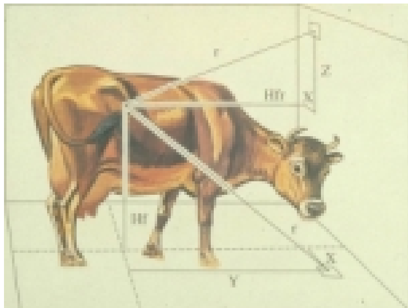
Radiosity



References:

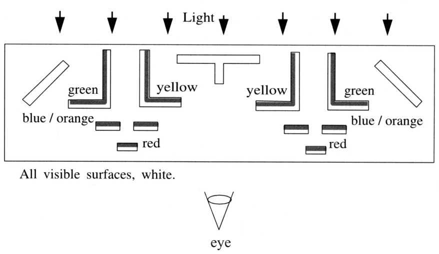
Cohen and Wallace, *Radiosity and Realistic Image Synthesis*

Sillion and Puech, *Radiosity and Global Illumination*

Thanks to François Sillion for images

An early application of radiative heat transfer in stables.

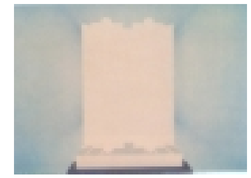
Why Radiosity?



A powerful demonstration introduced by Goral et al. of the differences between radiosity and traditional ray tracing is provided by a sculpture by John Ferren. The sculpture consists of a series of vertical boards painted white on the faces visible to the viewer. The back faces of the boards are painted bright colors. The sculpture is illuminated by light entering a window behind the sculpture, so light reaching the viewer first reflects off the colored surfaces, then off the white surfaces before entering the eye. As a result, the colors from the back boards "bleed" onto the white surfaces.

Radiosity vs. Ray Tracing

Original sculpture lit by daylight from the rear.



Ray traced image. A standard Ray tracer cannot simulate the interreflection of light between diffuse Surfaces.

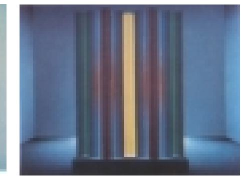
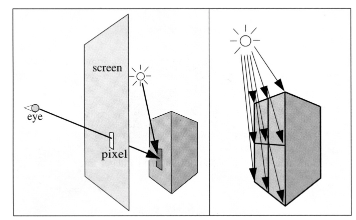


Image rendered with radiosity. note color bleeding effects.

Ray Tracing vs. Radiosity



Ray tracing is an *image-space* algorithm, while radiosity is computed in *object-space*.

Because the solution is limited by the view, ray tracing is often said to provide a *view-dependent solution*, although this is somewhat misleading in that it implies that the radiance itself is dependent on the view, which is not the case. The term *view-independent* refers only to the use of the view to limit the set if locations and directions for which the radiance is computed.

Radiosity Introduction

The radiosity approach to rendering has its basis in the theory of heat transfer. This theory was applied to computer graphics in 1984 by Goral et al.

Surfaces in the environment are assumed to be perfect (or Lambertian) diffusers, reflectors, or emitters. Such surfaces are assumed to reflect incident light in all directions with equal intensity.

A formulation for the system of equations is facilitated by dividing the environment into a set of small areas, or *patches*. The radiosity over a patch is constant.

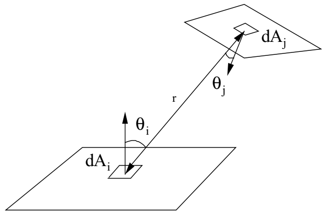
The radiosity, B, of a patch is the total rate of energy leaving a surface and is equal to the sum of the emitted and reflected energies:

Interchange Between Patches

We can set up an equation that relates the energy reflected from a patch to any self-emitted energy plus the energy incoming from all other patches as follows:

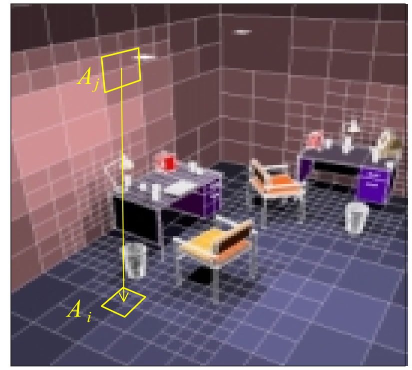
$$B_i dA_i = E_i dA_i + \rho_i \int_j B_j dA_j F_{dA_j dA_i}$$

Radiosity x area = emitted energy + reflected energy



Radiosity Equation

For an environment that has been discretized into n patches, over which the radiosity is constant, (i.e. both B and E are constant across a patch), we have the basic radiosity relationship:



reflectivity

$$B_i = E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j$$

Form factor

- discrete representation
- iterative solution
- costly geometric/visibility calculations

The Radiosity Matrix

Such an equation exists for each patch, and in a closed environment, a set of n simultaneous equations in n unknown B_i values is obtained:

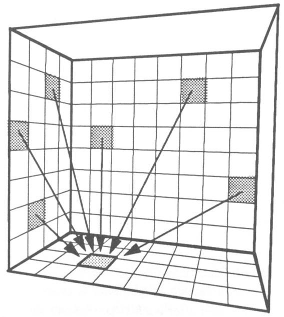
$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & & \\ \vdots & & \ddots & \\ -\rho_n F_{n1} & \cdots & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots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

A solution yields a single radiosity value B_i for each patch in the environment – a view-independent solution. The B_i values can be used in a standard renderer and a particular view of the environment constructed from the radiosity solution.

Standard Solution of the Radiosity Matrix

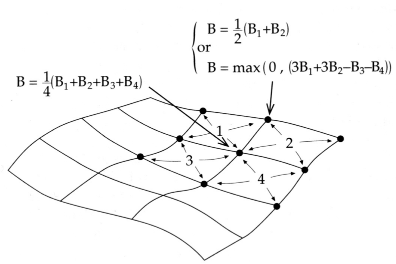
The radiosity of a single patch i is updated for each iteration by *gathering* radiosities from all other patches:

$$\begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_i \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_i \\ \vdots \\ E_n \end{bmatrix} + \begin{bmatrix} & & & \\ \rho_i F_{i1} & \rho_i F_{i1} & \cdots & \rho_i F_{in} \\ & & & \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_i \\ \vdots \\ B_n \end{bmatrix}$$



Computing Vertex Radiosities

- Recall that radiosity values are constant over the extent of a patch.
- A standard renderer requires vertex radiosities (intensities). These can be obtained for a vertex by computing the average of the radiosities of patches that contribute to the vertex under consideration.
- Vertices on the edge of a surface can be allocated values by extrapolation through interior vertex values, as shown on the right:



Stages in a Radiosity Solution

Input of scene geometry

Form factor calculation

Input of reflectance properties

Solution to the system of equations

Radiosity solution

Viewing conditions

Visualization

Radiosity image

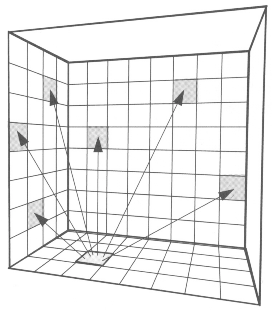
Progressive Refinement

- The idea of progressive refinement is to provide a quickly rendered image to the user that is then gracefully refined toward a more accurate solution. The radiosity method is especially amenable to this approach.
- The two major practical problems of the radiosity method are the storage costs and the calculation of the form factors.
- The requirements of progressive refinement and the elimination of precalculation and storage of the form factors are met by a restructuring of the radiosity algorithm.
- The key idea is that the entire image is updated at every iteration, rather than a single patch.

Reordering the Solution for PR

Shooting: the radiosity of all patches is updated for each iteration:

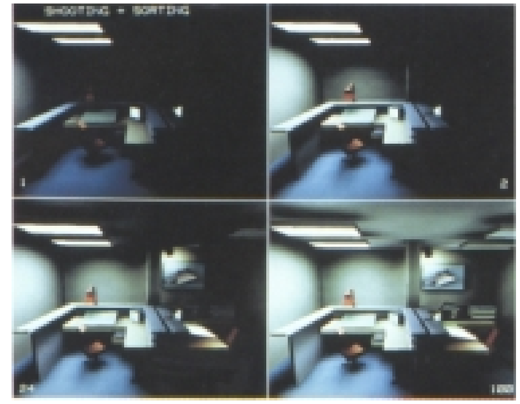
$$\begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ \vdots \\ B_n \end{bmatrix} + \begin{bmatrix} \cdots & \rho_1 F_{1i} & \cdots \\ \cdots & \rho_2 F_{2i} & \\ & & \\ & & \\ \cdots & \rho_n F_{ni} & \cdots \end{bmatrix} \begin{bmatrix} \vdots \\ B_i \\ \vdots \end{bmatrix}$$



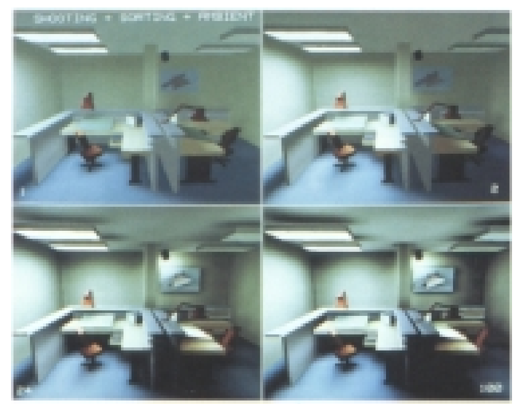
Progressive Refinement Pseudocode

```
while(not converged){
   pick i, such that ΔB_i * A_i is largest;
   for (every element){
      Δrad = ΔB_i * ρ_j F_ji ;
      ΔB_j = ΔB_j + Δrad;
      B_j = B_j + Δrad;
   }
   ΔB_i = 0
   display image using B_i as the intensity of element i;
}
```


Progressive Refinement w/out Ambient Term

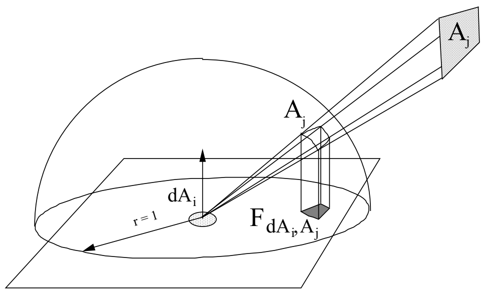


Progressive Refinement with Ambient Term



Form Factor Determination

The Nusselt analog: the form factor of a patch is equivalent to the faction of the the unit circle that is formed by taking the projection of the patch onto the hemisphere surface and projecting it down onto the circle.



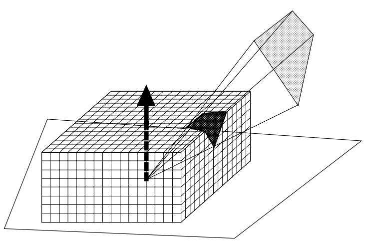
Hemicube Algorithm

A hemicube is constructed around the center of each patch. (Faces of the hemicube are divided into 'pixels'.)

We project a patch onto the faces of the hemicube. The form factor is determined by summing the pixels onto which the patch projects

Occlusion is handled by comparing distances of patches that project onto the same hemicube pixels.

Simultaneously offers an efficient (though approximate) method of form factor determination and a solution to the occlusion problem between patches.

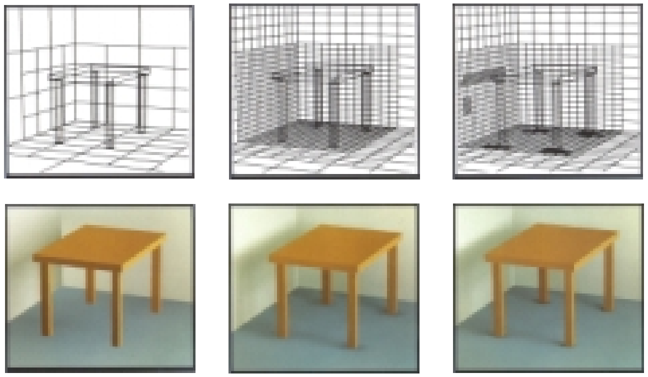


Increasing the Accuracy of the Solution

- ✔The quality of the image is a function of the size of the patches.
- ✔In regions of the scene, such as shadow boundaries, that exhibit a high radiosity gradient, the patches should be subdivided. We call this *adaptive subdivision*.
- ✔The basic idea is as follows: Compute a solution on a uniform initial mesh; the mesh is then refined by subdividing elements that exceed some error tolerance.

What's wrong with this picture?

Adaptive Subdivision of Patches



Coarse patch solution (145 patches)

Improved solution (1021 subpatches)

Adaptive subdivision (1306 subpatches)

Adaptive Subdivision Pseudocode

```
Adaptive_subdivision (error_tolerance) {
   Create initial mesh of constant elements;
   Compute form factors;
   Solve linear system;
   do until (all elements within error tolerance
                  or minimum element size reached) {
        Evaluate accuracy by comparing adjacent element radiosities;
        Subdivide elements that exceed user-specified error tolerance;
        for (each new element) {
            Compute form factors from new element to all other elements;
            Compute radiosity of new element based on old radiosity values;
        }
   }
}
```


Structure of the Solution

- ✔ Calculation of form factors (> 90 %)
- ✔ Solution to the system of equations (< 10 %)
- ✔ Rendering the image (0 %)

Input of scene geometry

Form factor calculation

Input of reflectance properties

Solution to the system of equations

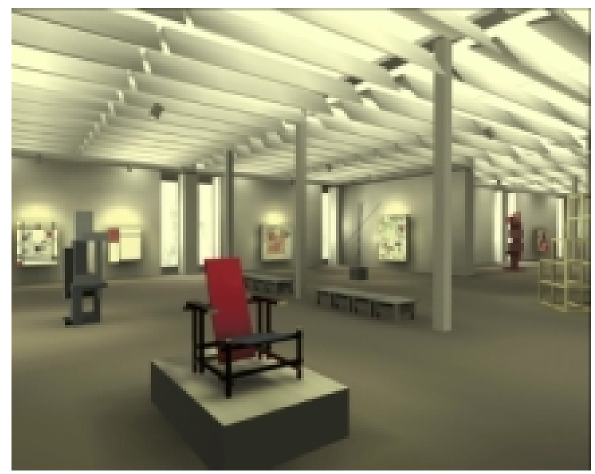
Radiosity solution

Viewing conditions

Visualization

Radiosity image

Factory simulation. Program of Computer Graphics, Cornell University. 30,000 patches.



Museum simulation. Program of Computer Graphics, Cornell University. 50,000 patches. Note indirect lighting from ceiling.

Next Time