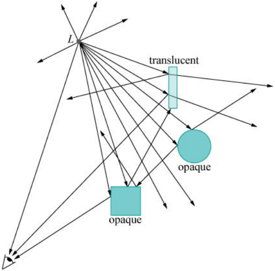
Ray Tracing and Radiosity

Alternative Rendering Methods

- Standard GPU pipeline (OpenGL): real-time, but shading based on local effects. No shadows in basic pipeline (must be added by ad-hoc methods).
- Ray tracing: Global shading model particularly good at specular effects (shiny surfaces). Too computationally expensive to be real-time.
- Radiosity: Global shading model particularly good at diffuse effects (matte surfaces, indirect light). Too computationally expensive to be real-time. But well suited for storing results as textures (as diffuse light is not viewpoint dependent).

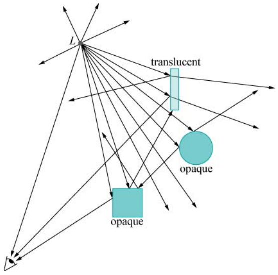
Ray Tracing

Follow photon paths to the eye.



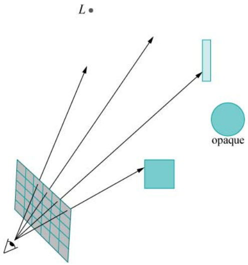
Ray Tracing

Follow photon paths to the eye.



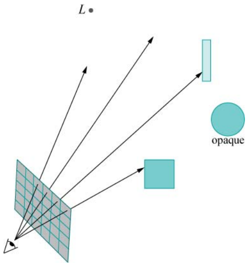
For efficiency, follow these in a backwards fashion (only spend time on photons actually hitting the eye).

Ray Tracing Level 0



At end of rays: calculate colors by Phongs lighting model.

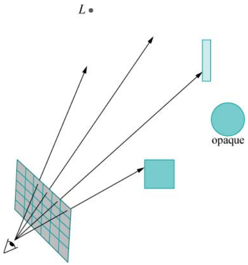
Ray Tracing Level 0



At end of rays: calculate colors by Phongs lighting model.

Same result as standard GPU pipeline.

Ray Tracing Level 0



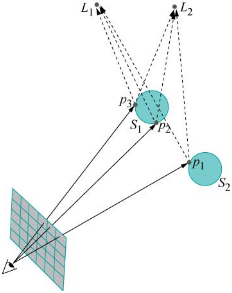
At end of rays: calculate colors by Phongs lighting model.

Same result as standard GPU pipeline.

Requires mechanism for fast determination of intersection points between rays and objects of the scene (e.g., store objects in spatial data structures—more on these in DM815).

Ray Tracing Level 1

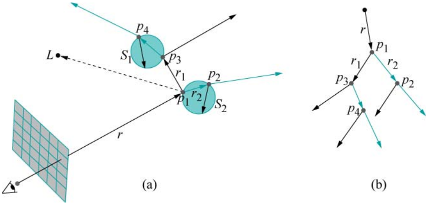
Add occlusion tests to light sources.



Gives shadows.

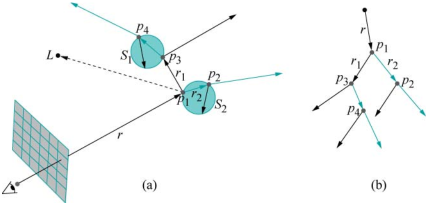
Ray Tracing Level 2+

Add reflection and transmission. Then recurse.



Ray Tracing Level 2+

Add reflection and transmission. Then recurse.



Note: simulating indirect light transfer between diffuse surfaces requires following many (approximating infinitely many) reflective rays from each ray intersection point in the recursive process.

Prohibitively costly. Similar thing with specular highlights on less glossy surfaces. So ray tracing works best for glossy materials.

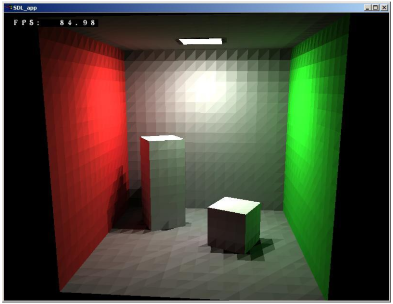
Radiosity

Model indirect light bouncing between purely diffuse (Lambertian) surfaces (of which some are light emitting).

(Figure by Jason Jacobs)

Patches

Start by patchifying the surfaces of the scene.



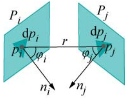
(Figure by Chuck Pheatt)

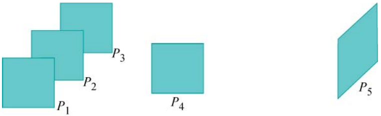
Entire path will be considered same ligth value (radiosity/brightness) B_i.

Radiosity: photons emitted per time and per area.

Form Factors

Form factor F_{ij}: measure of light transport between patch i and j.

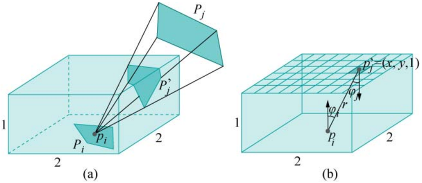




Calculate Form Factors

For F_{ij}: sum (integrate) contribution between (infinitesimal small areas around) all points on the two patches P_i and P_j.

Practical approximative calculation of form factors can be done via rendering in OpenGL:



Radiosity Equation

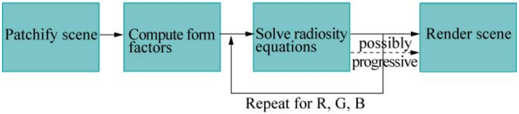
With M a specific $n \times n$ matrix (n is number of patches in schene) having entries depending on form factors and reflectance of patches, B the sought vector of brightness/radiosity values for patches and E the vector of emissive values for patches, one can prove:

$$MB = E$$

Using properties of the matrix M and results from matrix theory, it can be proven that the iterative process

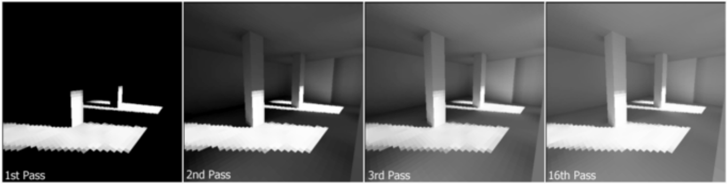
$$B_{i+1} = E + (I - M)B_i$$

for any start vector B_0 will converge to B. Usually faster than directly solving $MB = E$ (by e.g. inverting M), and less memory is used.



Iterative Process

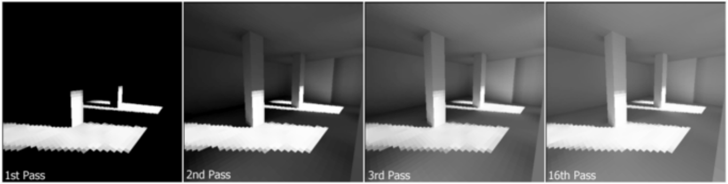
Here is the result of rendering a specific scene with B_1, B_2, B_3, B_{16}.



(Figure by Hugo Elias)

Iterative Process

Here is the result of rendering a specific scene with B_1, B_2, B_3, B_{16}.



(Figure by Hugo Elias)

The patching of the room may be refined based on one run of radiosity, increasing the resolution in areas with large variation in light values (edges of shadows, e.g.), and lowering the resolution in areas with small variation.