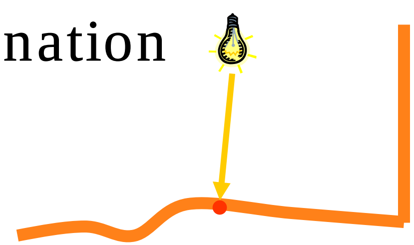
Computer graphics III – Rendering equation and its solution

Jaroslav Křivánek, MFF UK

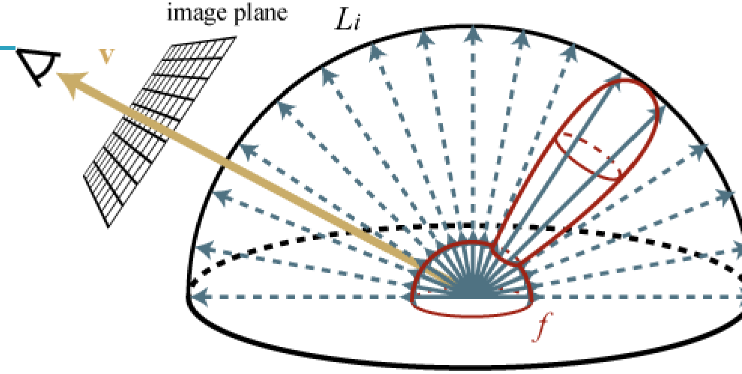
Jaroslav.Krivanek@mff.cuni.cz

Global illumination – GI

Direct illumination

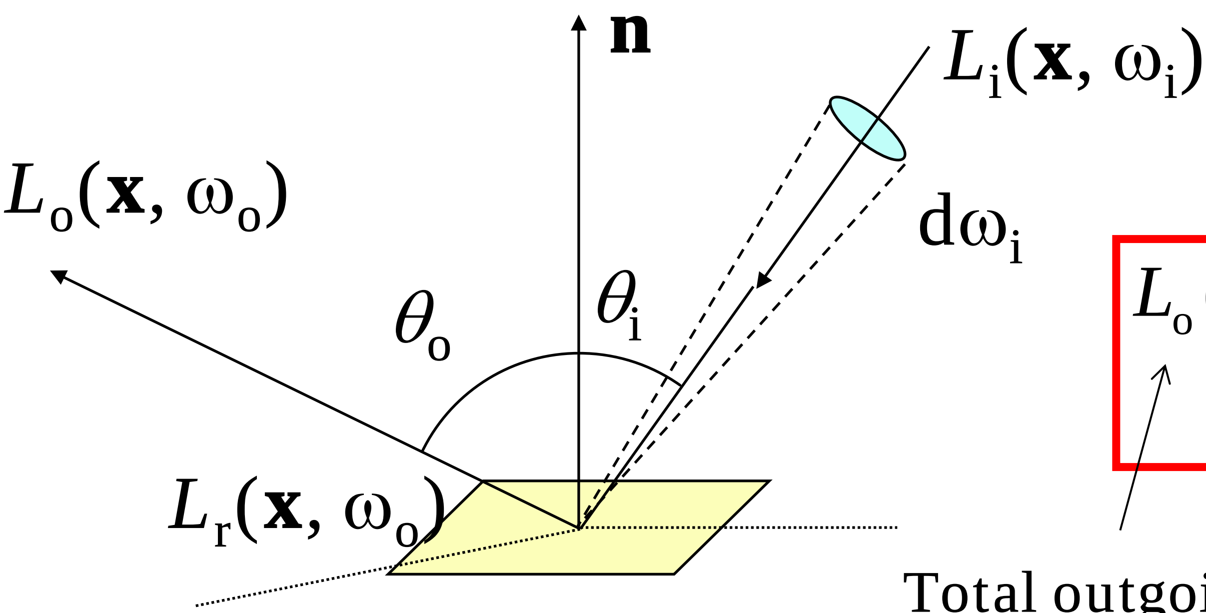


Review: Reflection equation



- "Sum" (integral) of contributions over the hemisphere:

$$\boxed{L_r(\mathbf{x},\omega_o) = \int_{H(\mathbf{x})} L_i(\mathbf{x},\omega_i)\cdot f_r(\mathbf{x},\omega_i \to \omega_o)\cdot \cos\theta_i \, d\omega_i}$$



Emitted radiance

$$\boxed{\begin{aligned} L_o(\mathbf{x},\omega_o) &= L_e(\mathbf{x},\omega_o) \\ &+ L_r(\mathbf{x},\omega_o) \end{aligned}}$$

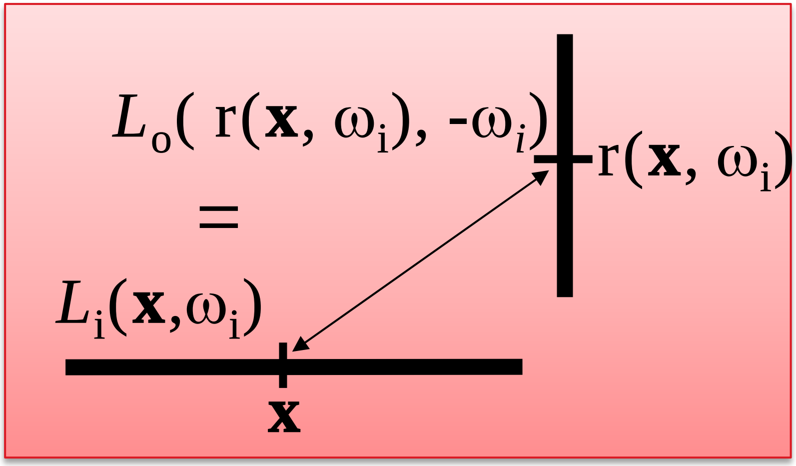
Total outgoing rad. Reflected rad.

From local reflection to global light transport

- Reflection equation (local reflection)

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{H(\mathbf{x})} L_i(\mathbf{x}, \omega_i) \cdot f_r(\mathbf{x}, \omega_i \to \omega_o) \cdot \cos\theta_i \, d\omega_i$$

- Where does the incoming radiance $L_i(\mathbf{x}, \omega_i)$ come from?
  - From other places in the scene !



$$\boxed{L_i(\mathbf{x}, \omega_i) = L_o(\mathrm{r}(\mathbf{x}, \omega_i), -\omega_i)}$$

Ray casting function

From local reflection to global light transport

- Plug for L_i into the reflection equation

$$L_\mathrm{o}(\mathbf{x},\omega_\mathrm{o}) = L_\mathrm{e}(\mathbf{x},\omega_\mathrm{o}) + \int_{H(\mathbf{x})} L_\mathrm{o}(\mathrm{r}(\mathbf{x},\omega_\mathrm{i}),-\omega_\mathrm{i}) \cdot f_r(\mathbf{x},\omega_\mathrm{i}\to\omega_\mathrm{o}) \cdot \cos\theta_\mathrm{i}\,\mathrm{d}\omega_\mathrm{i}$$

- Incoming radiance L_i drops out
- Outgoing radiance L_o at $\mathbf{x}$ described in terms of L_o at other points in the scene

Rendering equation

- Remove the subscript "o" from the outgoing radiance:

$$L(\mathbf{x},\omega_\mathrm{o}) = L_\mathrm{e}(\mathbf{x},\omega_\mathrm{o}) + \int_{H(\mathbf{x})} L(\mathrm{r}(\mathbf{x},\omega_\mathrm{i}),-\omega_\mathrm{i})\, f_r(\mathbf{x},\omega_\mathrm{i}\to\omega_\mathrm{o}) \cos\theta_\mathrm{i}\,\mathrm{d}\omega_\mathrm{i}$$

- Description of the steady state = **energy balance** in the scene
- **Rendering** = calculate $L(\mathbf{x}, \omega_o)$ for all points visible through pixels, such that it fulfils the rendering equation

Reflection equation vs. Rendering equation

Similar form – different meaning

$$L_o(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \int_{H(\mathbf{x})} L_i(\mathbf{x},\omega_i)\cdot f_r(\mathbf{x},\omega_i \to \omega_o)\cdot \cos\theta_i \,\mathrm{d}\omega_i$$

- **Reflection equation**
  - Describes **local light reflection** at a single point
  - Integral that can be used to calculate the outgoing radiance if we know the incoming radiance

$$L(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \int_{H(\mathbf{x})} L(\mathrm{r}(\mathbf{x},\omega_i),-\omega_i)\cdot f_r(\mathbf{x},\omega_i \to \omega_o)\cdot \cos\theta_i \,\mathrm{d}\omega_i$$

- **Rendering equation**
  - Condition on the **global distribution of light** in scene
  - Integral equation – unknown quantity L on both sides

Rendering Equation – Kajiya 1986

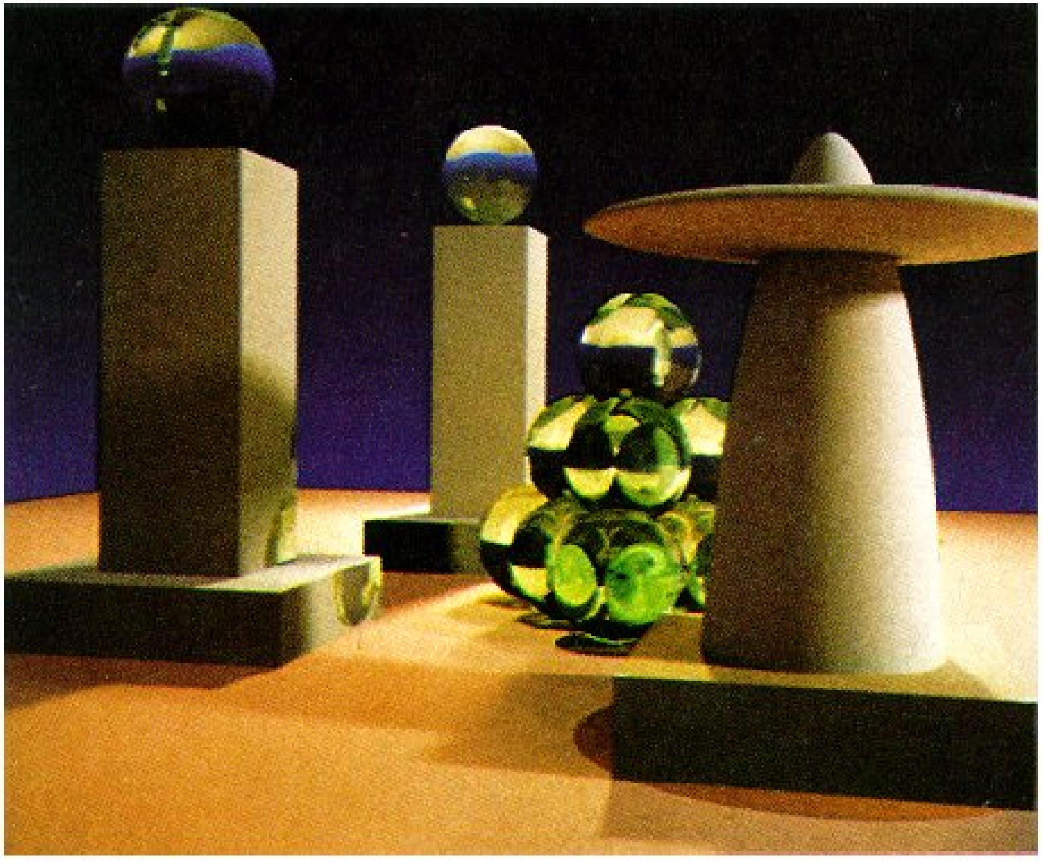


Figure 6. A sample image. All objects are neutral grey. Color on the objects is due to caustics from the green glass balls and color bleeding from the base polygon.

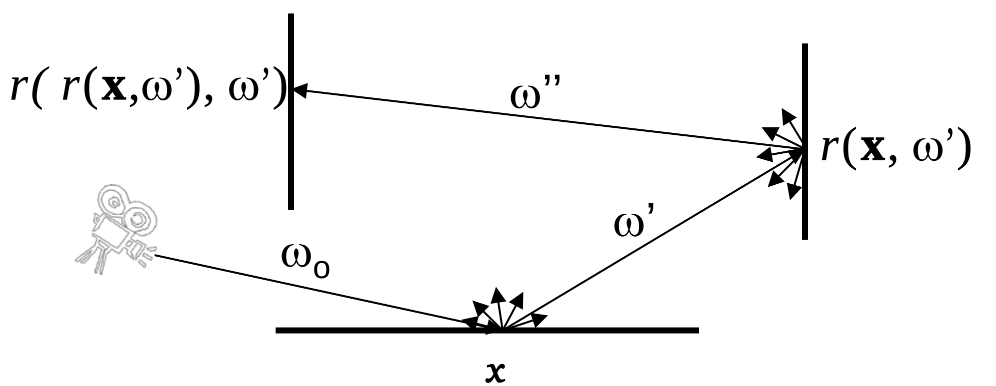
Path tracing sketch

Recursive unwinding of the RE

- Angular form of the RE

$$L(\mathbf{x},\omega_\mathrm{o}) = L_\mathrm{e}(\mathbf{x},\omega_\mathrm{o}) + \int_{H(\mathbf{x})} L(\mathrm{r}(\mathbf{x},\omega'),-\omega')\cdot f_r(\mathbf{x},\omega'\to\omega_\mathrm{o})\cdot\cos\theta'\,\mathrm{d}\omega'$$

- To calculate $L(\mathbf{x}, \omega_o)$, we need to calculate $L(\mathrm{r}(\mathbf{x}, \omega'), -\omega')$ for all directions ω' around the point $\mathbf{x}$
- For the calculation of each $L(\mathrm{r}(\mathbf{x}, \omega'), -\omega')$, we need to do the same thing recursively,
- etc.



Path tracing, v. zero (recursive form)

```
getLi (x, ω):
    y = traceRay(x, ω)
    return
        Le(y, –ω) +                              // emitted radiance
        Lr (y, –ω)                               // reflected radiance

Lr(x, ω):
    ω′ = genUniformHemisphereRandomDir( n(x) )
    return 2π * brdf(x, ω, ω′)  * dot(n(x), ω′) * getLi(x, ω′)
```

Back to the theory: Angular and area form of the rendering equation

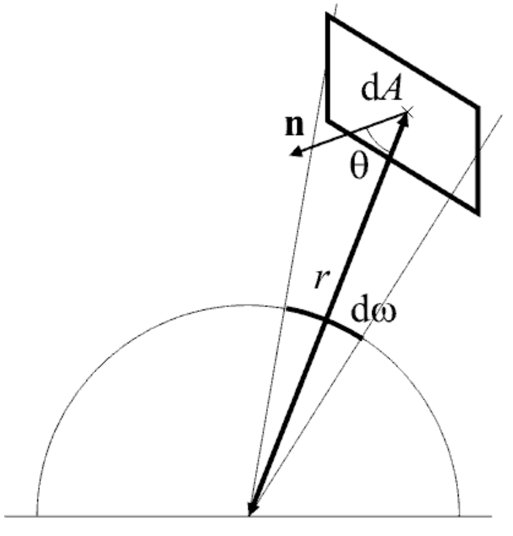
Angular vs. area form of the RE

- **Angular form**
  - integral over the hemisphere in incoming directions

$$L(\mathbf{x},\omega_{\mathrm{o}}) = L_{\mathrm{e}}(\mathbf{x},\omega_{\mathrm{o}}) + \int_{H(\mathbf{x})} L(\mathrm{r}(\mathbf{x},\omega_{\mathrm{i}}),-\omega_{\mathrm{i}}) \cdot f_r(\mathbf{x},\omega_{\mathrm{i}} \rightarrow \omega_{\mathrm{o}}) \cdot \cos\theta_{\mathrm{i}} \, \mathrm{d}\omega_{\mathrm{i}}$$

- Substitution

$$\mathrm{d}\omega = \mathrm{d}A \frac{\cos\theta}{r^2}$$



Angular vs. area form of the RE

- **Area form**
  - Integral over the scene surface

$$L(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \int_M L(\mathbf{y}\to\mathbf{x})\cdot f_r(\mathbf{y}\to\mathbf{x}\to\omega_o)\cdot G(\mathbf{x}\leftrightarrow\mathbf{y})\cdot V(\mathbf{x}\leftrightarrow\mathbf{y})\,\mathrm{d}A_{\mathbf{y}}$$

scene surface

geometry term

$$G(\mathbf{x}\leftrightarrow\mathbf{y}) = \frac{\cos\theta_{\mathbf{x}}\cdot\cos\theta_{\mathbf{y}}}{\left\|\mathbf{x}-\mathbf{y}\right\|^2}$$

visibility

1 ... **y** visible from **x**

0 ... otherwise

Angular form

- Add radiance contributions to a point from all directions
- For each direction, find the nearest surface
- Implementation in stochastic path tracing:
  - For a given **x**, generate random direction(s), for each find the nearest intersection, return the outgoing radiance at that intersection and multiply it with the cosine-weighted BRDF. Average the result of this calculation over all the generated directions over the hemisphere.

Area form

- Sum up contributions to a point from all other points on the scene surface
- Contribution added only if the two points are mutually visible
- Implementation in stochastic path tracing:
  - Generate randomly point **y** on scene geometry. Test visibility between **x** and **y**. If mutually visible, add the outgoing radiance at **y** modulated by the geometry factor.
- Typical use: **direct illumination calculation** for area light sources

Most rendering algorithms = (approximate) solution of the RE

- **Local illumination** (OpenGL)
  - Only point sources, integral becomes a sum
  - Does not calculate equilibrium radiance, is not really a solution of the RE

- **Finite element methods** (radiosity) [Goral, '84]
  - Discretize scene surface (finite elements)
  - Disregard directionality of reflections: everything is assumed to be diffuse
  - Cannot reproduce glossy reflections

Most rendering algorithms = (approximate) solution of the RE

- **Ray tracing** [Whitted, '80]
  - Direct illumination on diffuse and glossy surfaces due to point sources
  - Indirect illumination only on ideal mirror reflection / refractions
  - Cannot calculate indirect illumination on diffuse and glossy scenes, soft shadows etc. …
- **Distributed ray tracing** [Cook, '84]
  - Estimate the local reflection using the MC method
  - Can calculate soft shadows, glossy reflections, camera defocus blur, etc.

Most rendering algorithms = (approximate) solution of the RE

- **Path tracing** [Kajiya, '86]
  - True solution of the RE via the Monte Carlo method
  - Tracing of random paths (random walks) from the camera
  - Can calculate indirect illumination of higher order

From the rendering equation to finite element radiosity

From the rendering equation to radiosity

- Start from the area form of the RE:

$$L(\mathbf{x}, \omega_{\mathrm{o}}) = L_e(\mathbf{x}, \omega_{\mathrm{o}}) + \int_M L(\mathbf{y} \to \mathbf{x}) \cdot f_r(\mathbf{y} \to \mathbf{x} \to \omega_{\mathrm{o}}) \cdot G(\mathbf{x} \leftrightarrow \mathbf{y}) \cdot V(\mathbf{x} \leftrightarrow \mathbf{y}) \, \mathrm{d}A_{\mathbf{y}}$$

- The Radiosity method– **assumptions**
  - Only diffuse surfaces (BRDF constant in ω_i and ω_o)
  - Radiosity (i.e. radiant exitance) is spatially constant (flat) over the individual elements

From the rendering equation to radiosity

- **Diffuse surfaces only**
  - The BRDF is constant in ω_i and ω_o

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \frac{\rho(\mathbf{x})}{\pi} \int_M L(\mathbf{y} \to \mathbf{x}) \cdot G(\mathbf{x} \leftrightarrow \mathbf{y}) \cdot V(\mathbf{x} \leftrightarrow \mathbf{y}) \, \mathrm{d}A_{\mathbf{y}}$$

  - **Outgoing radiance is independent of $\boldsymbol{\omega_o}$** and it is equal to radiosity *B* divided by π

$$B(\mathbf{x}) = B_e(\mathbf{x}) + \rho(\mathbf{x}) \cdot \int_M B(\mathbf{y}) \cdot \underbrace{\frac{G(\mathbf{x} \leftrightarrow \mathbf{y}) \cdot V(\mathbf{x} \leftrightarrow \mathbf{y})}{\pi}}_{G'(\mathbf{x} \leftrightarrow \mathbf{y})} \, \mathrm{d}A_{\mathbf{y}}$$

From the rendering equation to radiosity

- Spatially constant (flat) radiosity B **of the contributing surface elements**

$$B(\mathbf{x}) = B_{\mathrm{e}}(\mathbf{x}) + \rho(\mathbf{x}) \cdot \sum_{j=1}^{N} B_j \cdot \underbrace{\left(\int_{A_j} G'(\mathbf{x} \leftrightarrow \mathbf{y}) \,\mathrm{d}A_{\mathbf{y},j} \right)}$$

Radiosity of the j-th element

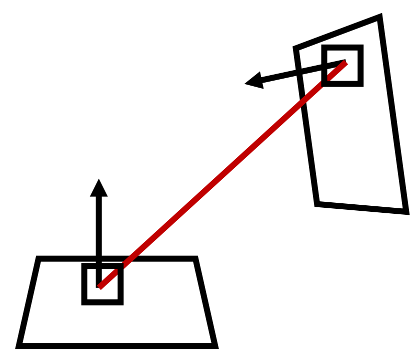
Geometry factor between surface element j and point $\mathbf{x}$

From the rendering equation to radiosity

- Spatially constant (flat) radiosity of the **receiving surface element** *i*:
  - Average radiosity over the element

$$B_i = \frac{1}{A_i} \int_{A_i} B(\mathbf{x}) \,\mathrm{d}A_i =$$

$$= B_{\mathrm{e},i} + \rho_i \cdot \sum_{j=1}^{N} B_j \cdot \underbrace{\left(\frac{1}{A_i} \int_{A_i} \int_{A_j} G'(\mathbf{x} \leftrightarrow \mathbf{y}) \,\mathrm{d}A_{\mathbf{y},j} \,\mathrm{d}A_{\mathbf{x},i} \right)}_{F_{ij} \text{ ... form factor}}$$



Classic radiosity equation

- System of linear equations

$$\boxed{B_i = B_{e,i} + \rho_i \cdot \sum_{j=1}^{N} B_j \cdot F_{ij}}$$

- Form factors

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} G'(\mathbf{x} \leftrightarrow \mathbf{y}) \, \mathrm{d}A_{\mathbf{y},j} \, \mathrm{d}A_{\mathbf{x},i}$$

- **Conclusion**: the radiosity method is nothing but a way to solve the RE under a specific set of assumptions

Radiosity method

- **Classical radiosity**
  1. Form facto calculation (Monte Carlo, hemicube, …)
  2. Solve the linear system (Gathering, Shooting, …)

- **Stochastic radiosity**
  - Avoids explicit calculation of form factors
  - Metoda Monte Carlo

- **Radiosity is not practical, not used**
  - Scene subdivision -> sensitive to the quality of the geometry model (but in reality, models are always broken)
  - High memory consumption, complex implementation

The operator form of the RE

RE is a Fredhom integral equation of the 2nd kind

General form the Fredholm integral equation of the 2nd kind

$$\boxed{f(x)} = \boxed{g(x)} + \int \boxed{k(x,x')} \boxed{f(x')} \boxed{dx'}$$

unknown function

known functions

equation "kernel"

Rendering equation:

$$\boxed{L(\mathbf{x},\omega_\mathrm{o})} = \boxed{L_\mathrm{e}(\mathbf{x},\omega_\mathrm{o})} + \int_{H(\mathbf{x})} \boxed{L(\mathrm{r}(\mathbf{x},\omega_\mathrm{i}),-\omega_\mathrm{i})} \cdot \boxed{f_r(\mathbf{x},\omega_\mathrm{i}\to\omega_\mathrm{o}) \cdot \cos\theta_\mathrm{i}} \, \mathrm{d}\omega_\mathrm{i}$$

Linear operators

- Linear operators **act** on functions
  - (as matrices act on vectors)

$$h(x) = (L \circ f)(x)$$

- The operator is **linear** if the "acting" is a linear operation

$$L \circ (af + bg) = a(L \circ f) + b(L \circ g)$$

- Examples of linear operators

$$(K \circ f)(x) \equiv \int k(x, x') f(x') \, dx'$$

$$(D \circ f)(x) \equiv \frac{\partial f}{\partial x}(x)$$

Transport operator

$$(T \circ L)(\mathbf{x}, \omega_\mathrm{o}) \equiv \int_{H(\mathbf{x})} L(\mathbf{x}, \omega_\mathrm{i}) \cdot f_r(\mathbf{x}, \omega_\mathrm{i} \to \omega_\mathrm{o}) \cdot \cos\theta_\mathrm{i} \, \mathrm{d}\omega_\mathrm{i}$$

- Rendering equation

$$\boxed{L = L_\mathrm{e} + T \circ L}$$

Solution of the RE in the operator form

- Rendering equation

$$L = L_e + T \circ L$$

- Formal solution

$$(I - T) \circ L = L_e$$

$$\boxed{L = (I - T)^{-1} \circ L_e}$$

  - unusable in practice – the inverse cannot be explicitly calculated

Expansion of the rendering equation

- Recursive substitution L

$$\begin{aligned} L &= L_\mathrm{e} + TL \\ &= L_\mathrm{e} + T\left(L_\mathrm{e} + TL\right) \\ &= L_\mathrm{e} + TL_\mathrm{e} + T^2 L \end{aligned}$$

- n-fold repetition yields the Neumann series

$$L = \sum_{i=0}^{n} T^i L_\mathrm{e} + T^{n+1} L$$

Expansion of the rendering equation

- If T is a **contraction** (tj. $||T|| < 1$, which holds for the RE), then

$$\lim_{n\to\infty} T^{n+1} L = 0$$

- **Solution of the rendering equation** is then given by

$$L = \sum_{i=0}^{\infty} T^i L_\mathrm{e}$$

A different derivation of the Neumann series

- Formal solution of the rendering equation

$$L = (I - T)^{-1} \circ L_{\mathrm{e}}$$

- Proposition

$$(I - T)^{-1} = I + T + T^2 + \ldots$$

- Proof

$$\begin{aligned}(I - T) \circ (I - T)^{-1} &= (I - T) \circ (I + T + T^2 + \ldots) \\ &= (I + T + T^2 + \ldots) - (T + T^2 + T^3 + \ldots) \\ &= I\end{aligned}$$

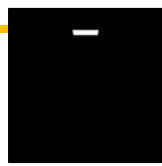
Rendering equation

$$\boxed{L = L_e + T \circ L}$$

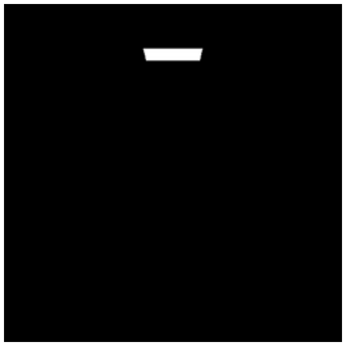
 $=$  $+\ T \circ$ 

- **Solution**: Neumann series

$$\boxed{L = L_e + TL_e + T^2 L_e + T^3 L_e + \ldots}$$

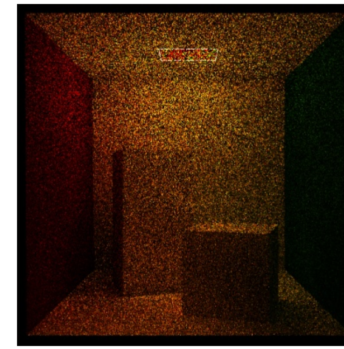
Progressive approximation



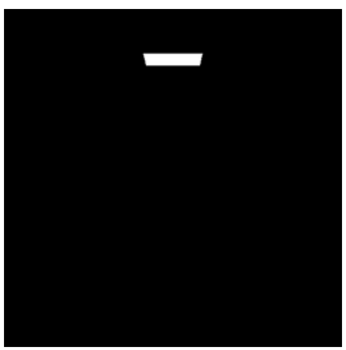
L_e

$T \circ L_e$

$T \circ T \circ L_e$

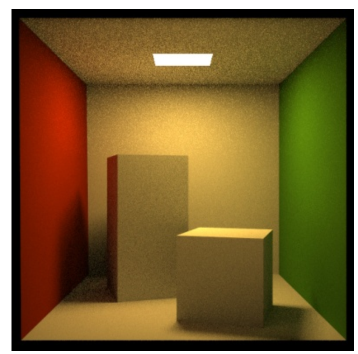


$T \circ T \circ T \circ L_e$

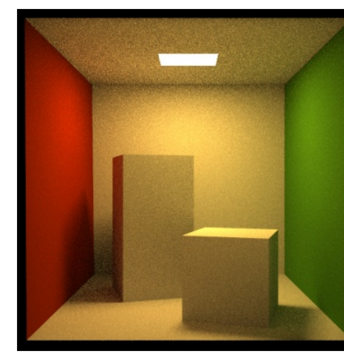


L_e

$L_e + T \circ L_e$



$L_e + TL_e + T^2 L_e$



$L_e + \ldots + T^3 L_e$

Progressive approximation

- Each application of *T* corresponds to one step of reflection & light propagation

$$L = \boxed{L_e + TL_e} + T^2 L_e + T^3 L_e + \dots$$

emission

direct illumination

one-bounce indirect illumination

two-bounce indirect illumination

OpenGL shading

Contractivity of T

- Holds for all physically correct models
  - Follows from the conservation of energy
- It means that repetitive application of the operator lower the remaining light energy (makes sense, since reflection/refraction cannot create energy)
- Scenes with white or highly specular surfaces
  - reflectivity close to 1
  - to achieve convergence, we need to simulate more bounces of light

Alright, so what have we achieved?

Rendering equation	Solution through the Neumann series
$L = L_e + T \circ L$	$L = \sum_{i=0}^{\infty} T^i L_e$

- We have replaced an integral equation by a sum of simple integrals
- Great we know how to calculate integrals numerically (the Monte Carlo method), which means that we know how to solve the RE, and that means that we can render images, yay!
- Recursive application to T corresponds to the recursive ray tracing from the camera

What exact integral are we evaluating, then?

$$L(\mathbf{x}, \omega_\mathrm{o}) = L_e(\mathbf{x}, \omega_\mathrm{o}) +$$

$$\boxed{\int_M} L_e(\mathbf{y} \to \mathbf{x}) \cdot f_r(\mathbf{y} \to \mathbf{x} \to \omega_\mathrm{o}) \cdot G(\mathbf{x} \leftrightarrow \mathbf{y}) \cdot V(\mathbf{x} \leftrightarrow \mathbf{y}) \, \mathrm{d}A_\mathbf{y} +$$

$$\boxed{\iint_M} L_e(\mathbf{z} \to \mathbf{y}) \cdot \left[f_r(\mathbf{z} \to \mathbf{y} \to \mathbf{x}) \cdot G(\mathbf{y} \leftrightarrow \mathbf{z}) \cdot V(\mathbf{y} \leftrightarrow \mathbf{z}) \right] \cdot$$
$$\cdot \left[f_r(\mathbf{y} \to \mathbf{x} \to \omega_\mathrm{o}) \cdot G(\mathbf{x} \leftrightarrow \mathbf{y}) \cdot V(\mathbf{x} \leftrightarrow \mathbf{y}) \right] \mathrm{d}A_\mathbf{y} \mathrm{d}A_\mathbf{z} +$$

$$\boxed{\iiint_M} L_e(\check{\mathbf{z}} \to \mathbf{z}) \ldots \mathrm{d}A_\mathbf{y} \mathrm{d}A_\mathbf{z} \mathrm{d}A_{\check{\mathbf{z}}}$$

Paths vs. recursion: Same thing, depends on how we look at it

- Paths in a high-dimensional path space

$$L = L_e + TL_e + T^2 L_e + T^3 L_e + \ldots$$

- Recursive solution of a series of nested (hemi)spherical integrals:

$$L = L_e + T(L_e + T(L_e + T(L_e + \ldots$$

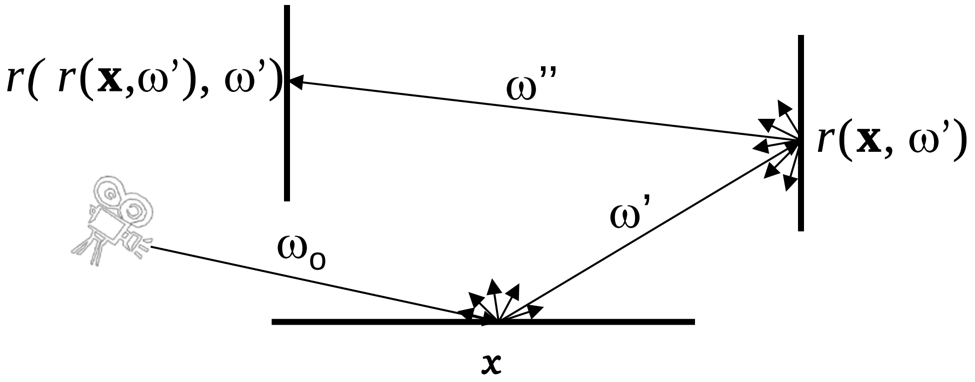
Recursive interpretation

We've seen this already, right? But unlike at the beginning of the lecture, by now we know this actually solves the RE.

- Angular form of the RE

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{H(\mathbf{x})} L(\mathrm{r}(\mathbf{x}, \omega'), -\omega') \cdot f_r(\mathbf{x}, \omega' \to \omega_o) \cdot \cos\theta' \, \mathrm{d}\omega'$$

- To calculate $L(\mathbf{x}, \omega_o)$ I need to calculate $L(\mathrm{r}(\mathbf{x}, \omega'), -\omega')$ for all directions ω' around the point **x**.
- For the calculation of each $L(\mathrm{r}(\mathbf{x}, \omega'), -\omega')$ I need to do the same thing recursively
- etc.



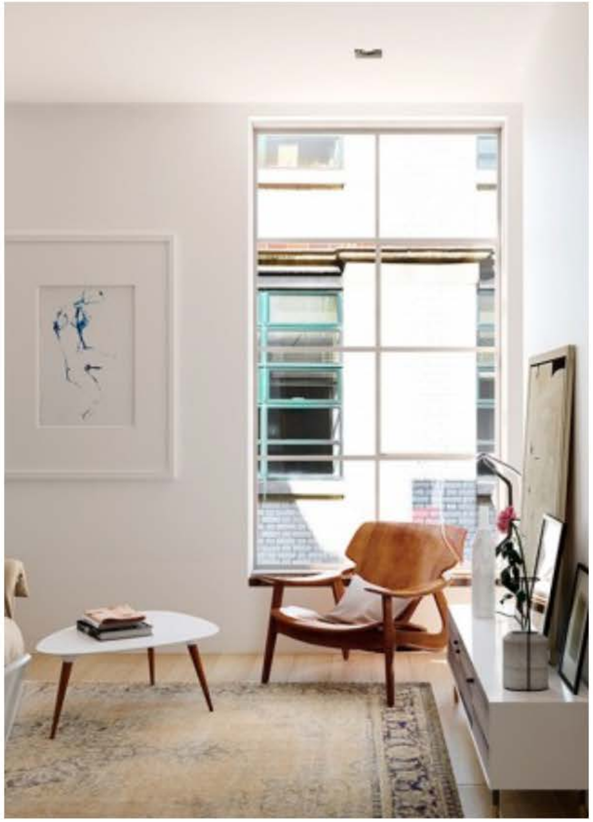
Path tracing, v. zero (recursive form)

```
getLi (x, ω):
    y = traceRay(x, ω)
    return
        Le(y, –ω) +                         // emitted radiance
        Lr (y, –ω)                          // reflected radiance

Lr(x, ω):
    ω′ = genUniformHemisphereRandomDir( n(x) )
    return 2π * brdf(x, ω, ω′) * dot(n(x), ω′) * getLi(x, ω′)
```

Path tracing, v. 2012, Arnold Renderer

© 2012 Columbia Pictures Industries, Inc. All Rights Reserved.





corona

Path tracing [Kajiya86]

- Only one secondary ray at each intersection
  - Random selection of interaction (diffuse reflection, refraction, etc., ...)
- Direct illumination: two strategies
  - Hope that the generated secondary ray hits the light source, or
  - Explicitly pick a point on the light source
- Trace hundreds of paths through each pixel and average the result
- Advantage over distributed ray tracing: now branching of the ray tree means no explosion of the number of rays with recursion depth