17 - Alternate Rendering Methods

Dr. Hamish Carr & Dr. Rafael Kuffner dos Anjos

Radiance Field

- $L = E + T(L)$
  - L is radiance field
  - E is emitted light
  - T is scattered light
    - $T = \int_{\omega \in \boldsymbol{S}^2_+(P)} L(R(P, \omega_i), -\omega_i) f_r(P, \omega_i, \omega_o) \omega_i \cdot \vec{n}(P) d\omega_i$

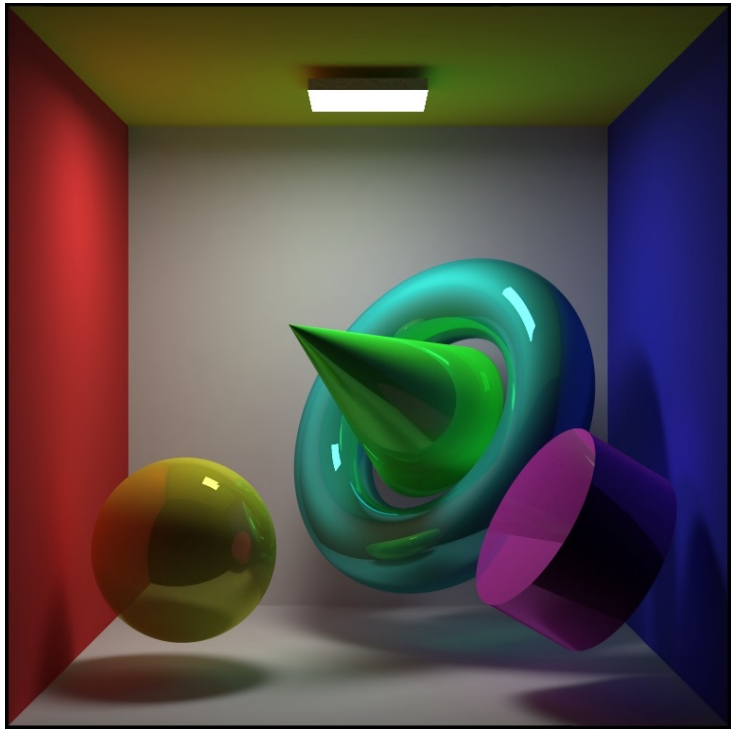
Mutual Visibility

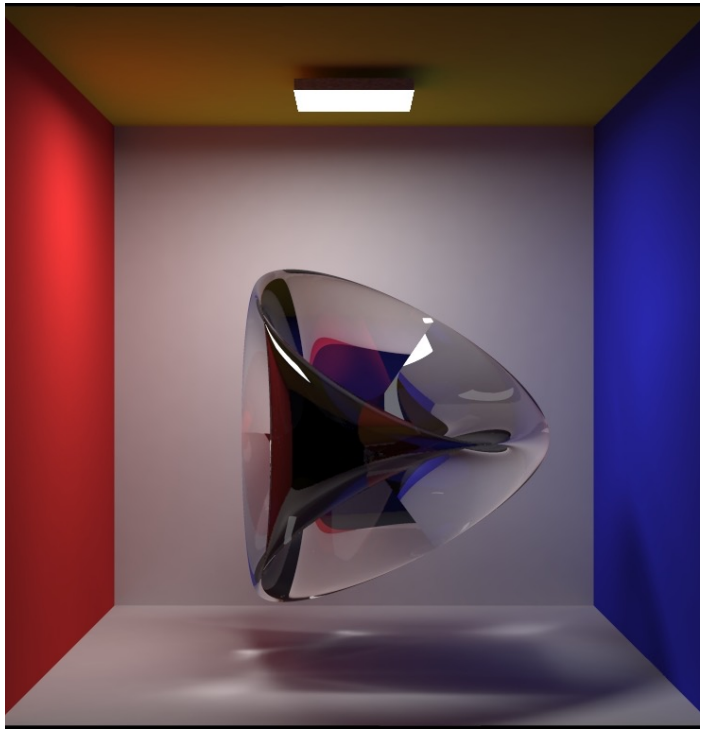
- Consider two points P & Q
  - Can they see *each other?*
  - If so, light travels from one to the other
  - The basis of many optimisations

Solution 1: Radiosity

- Divide the world into patches
- Compute light exchange between patches
- Matrix F expresses this interchange
  - $(I - D(\rho)F)b = e$
  - Turns it into a system of linear equations
  - Which we solve (expensively)

Radiosity Results

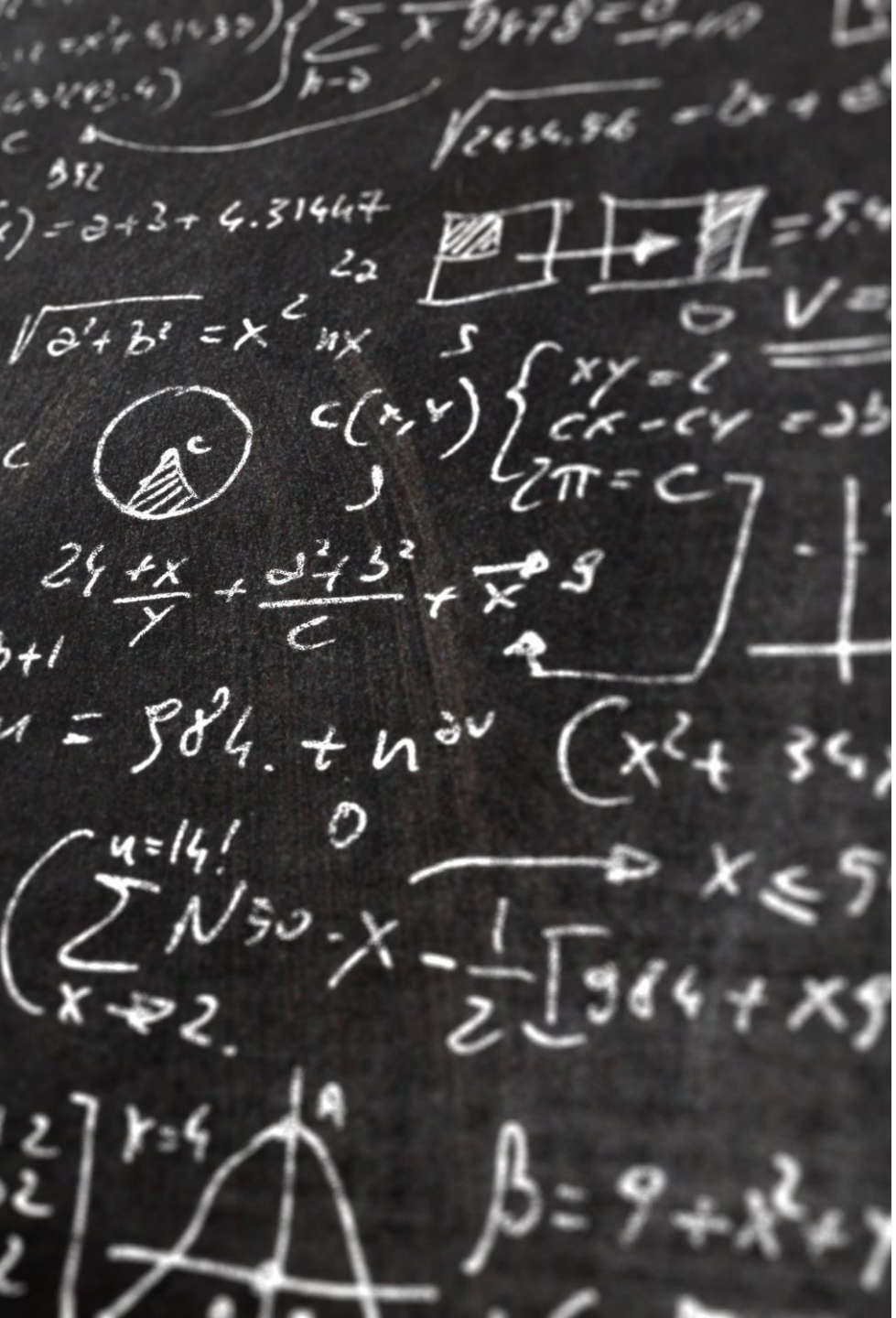




[ART - Radiosity Gallery (tuwien.ac.at)]

Radiosity Notes

- Matrix is sparse – many patches are occluded
- Patch shape / details are important
- We can progressively refine patches
- We can substitute spherical harmonics
- But it only works for Lambertian reflection
  - There are adaptations to add specular

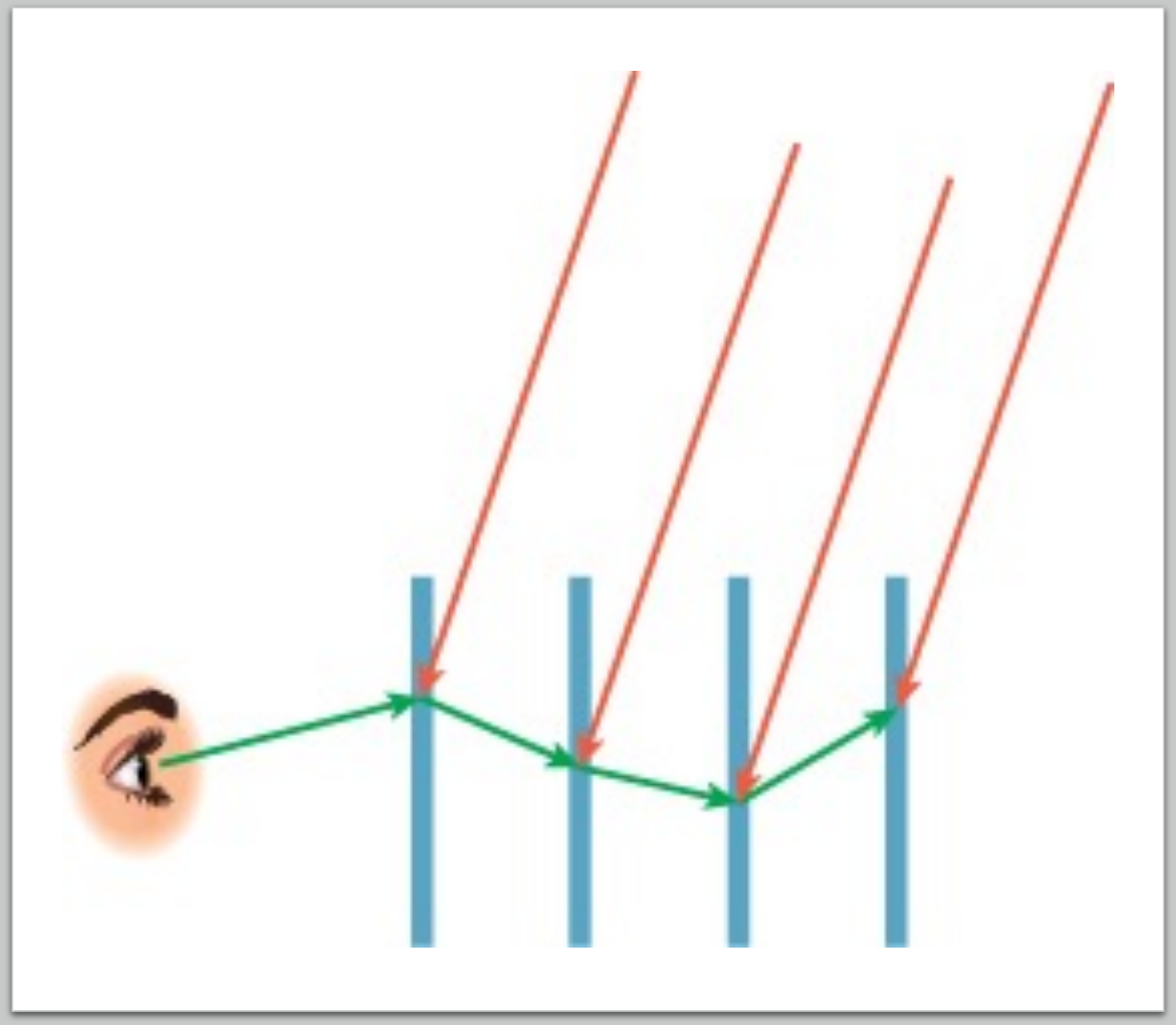


Series Solution

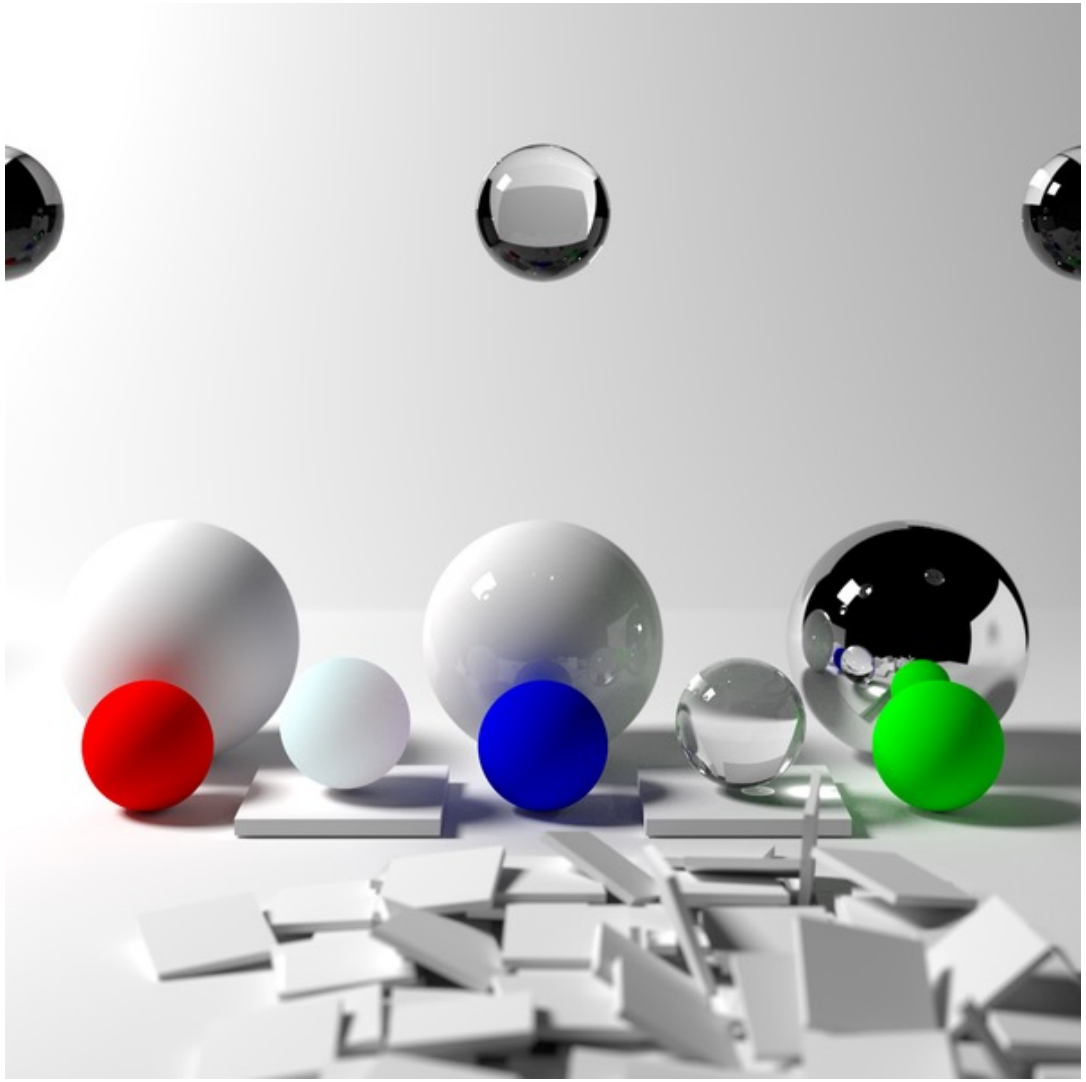
- Rewrite equation in terms of emitted light L^e:
  - $(I - T)L = L^e$
- And pretend they're matrices, so invert:
  - $L = (I - T)^{-1}L^e$
  - $L \approx (I + T + T^2 + T^3 + \cdots)L^e$
  - The sum of 0 + 1 + 2 + 3 + ... photon bounces
  - And we can exploit eigenanalysis

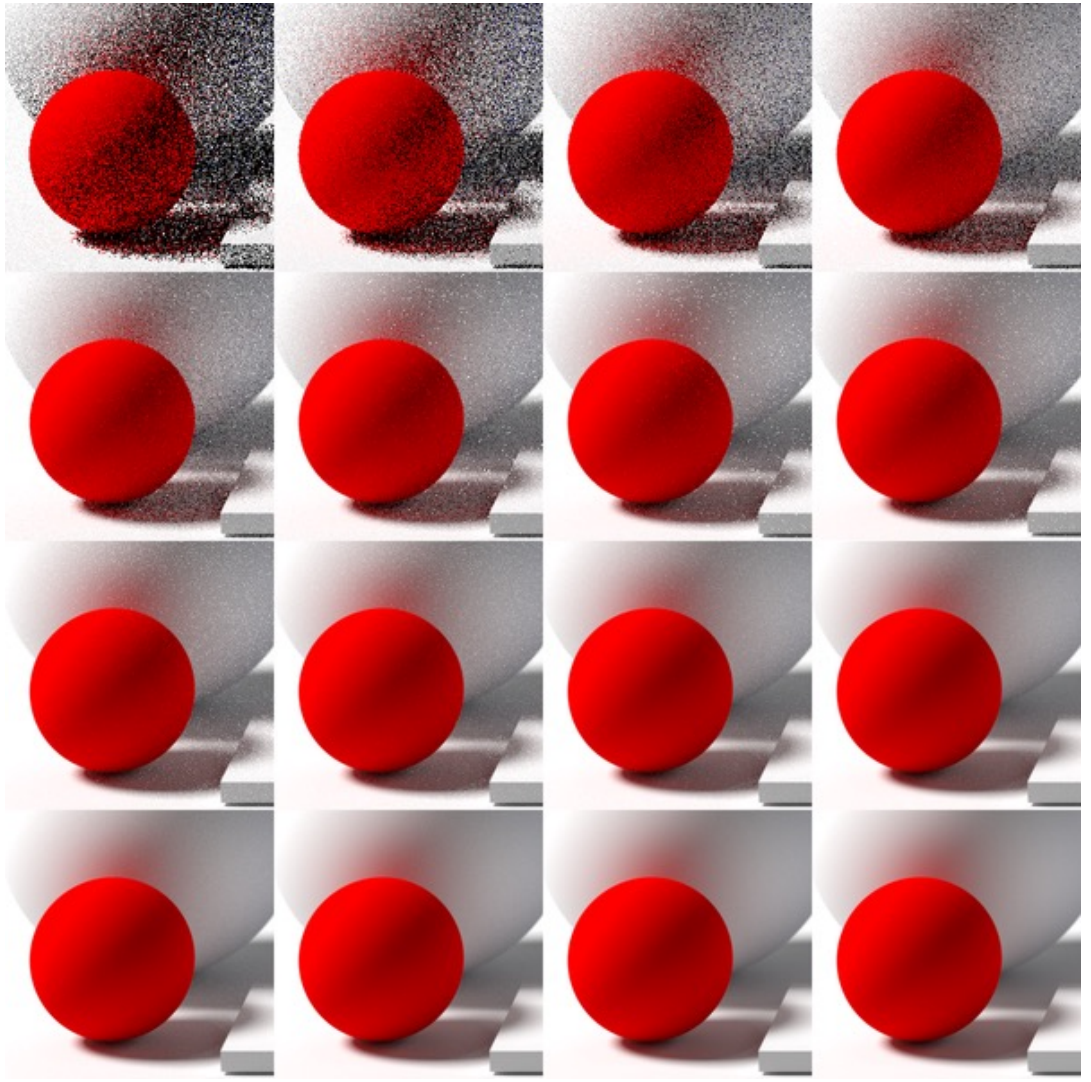
Path Tracing

- Only *one* recursive ray per path
- Subject to noise
- But you can repeat for many rays
- Or you can *share* the start of the path



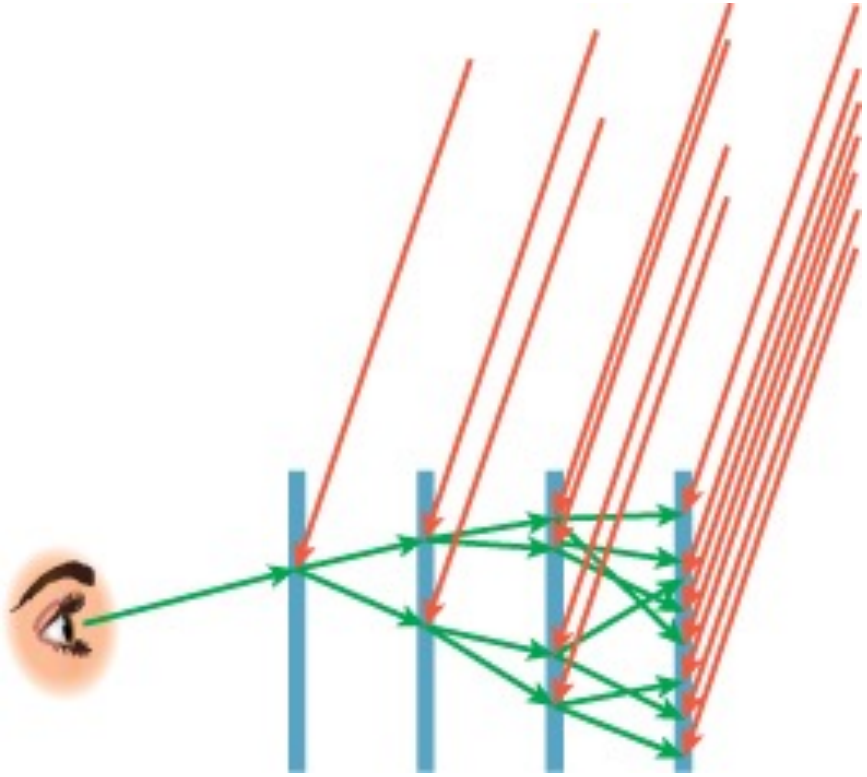
Path tracing results





Ray-tracing (Kajiya)

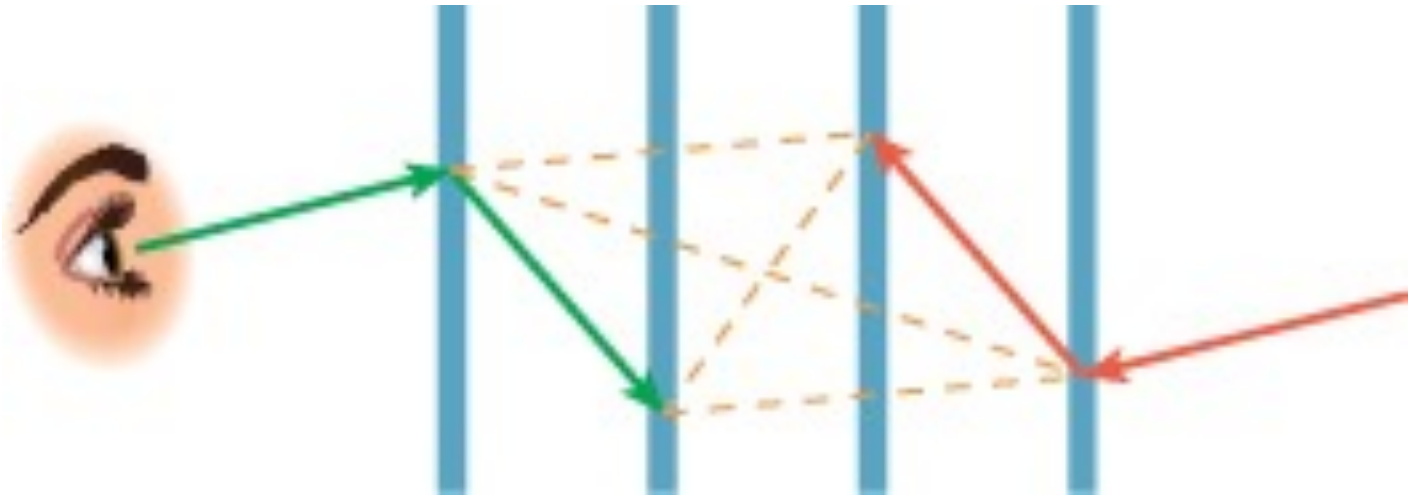
- Scatter multiple rays
- Share early stages of path
- Just a different way of doing it!



Bounce 1 Bounce 2 Bounce 3 Bounce 4

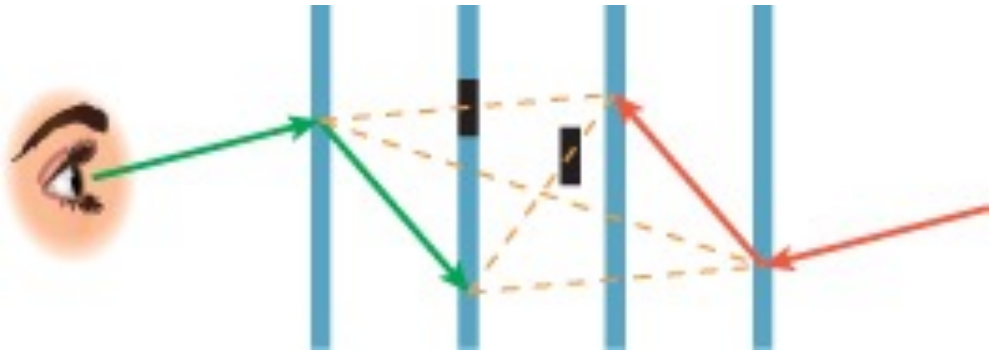
Bidirectional Path Tracing

- Trace light outwards from light source
- Trace paths outwards from eye
- Compute *all* splices between them
- May converge faster
- Uses Next event estimation



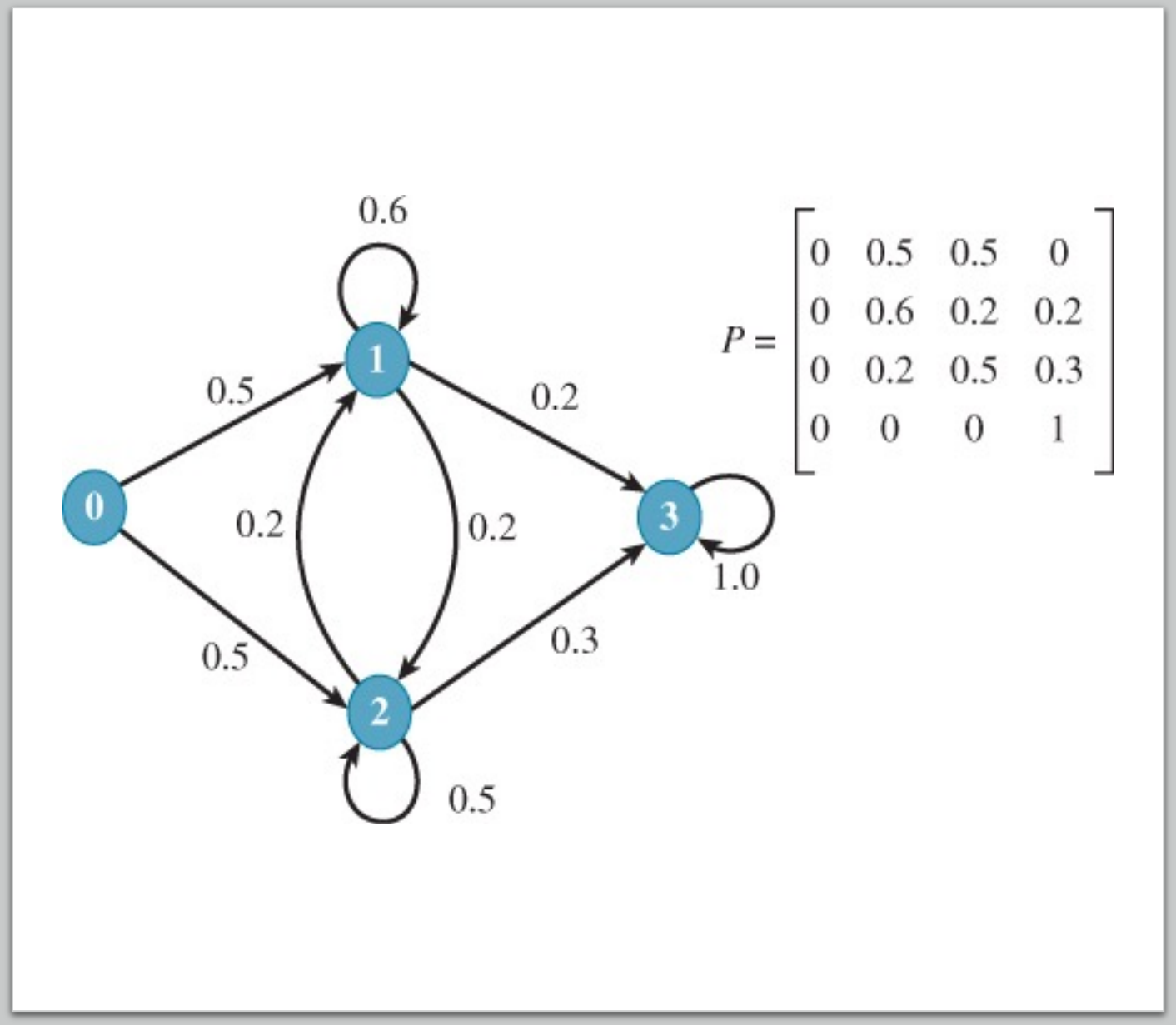
Occlusion

- Not all paths can be spliced
- Occluders (black) mean no contribution
- So this is wasteful for large scenes– $O(N^2)$



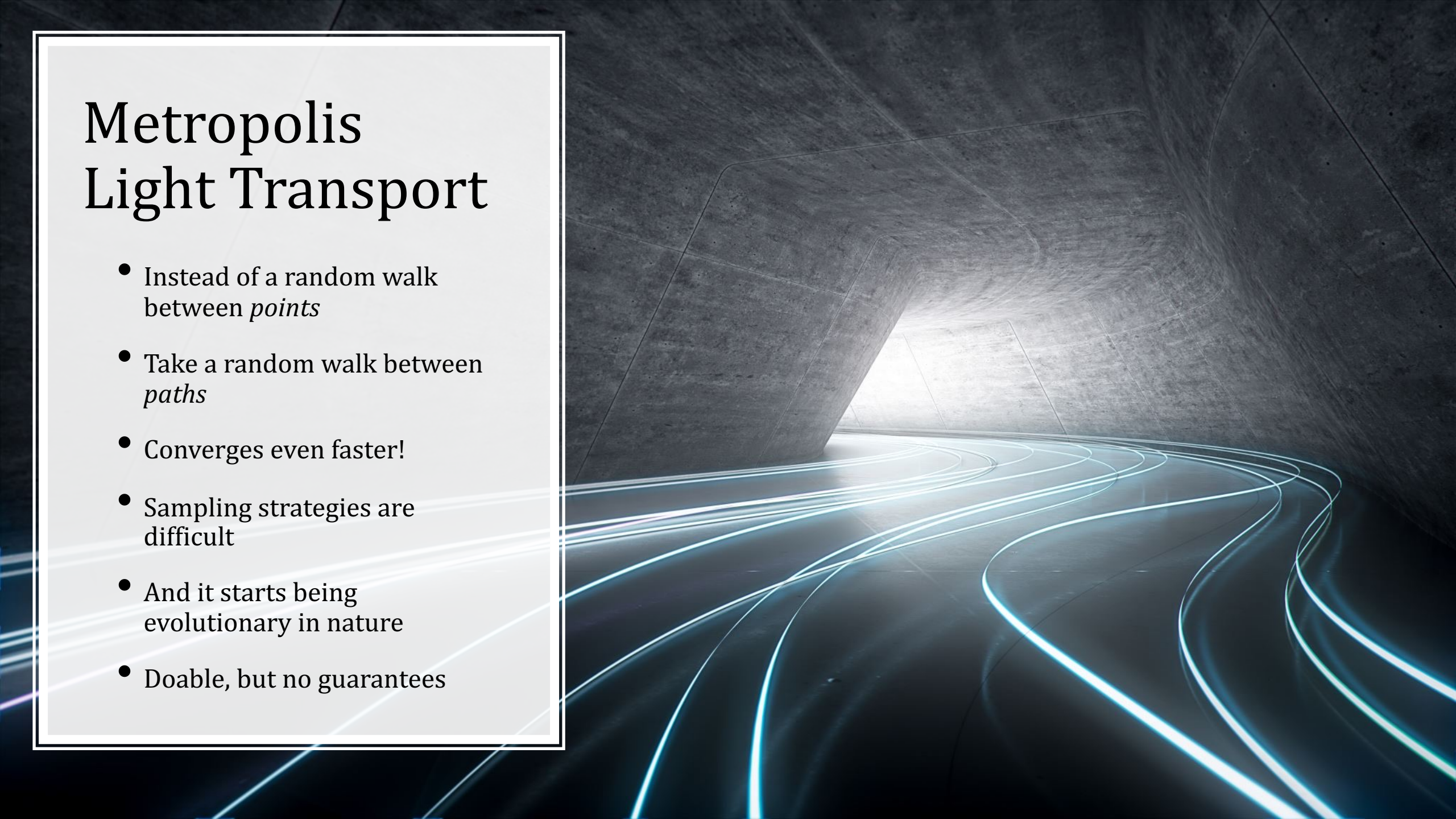
Markov Chains

- We can reframe this using Markov Chains
- Probability of light transport between points
- Each ray is a path through the FSM
- Expressed (again) as a matrix



Metropolis Light Transport

- Instead of a random walk between *points*
- Take a random walk between *paths*
- Converges even faster!
- Sampling strategies are difficult
- And it starts being evolutionary in nature
- Doable, but no guarantees



Photon Mapping

- Trace light from light sources to patches
- Accumulate light on patches
- Then trace rays from eye to patches
  - and collect accumulated photon map

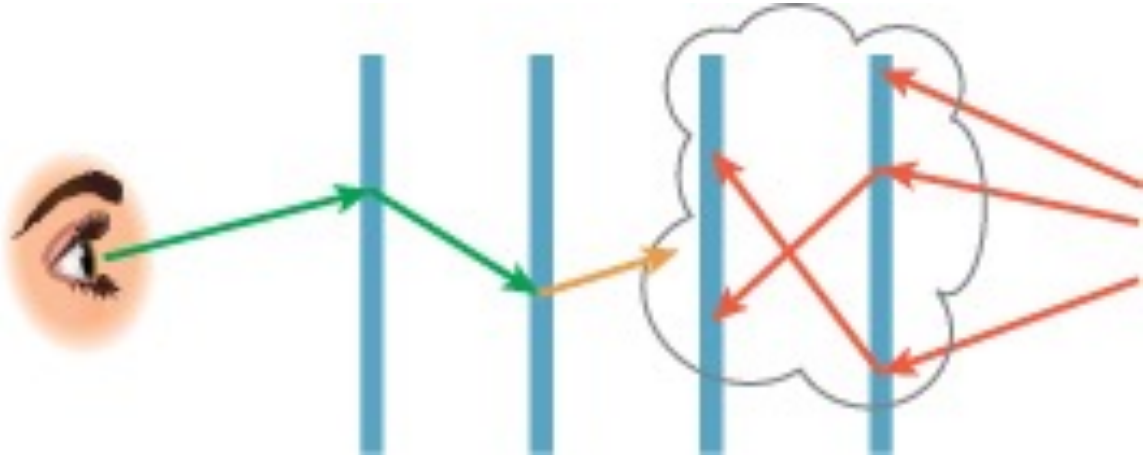


Image-Space Photon Maps

- 1st stage of photon mapping is photon tracing
- Trace photons from lights to surfaces
- 2nd stage is the actual render
- And we can substitute projective rendering
  - i.e. construct a photon map on CPU
  - render scene with photon map on GPU
- "Only" works for point-lights & pinhole

Phong Shading (in case you forgot)

- Replaces Gouraud shading of triangles
- Render the geometric object
- *Don't* interpolate shading on triangle
- Instead interpolate *normal*
- Then put Phong lighting in a shader
- But same pixel may compute many times
  - And Phong computation is expensive

Deffered Rendering

- Use projective rendering
- But with texture coordinates & IDs only
- Store the result in the frame buffer
- Now render the frame buffer as a quad
- Each pixel already knows its u,v & texture(s)
- So it can look up everything else

Summary

- There are *many* ways to do this
- All of them try to accelerate a basic idea
  - Trace rays from lights to surfaces
  - Trace rays from surfaces to surfaces
  - Trace rays from surfaces to camera
  - Integrate rays at a pixel or point

Images by

- S9: Qutorial