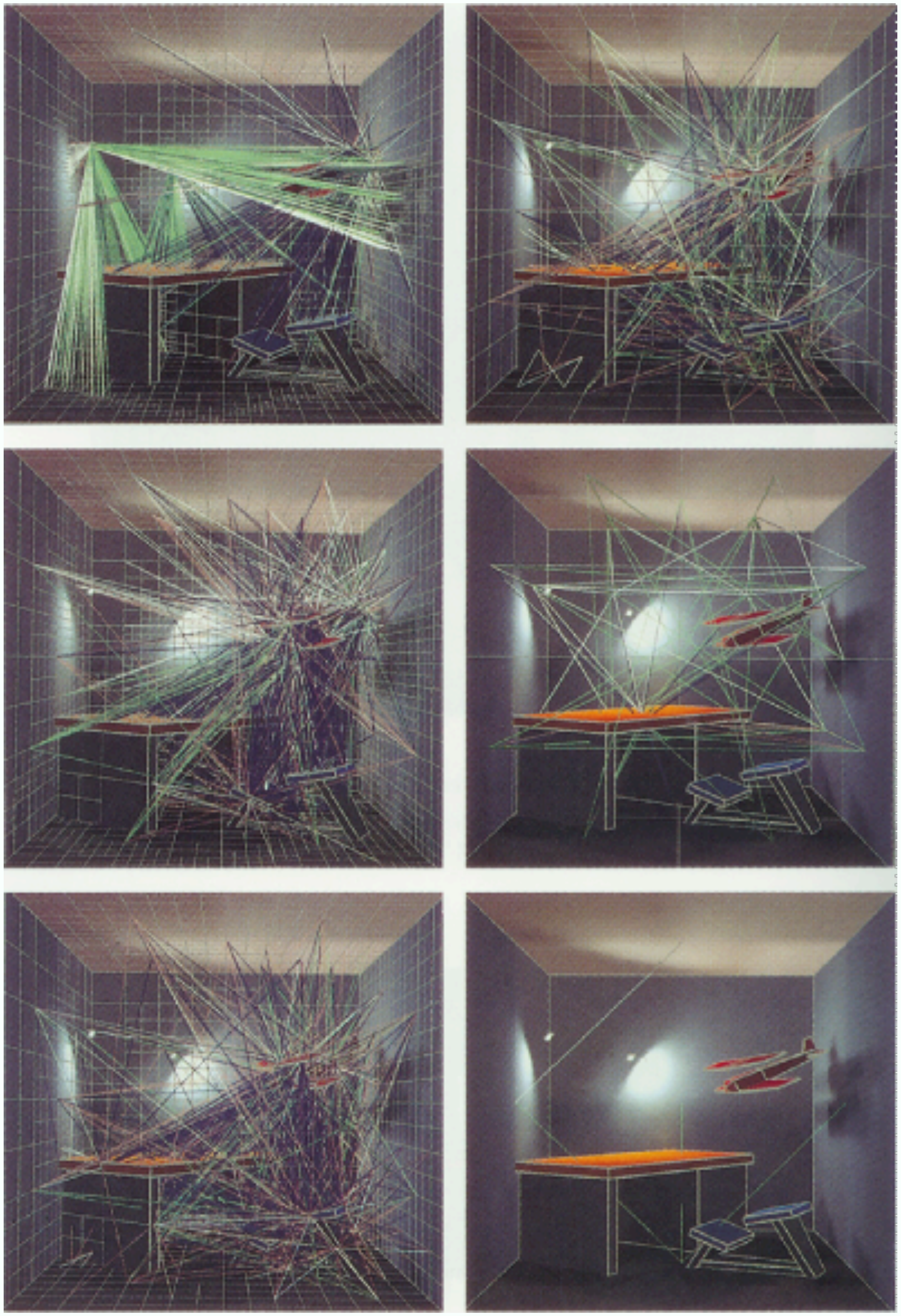
```
int RefineLink(Linknode *L, float BFε)
{
    int no_subdivision = TRUE;
    Quadnode* p = L→p ;      /* shooter */
    Quadnode* q = L→q ;      /* receiver */

    if ( Oracle2(L, BFε ) {
        no_subdivision = FALSE ;
        which = Subdiv( p, q );
        DeleteLink( L );
        if ( which == q )
            for (each child node r of q) Link( p, r );
        else
            for (each child node r of p) Link( r, q );
    }
    return(no_subdivision);
}
```

Figure 7.15: `RefineLink` *pseudocode.*

```
float Oracle2( Linknode *L, float BF_ε )
{
    Quadnode* p = L→p ;     /* shooter */
    Quadnode* q = L→q ;     /* receiver */
    if ( p→area < A_ε and q→area < A_ε )
        return( FALSE );
    if ( p→B_s == 0.0 )
        return( FALSE );
    if( (p→B_s * p→Area * L→F_pq) < BF_ε );
        return( FALSE );
    else return( TRUE );
}
```

Figure 7.16: `Oracle2` *pseudocode.*



BIF links, from Hanrahan et al, 91