Computer Graphics

The Rendering Equation

Konstantin Tretyakov
kt@ut.ee

Nov 20, 2013

Outline

- Raycasting
- Raytracing
- Raymarching / Sphere tracing

- Rendering equation solvers
  - Radiosity, Path tracing, Photon mapping

Quiz

- Explain recursive raytracing in simple terms.

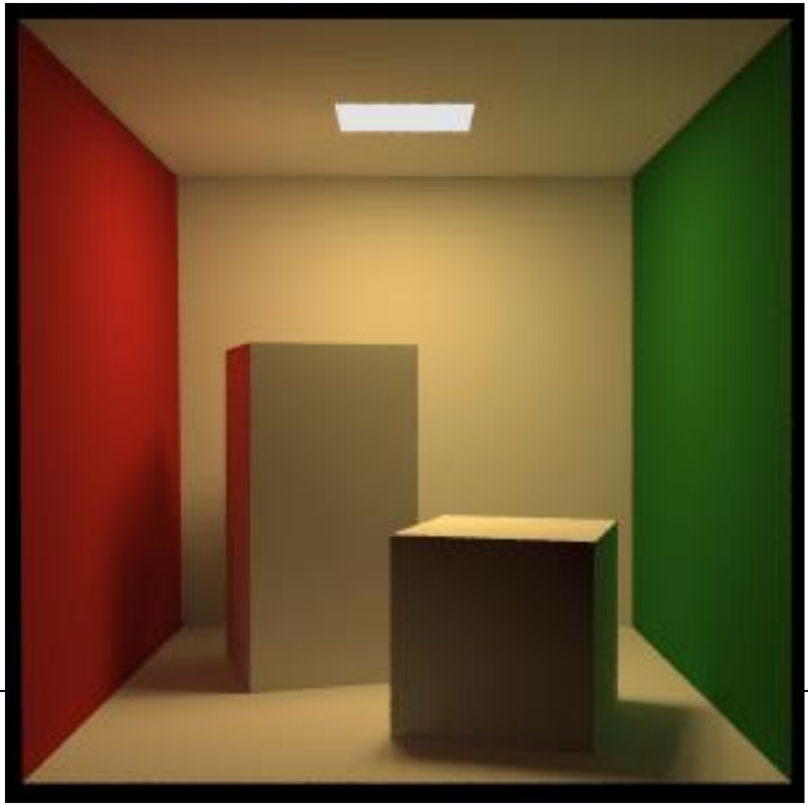
Main problem with raytracing

- Raytracing, although often producing nice results, is not a faithful model of real-world lighting.

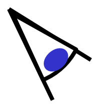
- Quiz: Why?

Main problem with raytracing

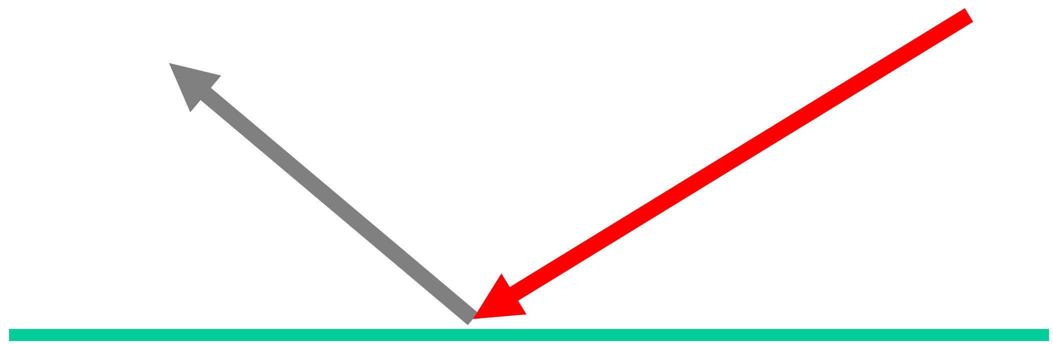
- In its pure form, it only considers “perfect” reflections and refractions, ignoring *diffuse light transfer* between surfaces.



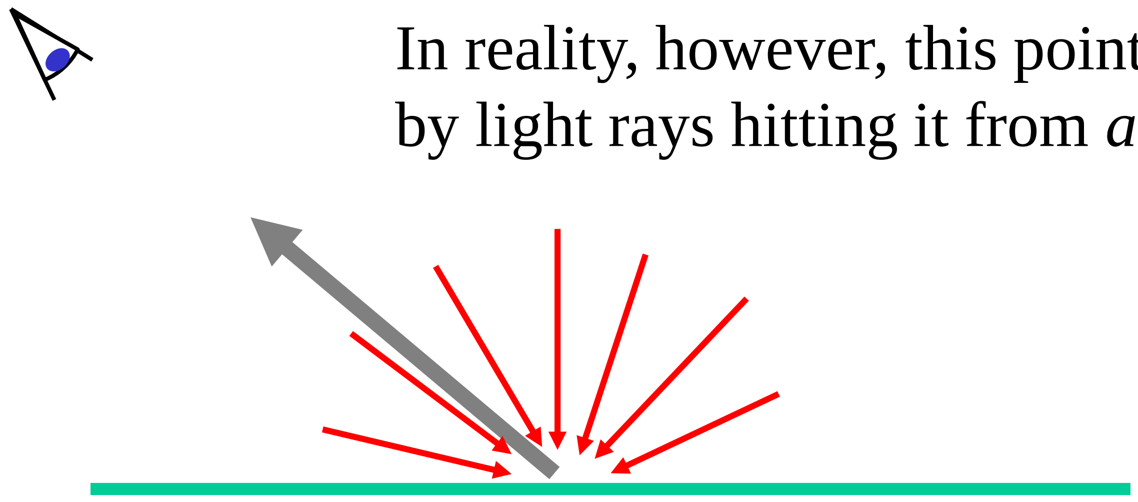
Raytracing vs Reality



Besides applying a local light model, standard raytracing only takes into account light rays affecting the point from **this** direction.

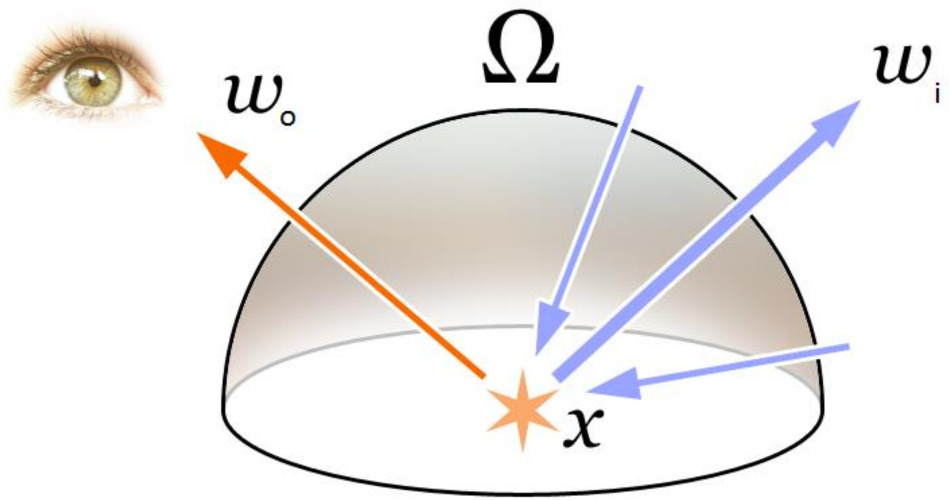


Raytracing vs Reality



In reality, however, this point is affected by light rays hitting it from *all* directions.

The Rendering Equation

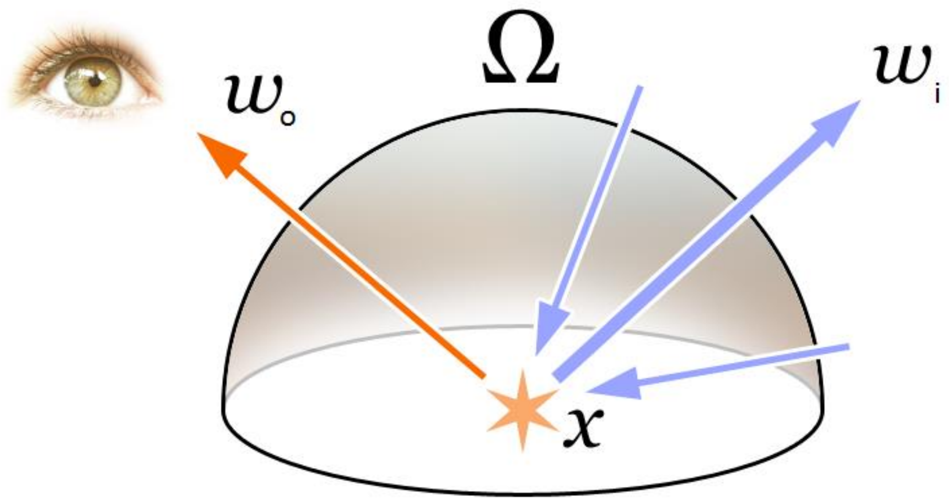


Consider light, leaving some point $\boldsymbol{x}$ in direction w_o:

$$L(\boldsymbol{w}_o, \boldsymbol{x}) =$$

http://en.wikipedia.org/wiki/File:Rendering_eq.png

The Rendering Equation

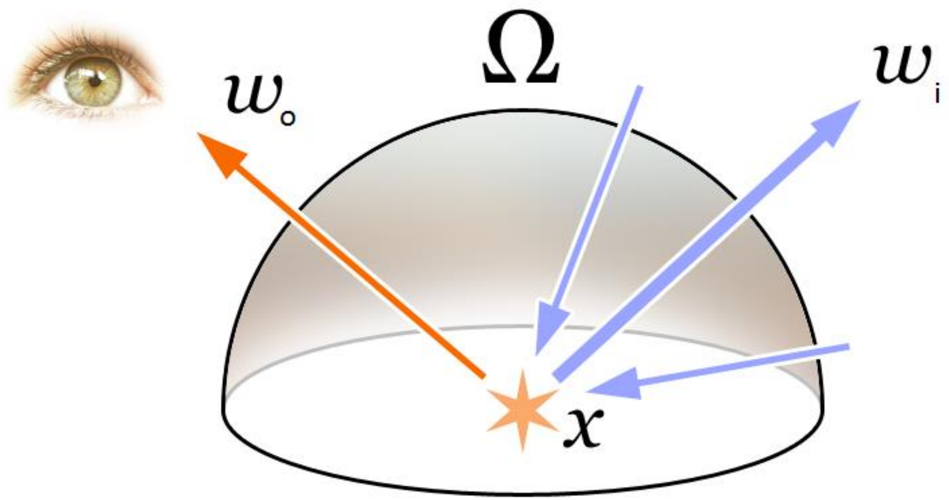


Consider light, leaving some point $\boldsymbol{x}$ in direction w_o:

$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) +$$

It consists of energy, **emitted** at that point in direction $\boldsymbol{w}_o$,…

The Rendering Equation



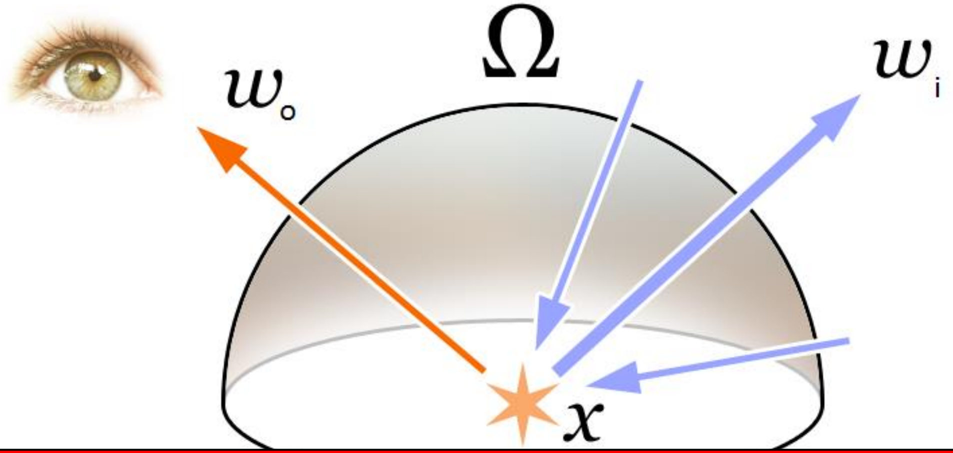
Consider light, leaving some point $\boldsymbol{x}$ in direction w_o:

…and energy, **reflected** towards $\boldsymbol{w}_o$ from direction $\boldsymbol{w}_{in}$

$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x})$$

It consists of energy, emitted at that point in direction $\boldsymbol{w}_o$,…

The Rendering Equation

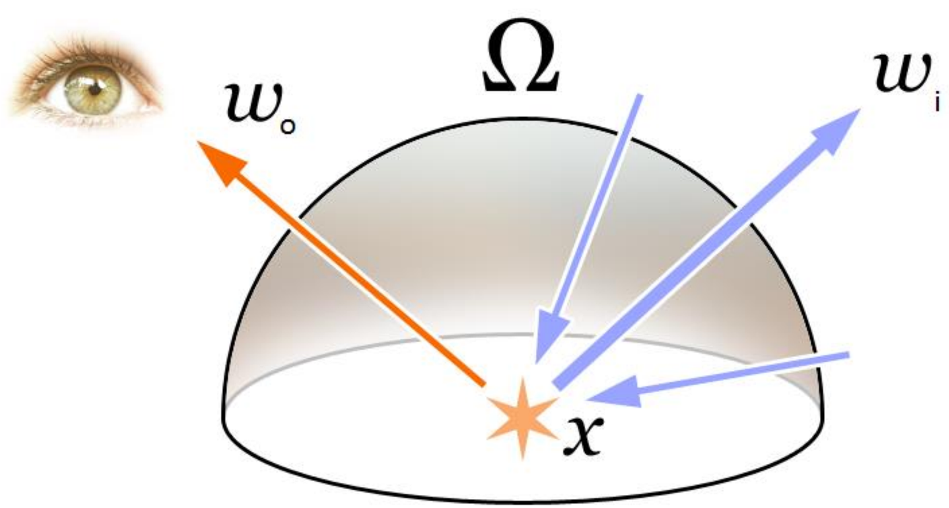


Bidirectional Reflectance Distribution Function (BRDF)
This function indicates how much energy incoming from $\boldsymbol{w}_{\text{in}}$ will be reflected towards $\boldsymbol{w}_o$. It is a property of a particular surface.

$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + \boxed{f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o)} L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x})$$

It consists of energy, emitted at that point in direction $\boldsymbol{w}_o$,…

The Rendering Equation



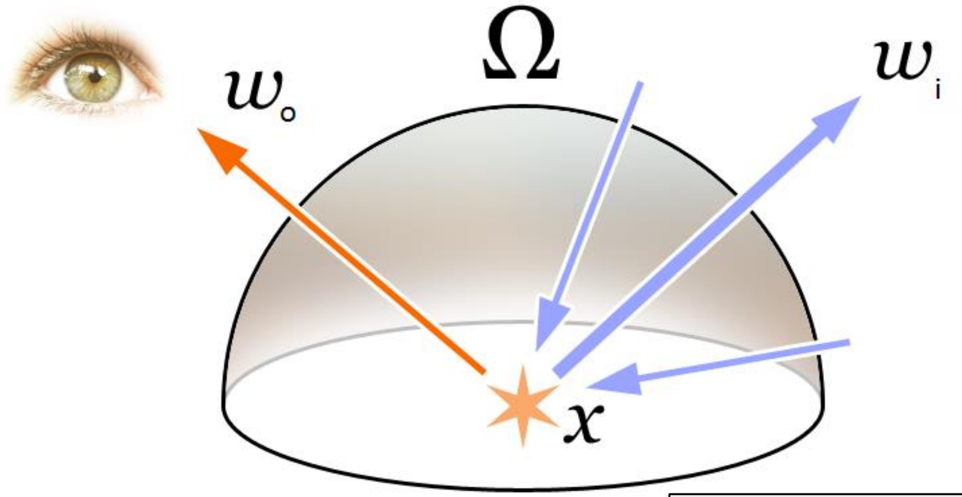
Consider light, leaving some point $\boldsymbol{x}$ in direction w_o:

…and energy, reflected towards $\boldsymbol{w}_o$ from direction $\boldsymbol{w}_{\text{in}}$

$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x})$$

It consists of energy, emitted at that point in direction $\boldsymbol{w}_o$,…

The Rendering Equation



Consider light, leaving some point $\boldsymbol{x}$ in direction w_o:

…averaged over all incoming directions

$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + \int_\Omega f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x}) (\boldsymbol{n}^T \boldsymbol{w}_{\text{in}}) d\boldsymbol{w}_{\text{in}}$$

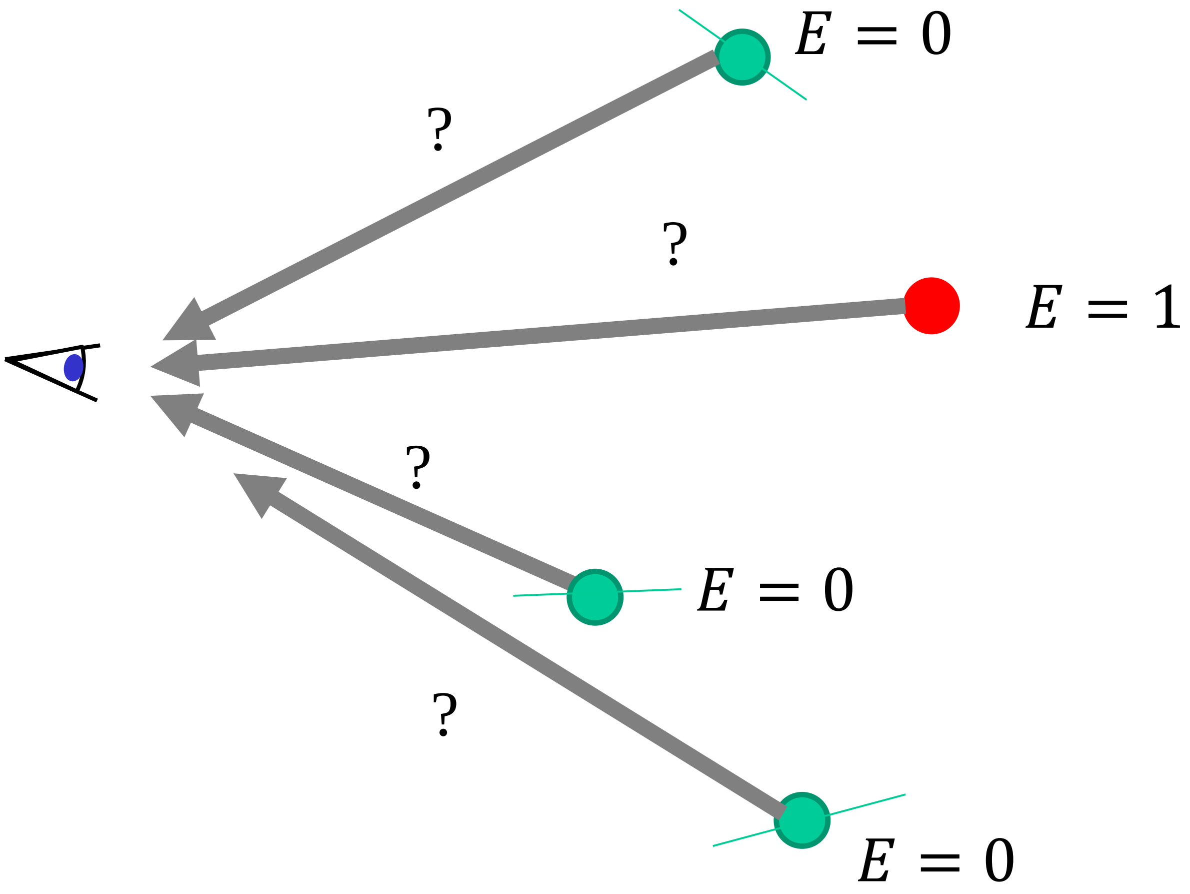
It consists of energy, emitted at that point in direction $\boldsymbol{w}_o$,…

The Rendering Equation

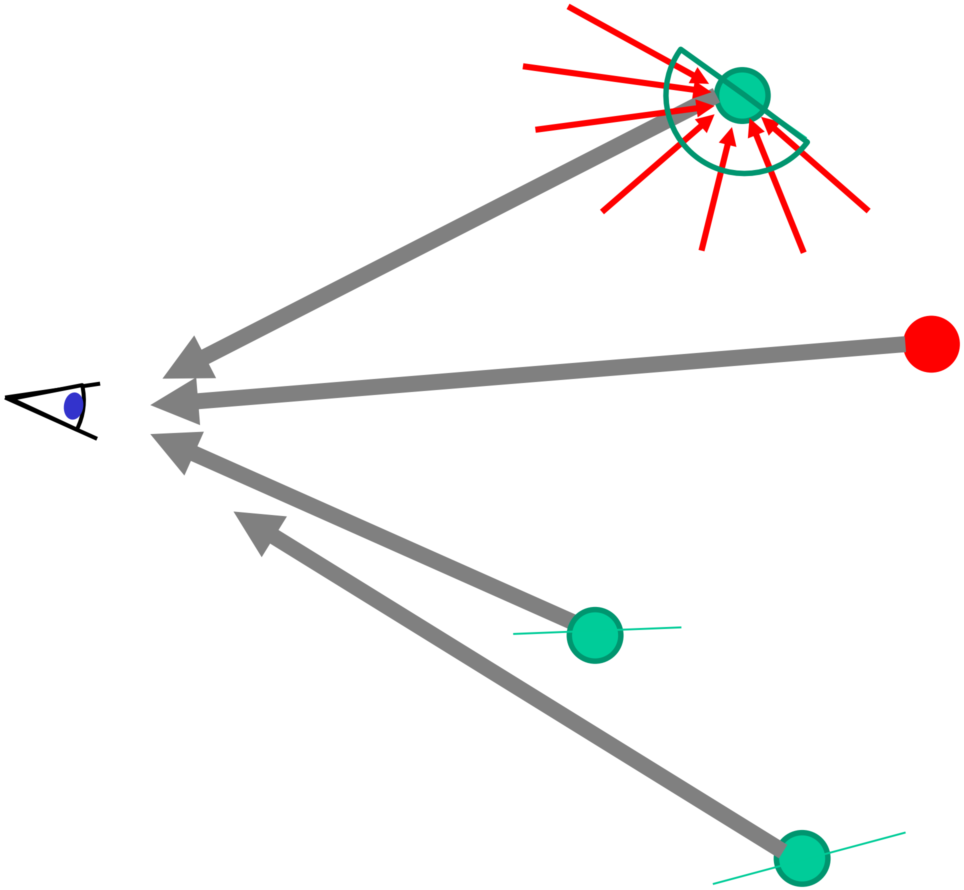
$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + \int_{\Omega} f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x})(\boldsymbol{n}^T \boldsymbol{w}_{\text{in}}) d\boldsymbol{w}_{\text{in}}$$

- The equation must hold true for all points $\boldsymbol{x}$ in the scene, for all color components (i.e. it is a system of equations, in fact).
- We initially know $E(\cdot)$ for all points (i.e. we know which points emit light)
- “Solving the rendering equation” means finding $L(\boldsymbol{w}_o, \boldsymbol{x})$ for points and light directions that reach the camera.

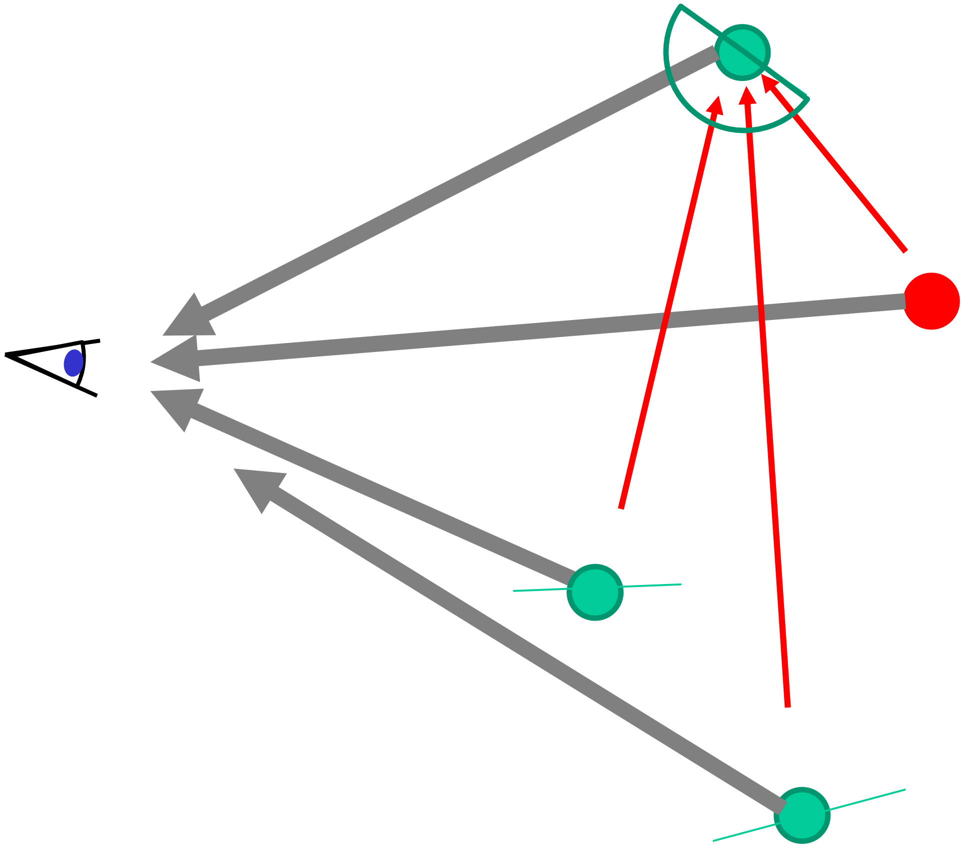
Rendering equation in 2D



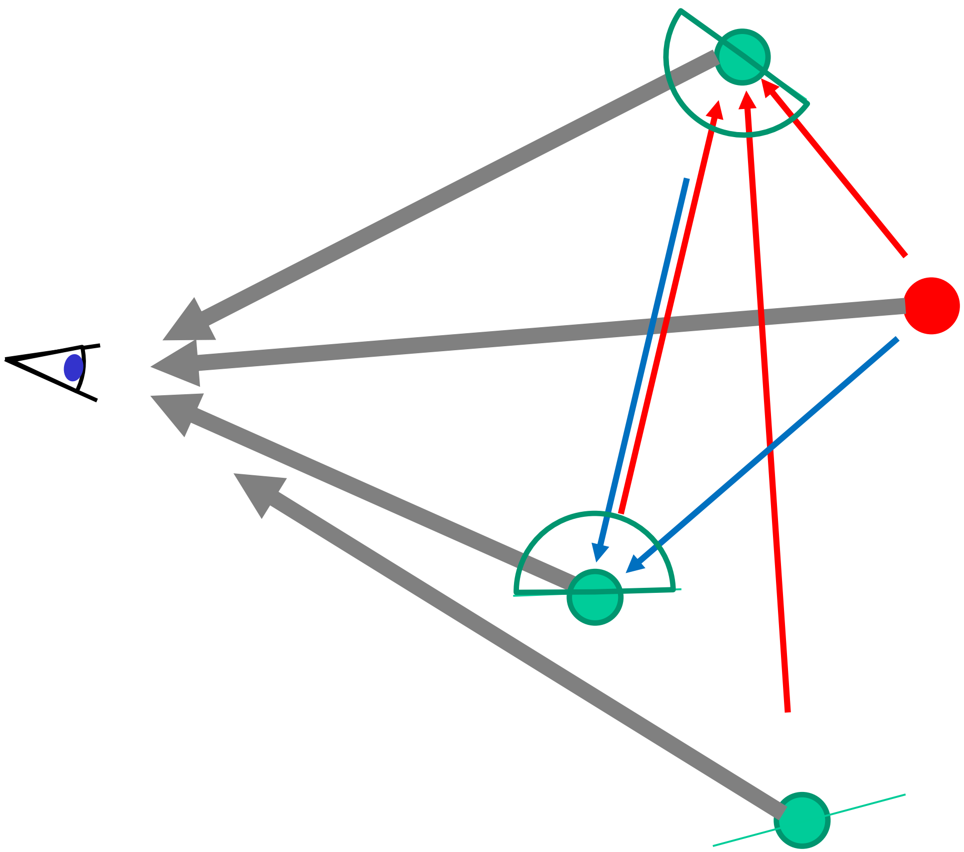
Rendering equation in 2D



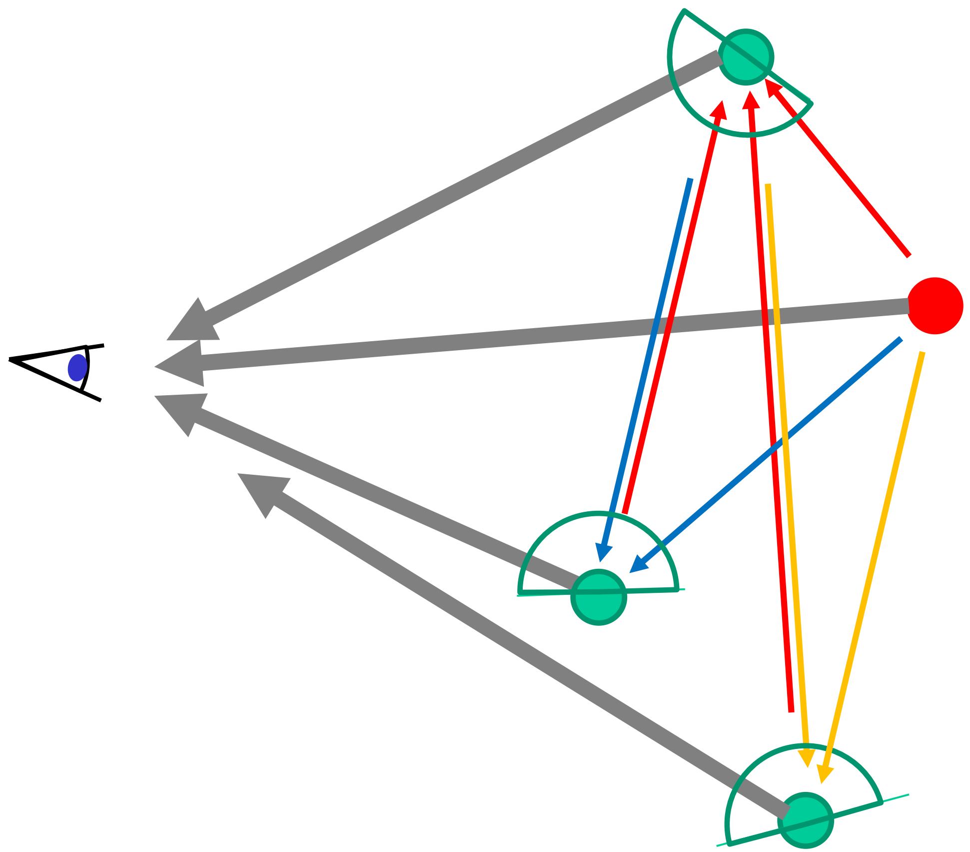
Rendering equation in 2D



Rendering equation in 2D



Rendering equation in 2D



Solving the Rendering Equation

Two primary approaches

- Replace the integral with a *finite sum*:
  - **Radiosity**
- Replace the integral with a *random sample*:
  - **Path tracing**
  - **Photon mapping**

Radiosity

$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + \int_\Omega f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x})(\boldsymbol{n}^T \boldsymbol{w}_{\text{in}}) d\boldsymbol{w}_{\text{in}}$$

First, assume the scene consists of
a **finite** number of patches (e.g triangles)

Radiosity

$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + \int_\Omega f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x})(\boldsymbol{n}^T \boldsymbol{w}_{\text{in}}) d\boldsymbol{w}_{\text{in}}$$

$$L(\boldsymbol{w}_o, \boldsymbol{x}_k) = E(\boldsymbol{w}_o, \boldsymbol{x}_k) + \sum_j f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{x}_j, \boldsymbol{x})(\boldsymbol{n}^T \boldsymbol{w}_{\text{in}})$$

First, assume the scene consists of
a **finite** number of patches (e.g triangles)

Radiosity

$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + \int_\Omega f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x})(\boldsymbol{n}^T \boldsymbol{w}_{\text{in}}) d\boldsymbol{w}_{\text{in}}$$

$$L(\boldsymbol{w}_o, \boldsymbol{x}_k) = E(\boldsymbol{w}_o, \boldsymbol{x}_k) + \sum_j f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{x}_j, \boldsymbol{x})(\boldsymbol{n}^T \boldsymbol{w}_{\text{in}})$$

Second, assume each patch radiates equally in all directions, i.e. all lighting is *diffuse*.

Radiosity

$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + \int_\Omega f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x})(\boldsymbol{n}^T \boldsymbol{w}_{\text{in}}) d\boldsymbol{w}_{\text{in}}$$

$$L(\boldsymbol{w}_o, \boldsymbol{x}_k) = E(\boldsymbol{w}_o, \boldsymbol{x}_k) + \sum_j f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{x}_j, \boldsymbol{x})(\boldsymbol{n}^T \boldsymbol{w}_{\text{in}})$$

$$L(\boldsymbol{x}_k) = E(\boldsymbol{x}_k) + \sum_j \rho_k F_{kj} L(\boldsymbol{x}_j)$$

Second, assume each patch radiates equally in all directions, i.e. all lighting is *diffuse*.

Radiosity

$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + \int_\Omega f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x})(\boldsymbol{n}^T \boldsymbol{w}_{\text{in}}) d\boldsymbol{w}_{\text{in}}$$

$$L(\boldsymbol{w}_o, \boldsymbol{x}_k) = E(\boldsymbol{w}_o, \boldsymbol{x}_k) + \sum_j f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{x}_j, \boldsymbol{x})(\boldsymbol{n}^T \boldsymbol{w}_{\text{in}})$$

$$L(\boldsymbol{x}_k) = E(\boldsymbol{x}_k) + \sum_j \rho_k F_{kj} L(\boldsymbol{x}_j)$$

$$L_k = E_k + \rho_k \sum_j F_{kj} L_j$$

Second, assume each patch radiates equally in all directions, i.e. all lighting is *diffuse*.

Radiosity

Energy created in patch k. (i.e. light source intensity)

Energy coming from another patch j.

"View factor": How much of j's energy is received by k.

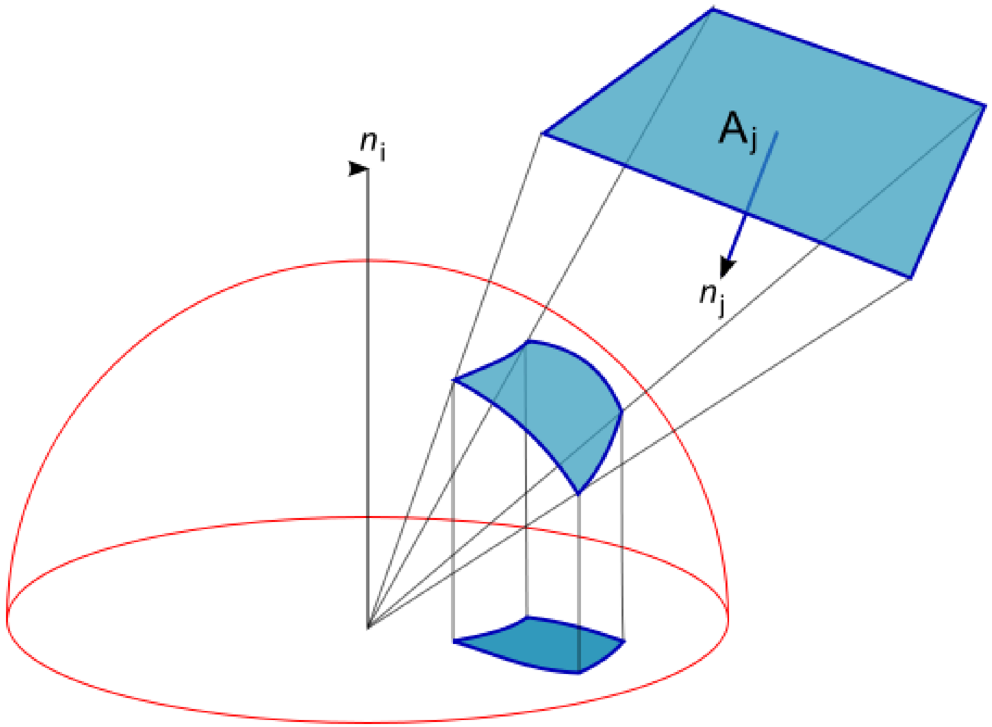
E.g. $F_{kj} = 0$ if k is occluded from j.

Energy leaving patch k.

$$\boxed{L_k} = \boxed{E_k} + \boxed{\rho_k} \sum_j \boxed{F_{kj}} \boxed{L_j}$$

How much incoming energy k re-diffuses. i.e. "patch color"

View factors



The amount of energy that patch *i* receives from patch *j* is proportional to the projected solid angle of that patch as seen from *i*

http://en.wikipedia.org/wiki/File:Nusselt_analog.svg

Radiosity

$$L_k = E_k + \rho_k \sum_j F_{kj} L_j$$

Radiosity

$$\boldsymbol{l} = \boldsymbol{e} + \boldsymbol{\rho} \cdot \boldsymbol{F}\boldsymbol{l}$$

Radiosity

$$\boldsymbol{l} = \boldsymbol{e} + \boldsymbol{\rho} \cdot \boldsymbol{F}\boldsymbol{l}$$

Radiosity

$$\boldsymbol{l} = \boldsymbol{e} + \boldsymbol{\rho} \cdot \boldsymbol{F}\boldsymbol{l}$$

This is a linear equation. Analytically the solution is:

$$\boldsymbol{l} \coloneqq (\boldsymbol{I} - \boldsymbol{\rho}\boldsymbol{F})^{-1}\boldsymbol{e}$$

The number of patches is usually prohibitively large to use this approach.

Radiosity

$$\boldsymbol{l = e + \rho \cdot Fl}$$

Instead we can use the Jacobi iteration method:

$$\boldsymbol{l_0 = e}$$
$$\boldsymbol{l_1 = e + \rho \cdot Fl_0}$$
$$\boldsymbol{l_2 = e + \rho \cdot Fl_1}$$
$$\boldsymbol{l_3 = e + \rho \cdot Fl_2}$$

…

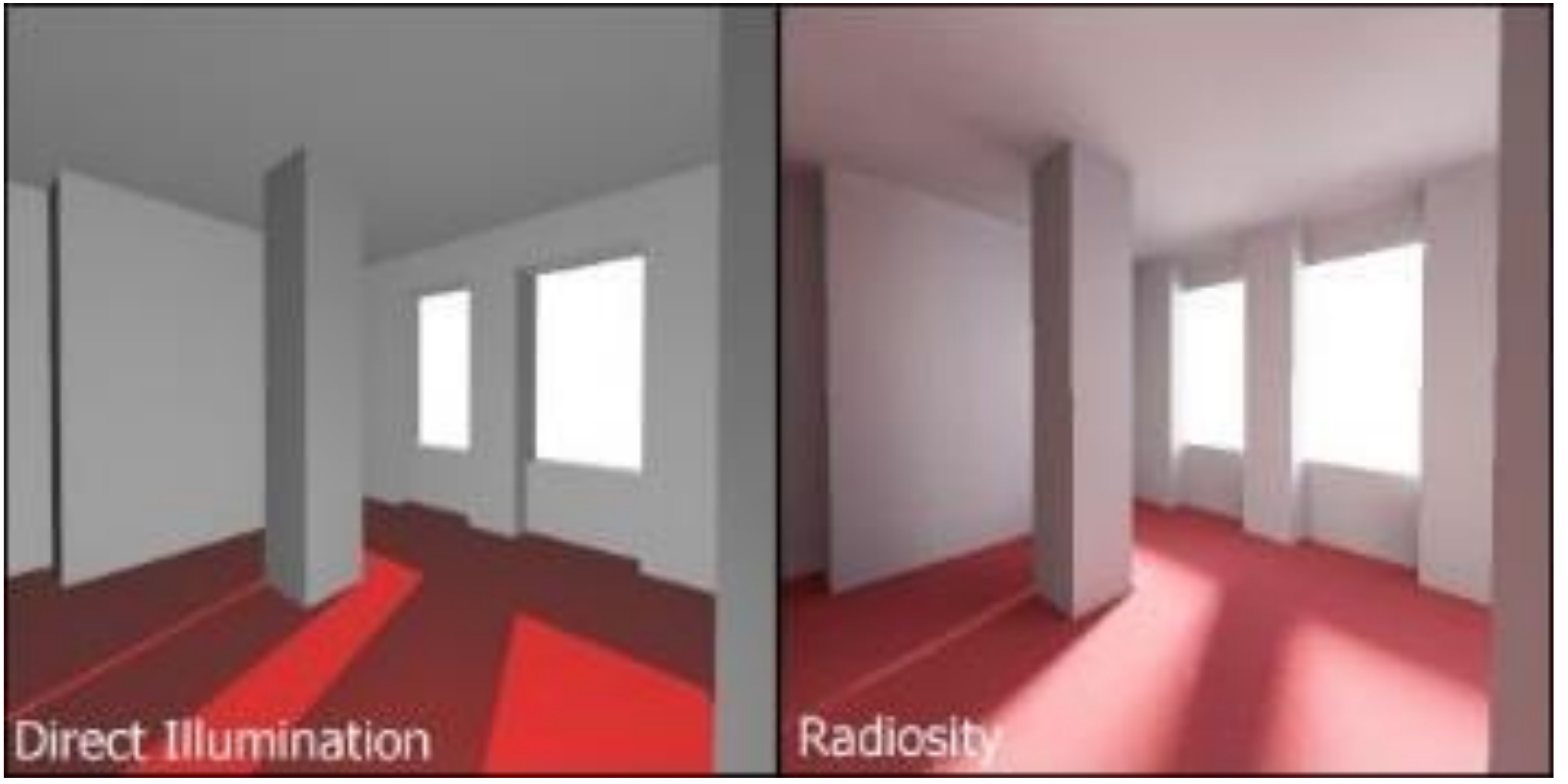
The algorithm converges in a few iterations.

Radiosity

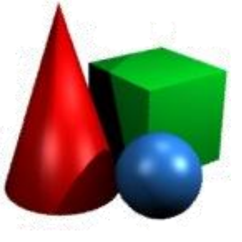
In simple terms:

- In the beginning all patches except light-emitting ones have zero *radiosity* value.
- For each patch, recompute its radiosity value as a weighted sum of received radiosity + emitted radiosity.
- Repeat several times.

Radiosity



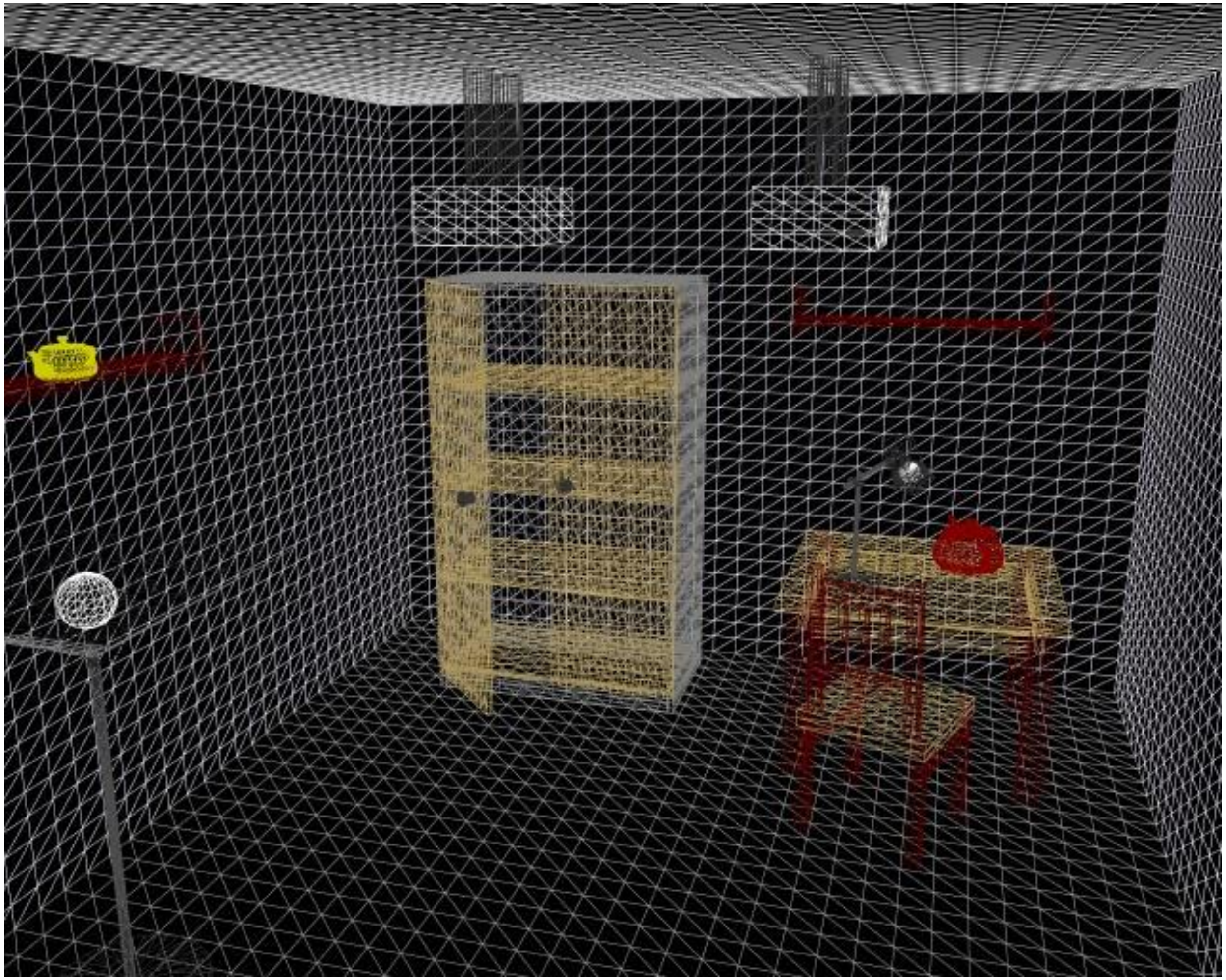
http://en.wikipedia.org/wiki/File:Radiosity_Comparison.jpg



Radiosity

http://dudka.cz/rrv/gallery

Radiosity



http://dudka.cz/rrv/gallery

Radiosity

http://dudka.cz/rrv/gallery

Radiosity

http://dudka.cz/rrv/gallery

Radiosity

http://dudka.cz/rrv/gallery

Radiosity

Radiosity

Quiz

- Assuming the scene has 10000 patches:
  - How much memory is used by the iterative Radiosity algorithm?
  - How many operations are performed per iteration (i.e. what is the computational complexity)?

Quiz

- Assuming the scene has 10000 patches:
  - How much memory is used by the iterative Radiosity algorithm? ~10000 (i.e. $O(n)$)
  - How many operations are performed per iteration (i.e. what is the computational complexity)?

    ~100 000 000

    (i.e. the complexity is $O(n^2)$)

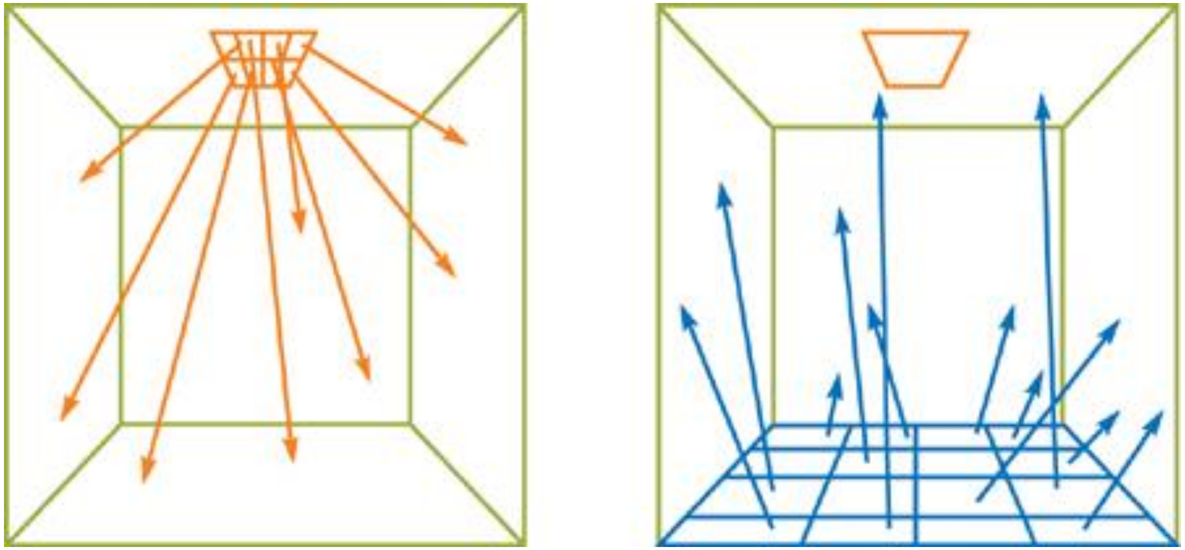
Progressive refinement radiosity

- Quadratic complexity is prohibitive for many scenes.

- A good approximation to radiosity which works in linear time per iteration (although requires more iterations) is *progressive refinement*.

Progressive refinement radiosity

- Deposit “energy chunks” on light-emitting patches only
- Pick the most energetic patch as the “shooter”, and distribute its energy to all other patches (“receivers”), according to view factors.
- Repeat



http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter39.html

Quiz

- Suppose we run the radiosity algorithm for a scene.
- Then viewer position changes.
- Do we need to re-run the algorithm?

Radiosity

- Radiosity values are fixed and viewer-independent, so they can be pre-computed for the scene and stored as light maps.

- Consequently, radiosity is usually used together with other techniques (e.g. raytracing or standard pipeline).

Radiosity

- Progressive-refinement radiosity can be GPU-optimized

- The technique is pretty much always used in professional architecture / interior design CAD renderings.

Solving the Rendering Equation

Two primary approaches

- Replace the integral with a *finite sum*:
  - **Radiosity**
- Replace integral with a *random sample*:
  - **Path tracing**
  - **Photon mapping**

Monte-carlo integration

$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + \int_\Omega f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x})(\boldsymbol{n}^T \boldsymbol{w}_{\text{in}}) d\boldsymbol{w}_{\text{in}}$$

Rather than taking the integral, sum over a *random sample* of directions

Monte-carlo integration

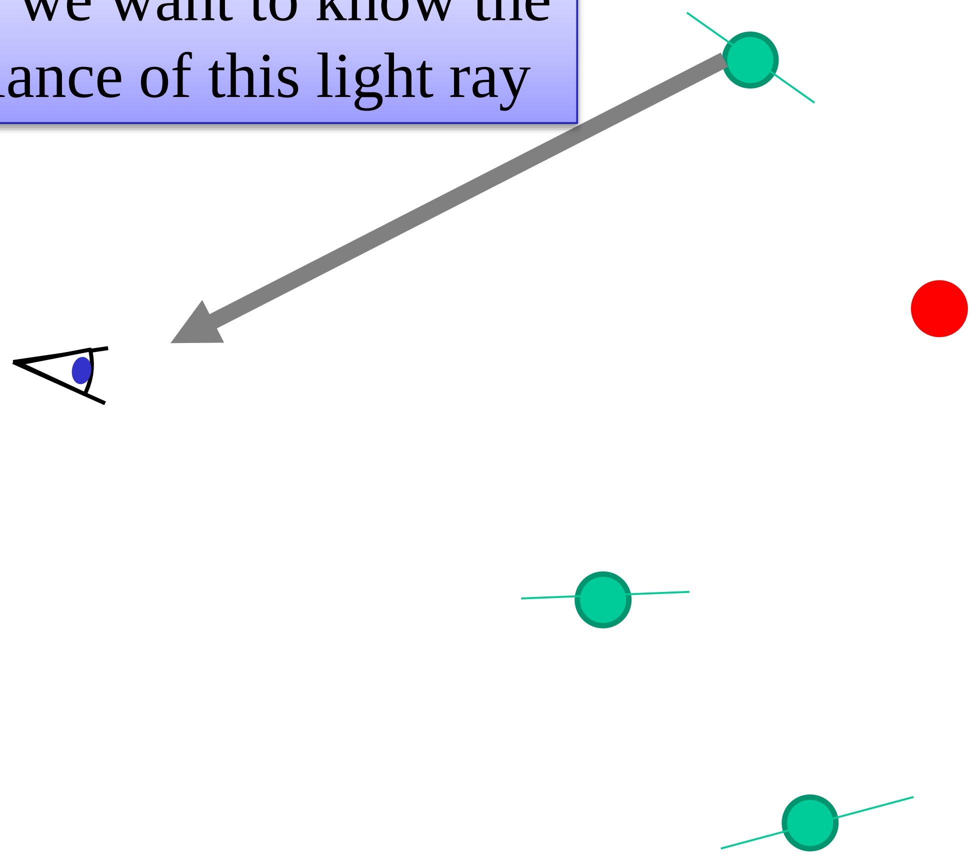
$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + \int_{\Omega} f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x}) (\boldsymbol{n}^T \boldsymbol{w}_{\text{in}}) d\boldsymbol{w}_{\text{in}}$$

$$L(\boldsymbol{w}_o, \boldsymbol{x}) = E(\boldsymbol{w}_o, \boldsymbol{x}) + \sum_{\boldsymbol{w}_{\mathbf{in}} \in \text{Random}} f(\boldsymbol{w}_{\text{in}}, \boldsymbol{w}_o) L_{\text{in}}(\boldsymbol{w}_{\text{in}}, \boldsymbol{x}) (\boldsymbol{n}^T \boldsymbol{w}_{\text{in}})$$

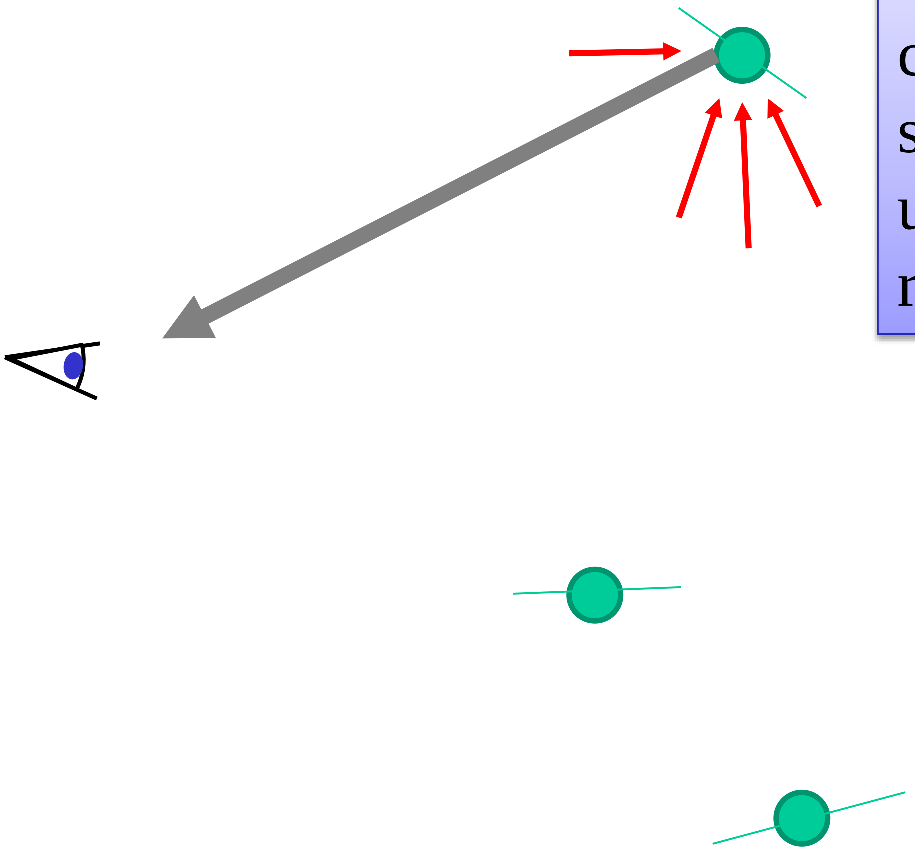
Rather than taking the integral, sum over a *random sample* of directions

Monte-carlo integration

Say we want to know the radiance of this light ray

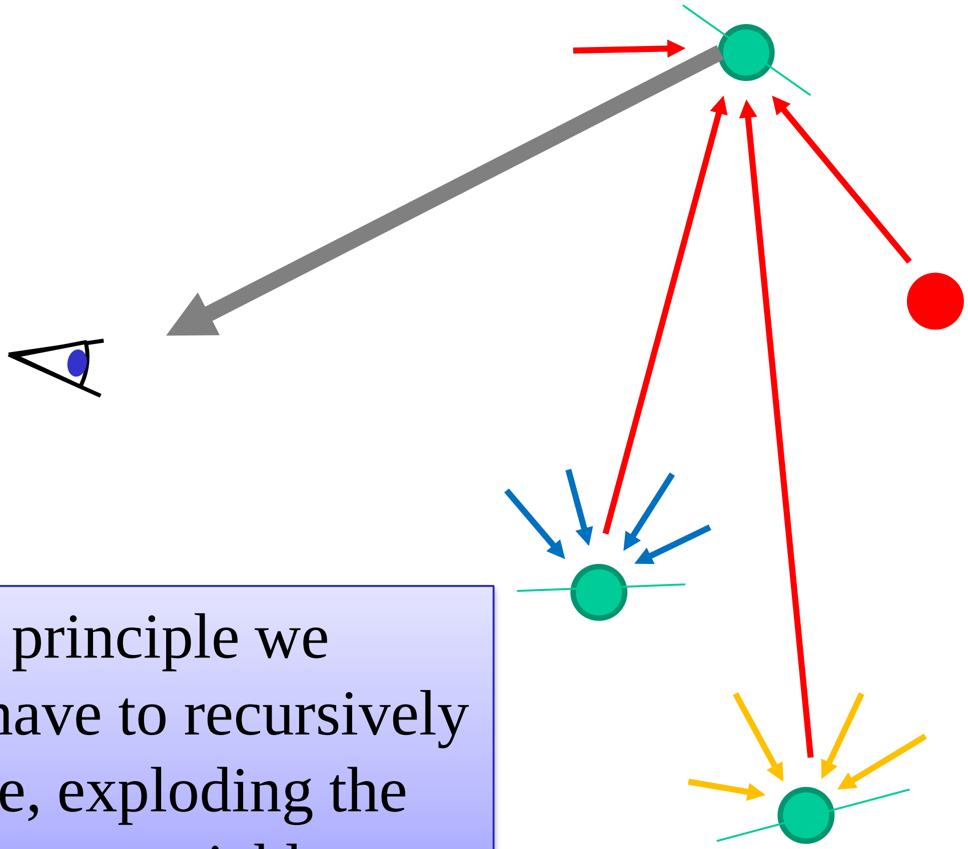


Monte-carlo integration



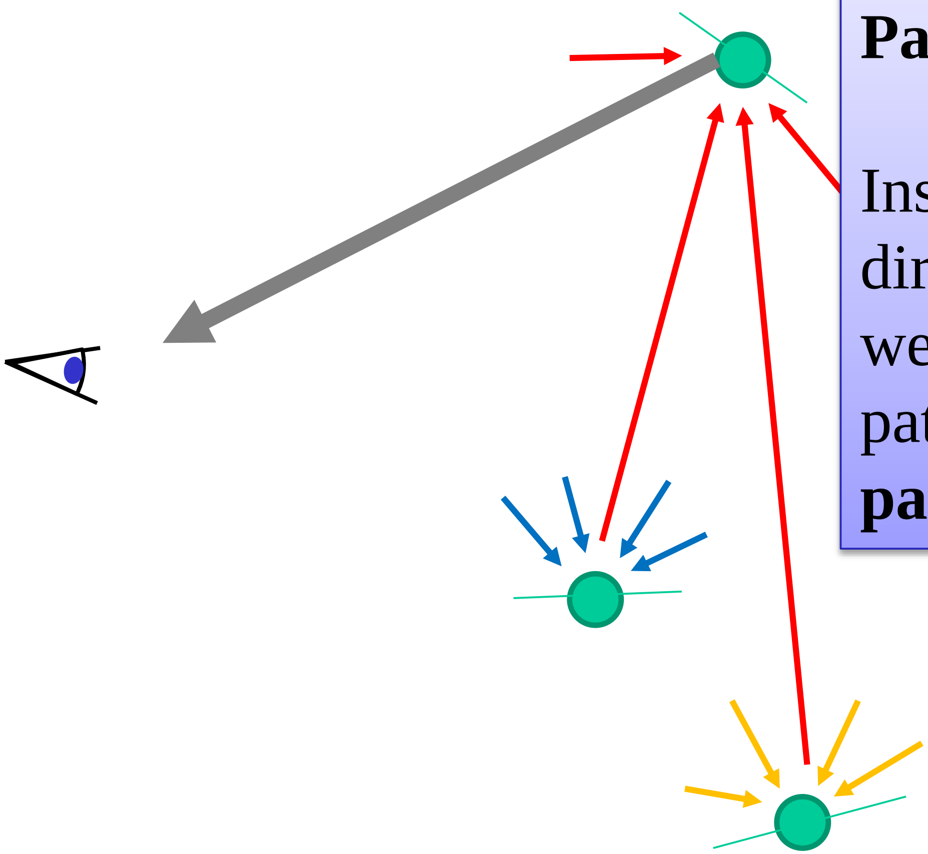
It is an integral over light coming into this point, so we can approximate it using a large enough random sample

Monte-carlo integration



Now in principle we would have to recursively continue, exploding the search space quickly

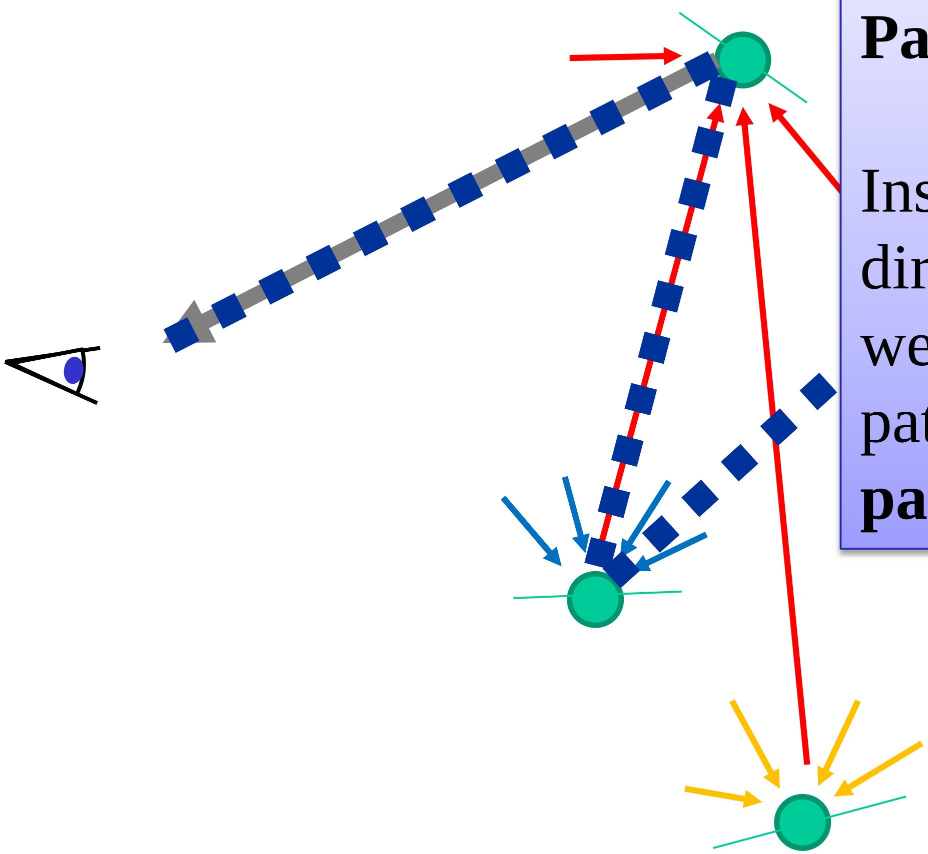
Monte-carlo integration



Path tracing

Instead of averaging directions at each step, we shoot several random paths and average **over paths**.

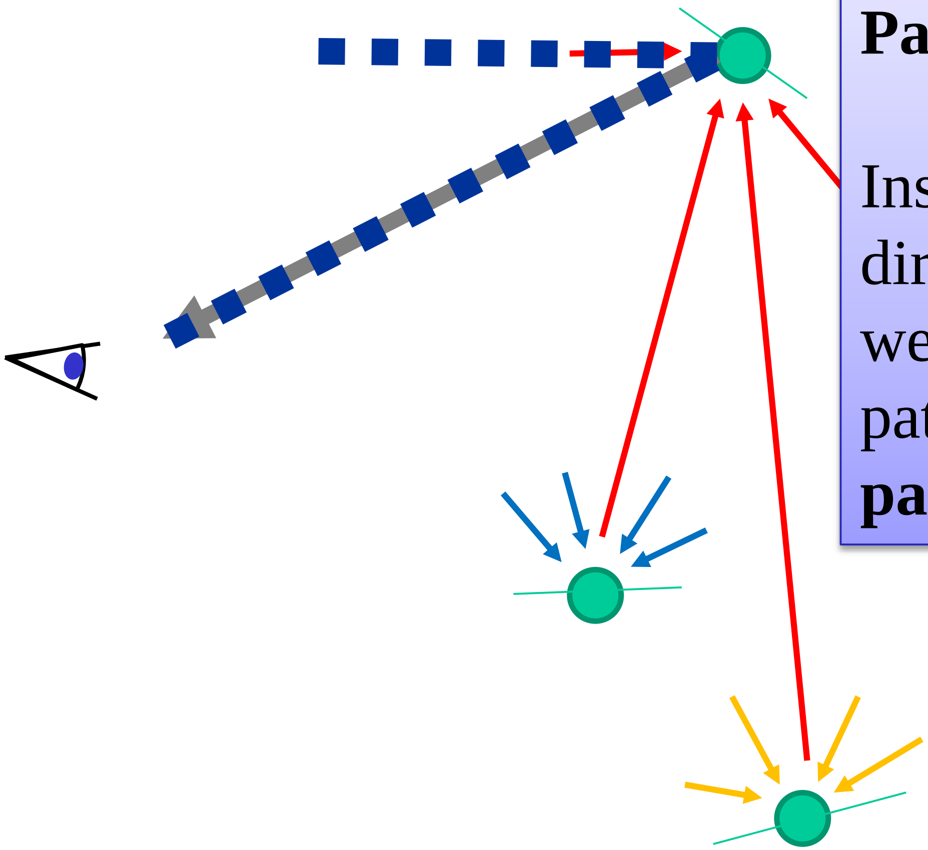
Monte-carlo integration



Path tracing

Instead of averaging directions at each step, we shoot several random paths and average **over paths**.

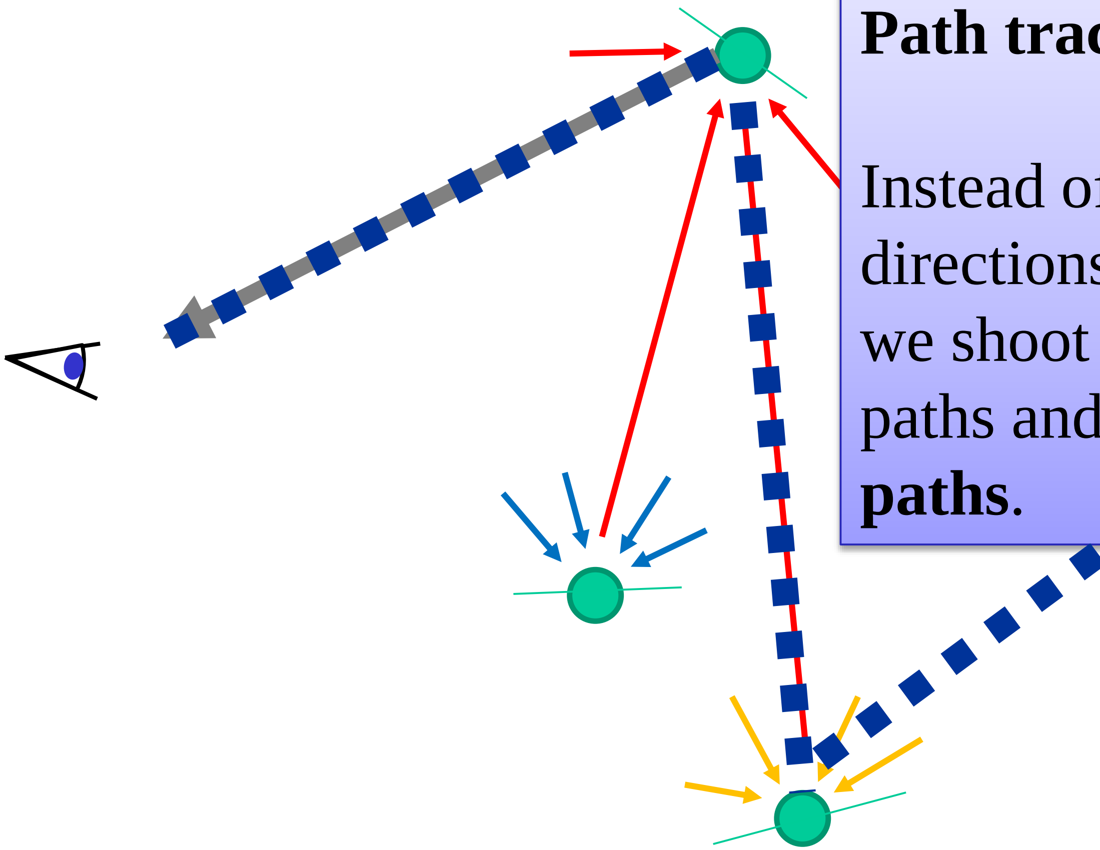
Monte-carlo integration



Path tracing

Instead of averaging directions at each step, we shoot several random paths and average **over paths**.

Monte-carlo integration



Path tracing

Instead of averaging directions at each step, we shoot several random paths and average **over paths**.

Path tracing

```
color trace_path(ray, depth) {
  hit = raycast_scene(ray);
  if (hit == None || depth is too large) return Black;
  else {
     dir = random_direction()
     new_ray = Ray(origin = hit.point,
                   direction = dir);

     incoming_color = trace_path(new_ray, depth+1)
     return hit.emittance +
             BRDF(hit.material, ray, new_ray) *
             dot(new_ray, hit.normal) *
             incoming_color
  }
```

Path tracing

- In simple terms:
  - Cast rays into the scene, each bouncing randomly around.
  - Rays that do not reach light sources contribute nothing.
  - Rays that reach light sources, contribute color, determined by the diffusion/reflectance properties of the surfaces they bounced off.

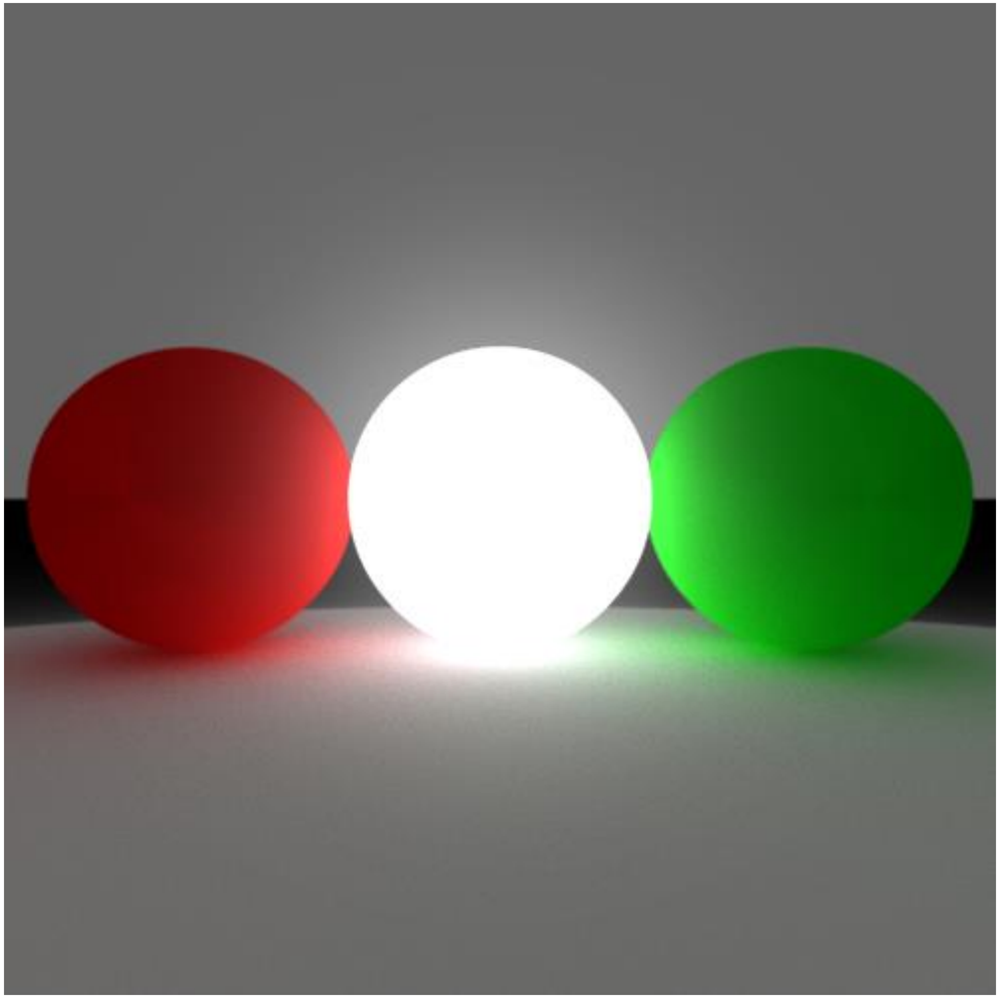
Quiz

- How is this different from conventional recursive raytracing?

Quiz

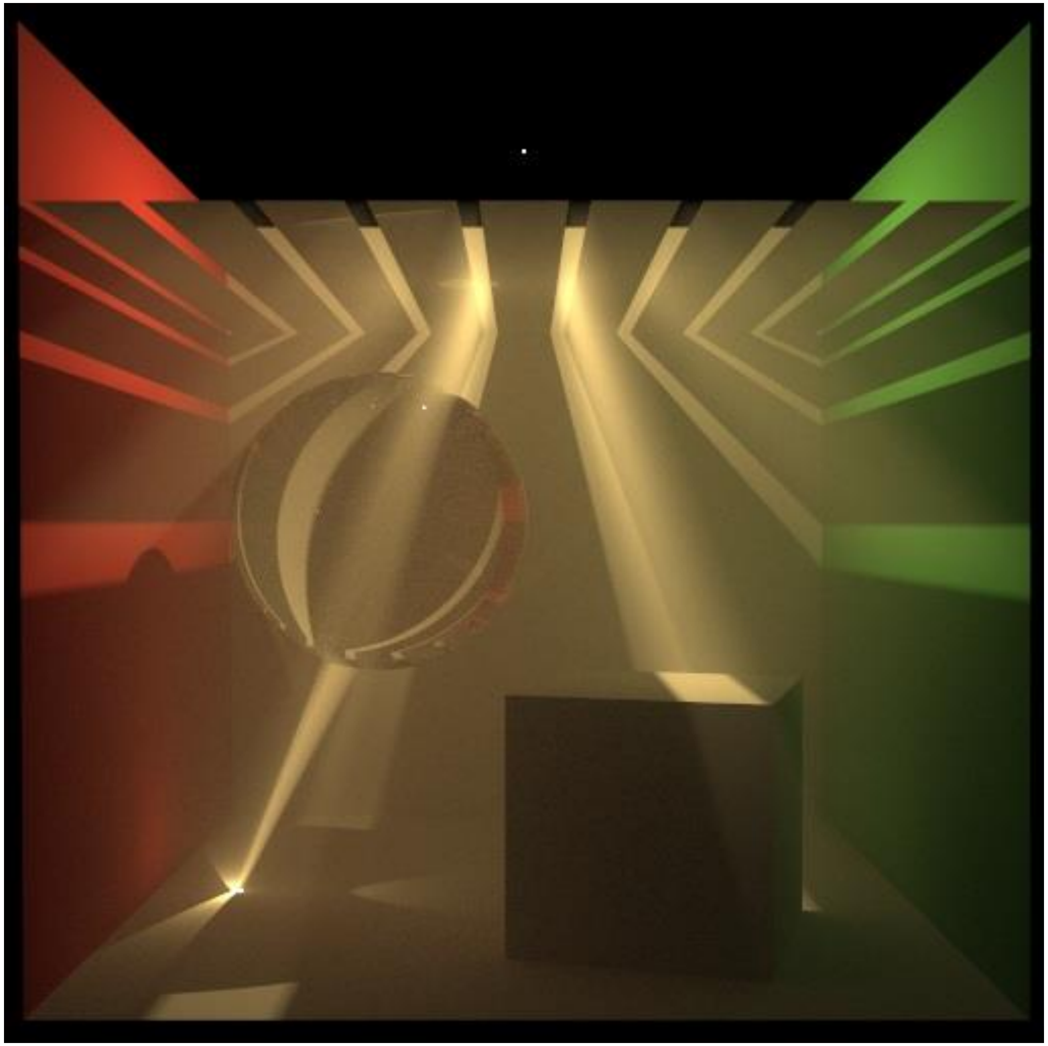
- How is this different from conventional recursive raytracing?
  - Rays bounce **randomly**, thus taking into account light diffusion, not only pure reflections or refractions.
  - **No light model computations** are performed at the bounce points. The ray **has to reach** a light source to contribute any color.

Path tracing



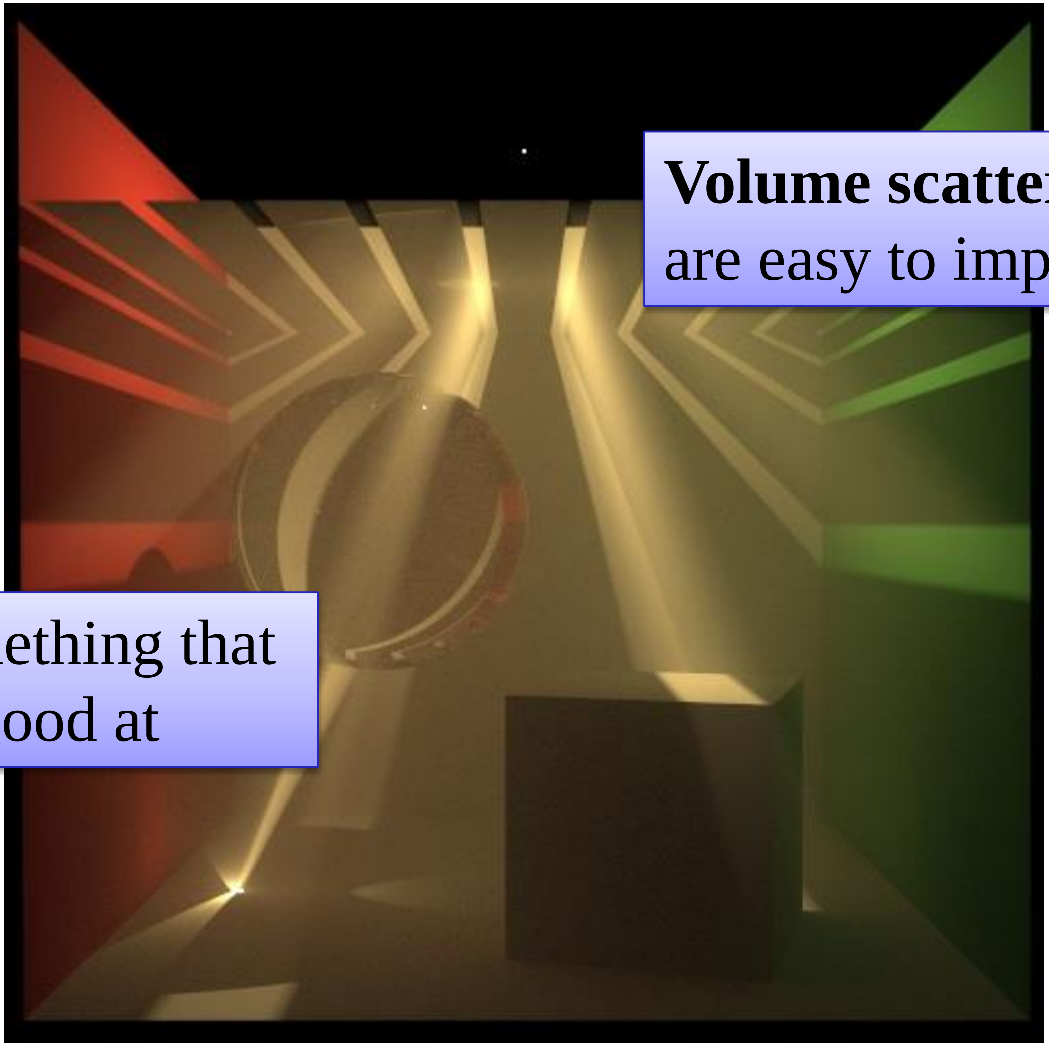
http://commons.wikimedia.org/wiki/File:Pathtrace3.png

Path tracing



http://www.sjbrown.co.uk/2012/07/15/bidirectional-path-tracing-in-participating-media/

Path tracing



Volume scattering effects are easy to implement

Caustics is something that path tracing is good at

http://www.sjbrown.co.uk/2012/07/15/bidirectional-path-tracing-in-participating-media/

Problem with Path tracing

- We have to trace the ray until it hits the light (or we are tired).

- If light sources are small (e.g. points), rays may never hit them (or hit extremely rarely).

Problem with Path tracing

- Solution?

Problem with Path tracing

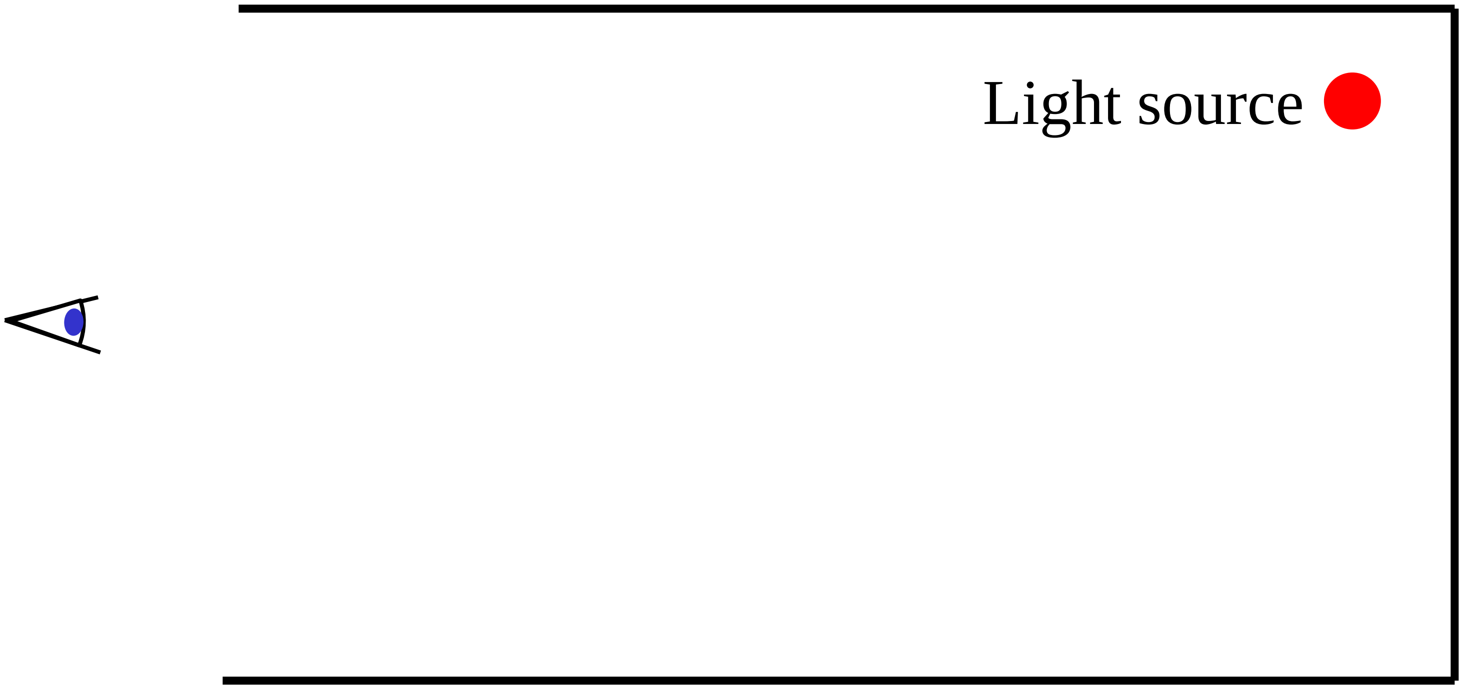
- Solution?

- We could try to trace the rays from light sources, but this leads to a similar problem: some rays wouldn't ever reach the camera.

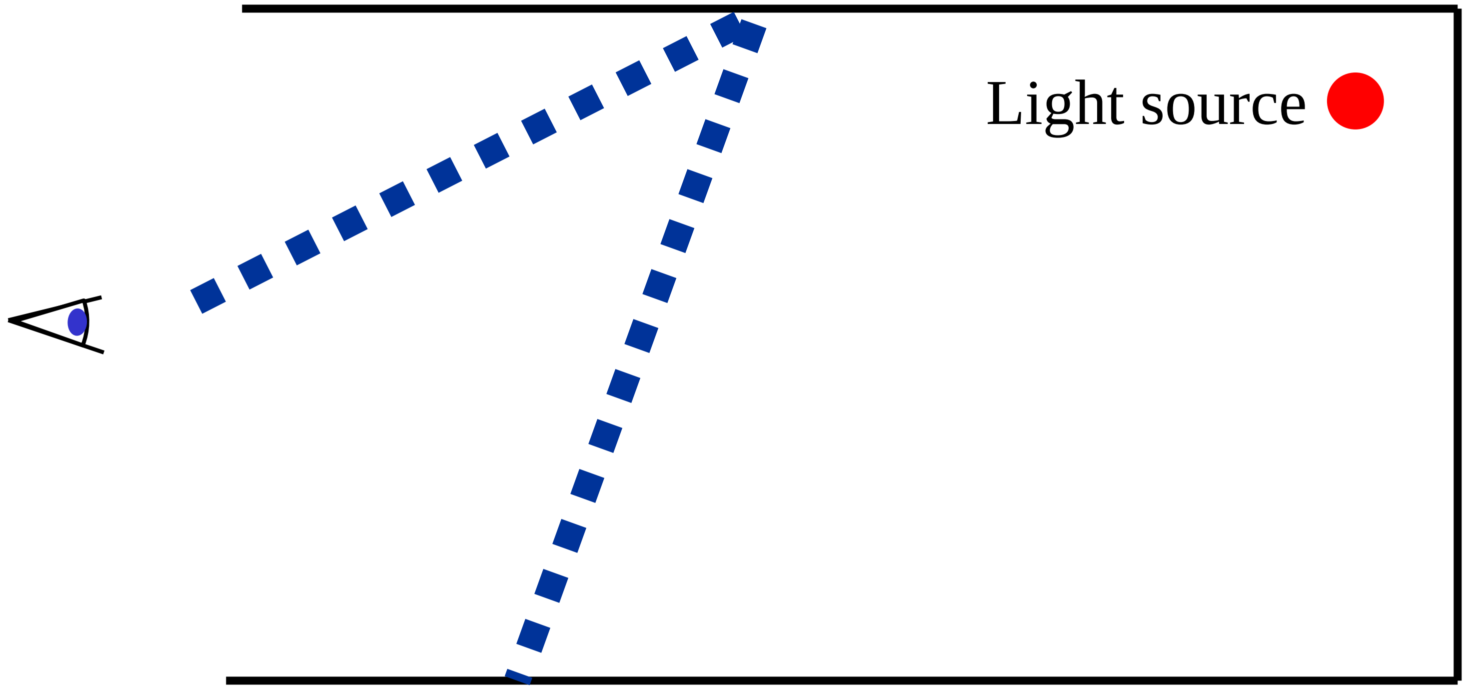
Bidirectional path tracing

- Solution:
  - Trace a path for some steps starting from the camera.
  - Trace another path for some steps starting from the light source.
  - Connect the two parts and process the result as a single random path.

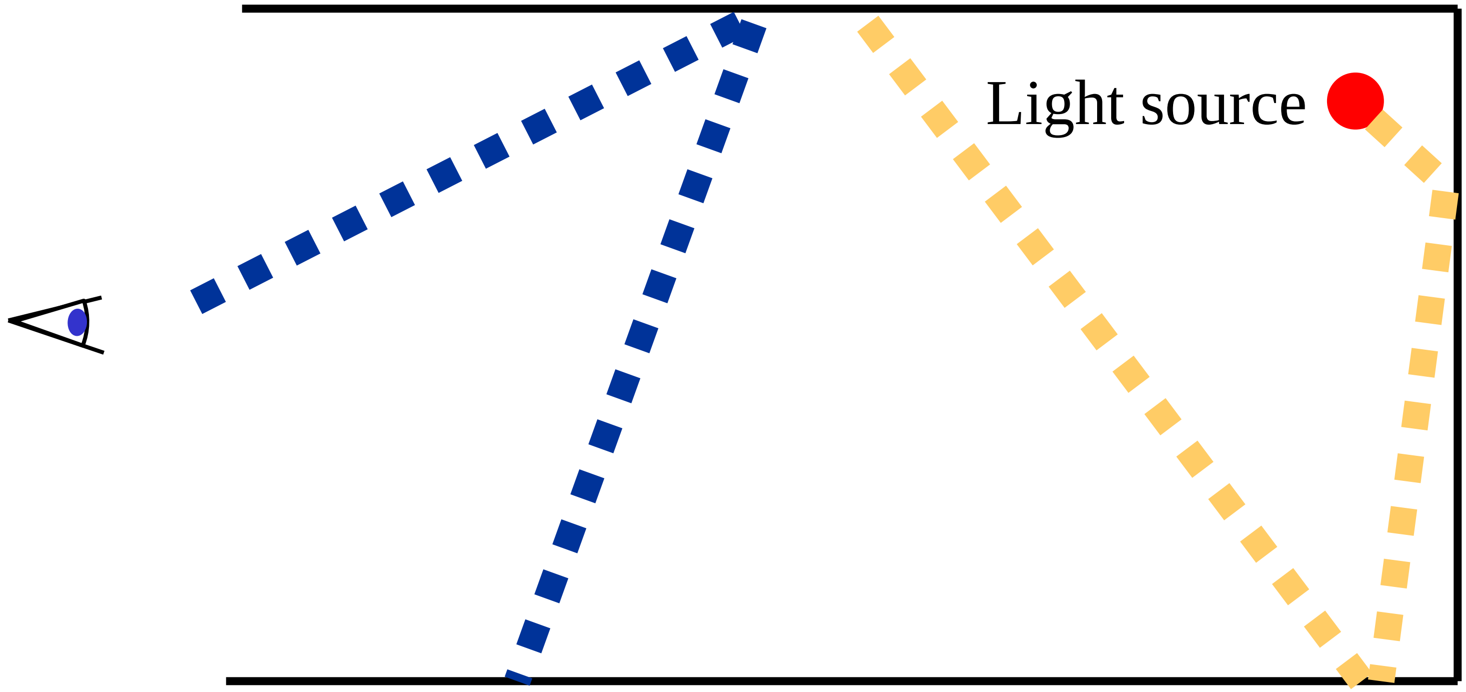
Bidirectional path tracing



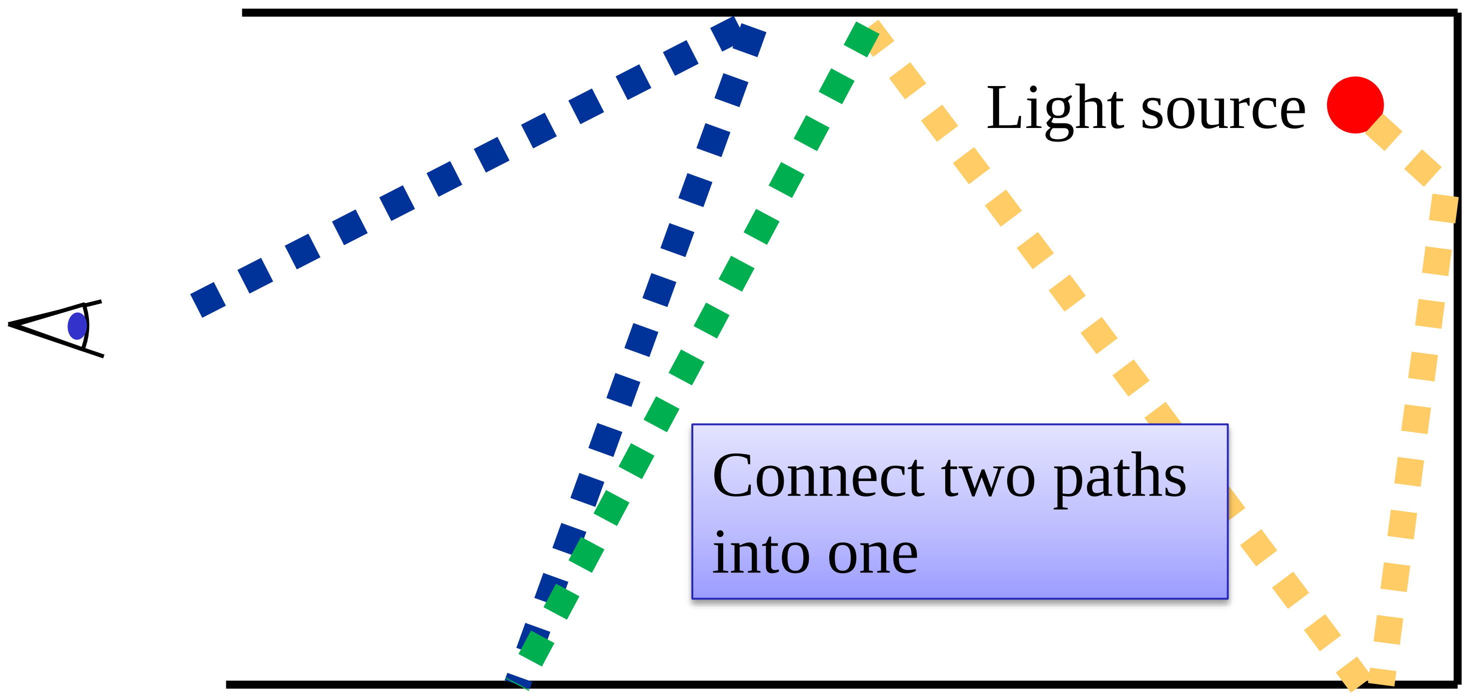
Bidirectional path tracing



Bidirectional path tracing



Bidirectional path tracing



Photon mapping

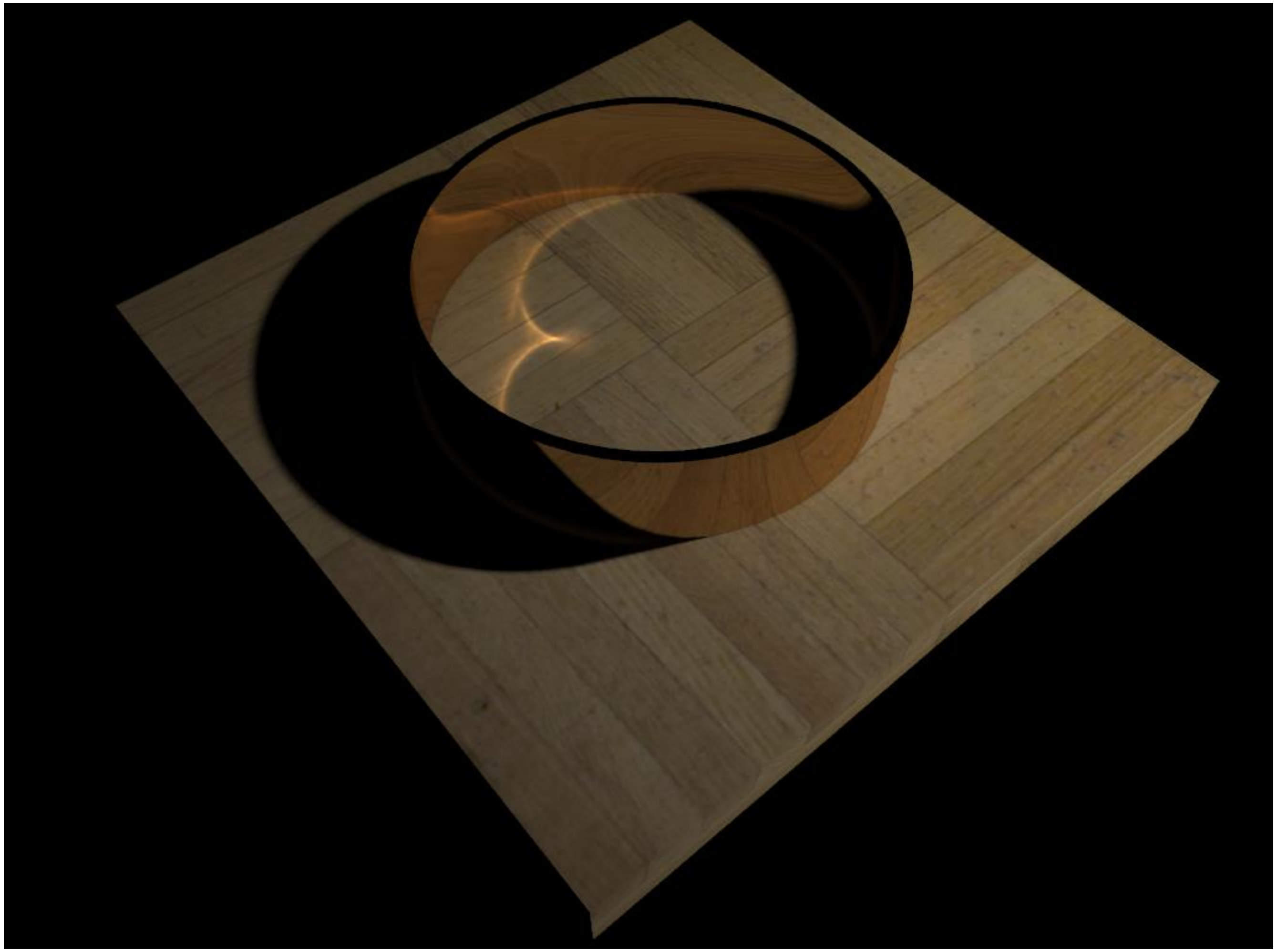
- A method “inbetween” path tracing and radiosity:
  - Simulate “photons” shooting from light sources in all directions.
  - When photon hits a surface, it is split into parts – part of its intensity “gets stuck” in the surface. Remaining intensity continues its path in some direction.
  - Eventually, many (~millions) of photons will be deposited all over the scene.

See H.W.Jensen. Global Illumination using Photon Maps

Photon mapping

- With each “stuck” photon we store the direction it came from and its color (intensity).
- We index all photons in a data structure for fast nearest-neighbor retrieval (e.g. k-d tree).
- Now for any point we can easily evaluate the rendering equation by replacing the integral with the sum over nearby photons.

Photon mapping



http://www.mpi-inf.mpg.de/departments/irg3/ws0304/lcn/projects/Michael/

Combined methods

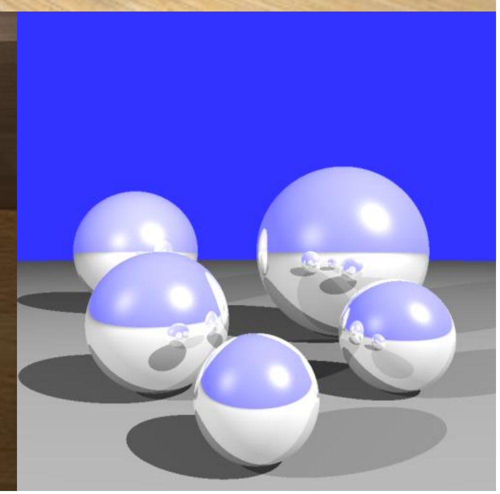
- Various combinations of methods are possible:
  - Radiosity light maps + standard pipeline
  - Radiosity / photon mapping + raytracing
  - Path tracing + Photon mapping (irradiance caching)

Summary: Core rendering methods

- **Standard graphics pipeline**
- **Direct raycasting**
  - Raytracing, Raymarching, Sphere tracing
- **Rendering equation solvers**
  - Radiosity, Path tracing, Photon mapping
- **Hybrid approaches**

Quiz





http://randomcontrol.com/blog/?p=243
http://filmesegames.com.br/2012/filme-de-world-of-warcraft-nao-conta-mais-com-sam-raimi/
http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter39.html
http://www.andreworlando.com/?p=64