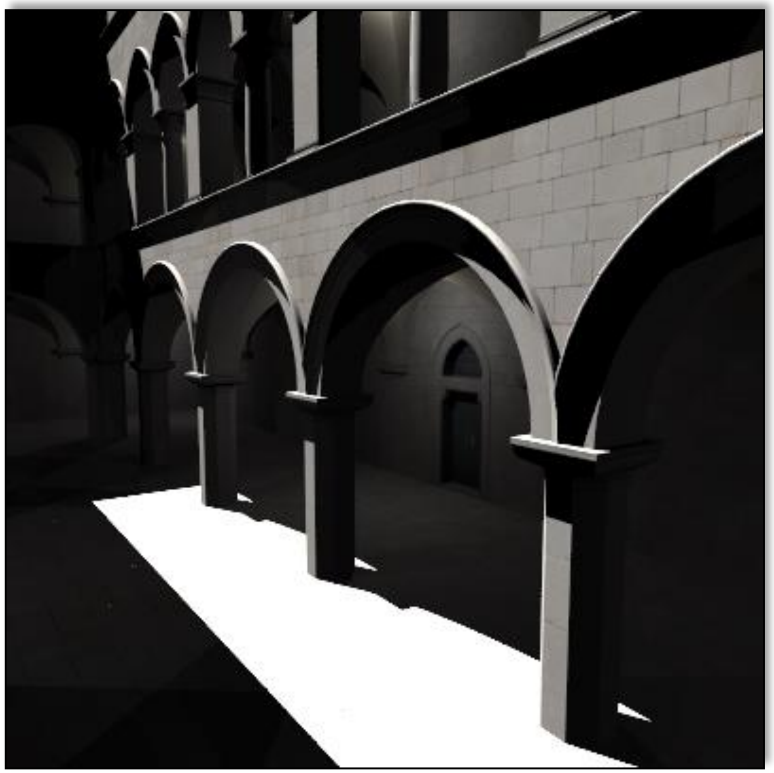
Computer graphics III – Rendering equation and its solution

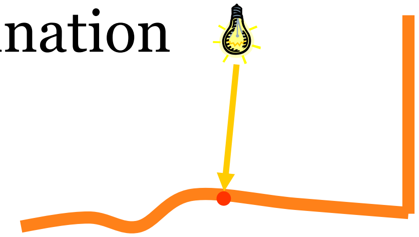
Jaroslav Křivánek, MFF UK

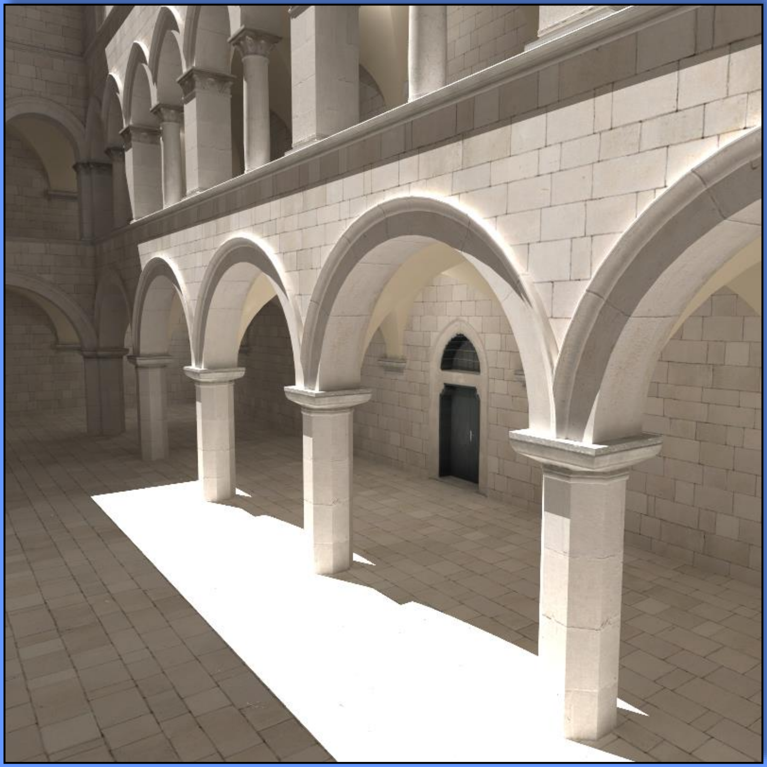
Jaroslav.Krivanek@mff.cuni.cz

Goal: Global illumination (GI)

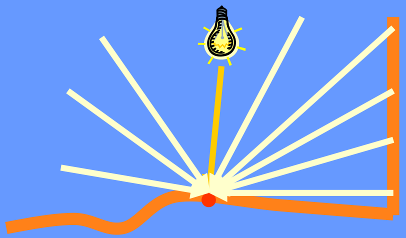


Direct illumination

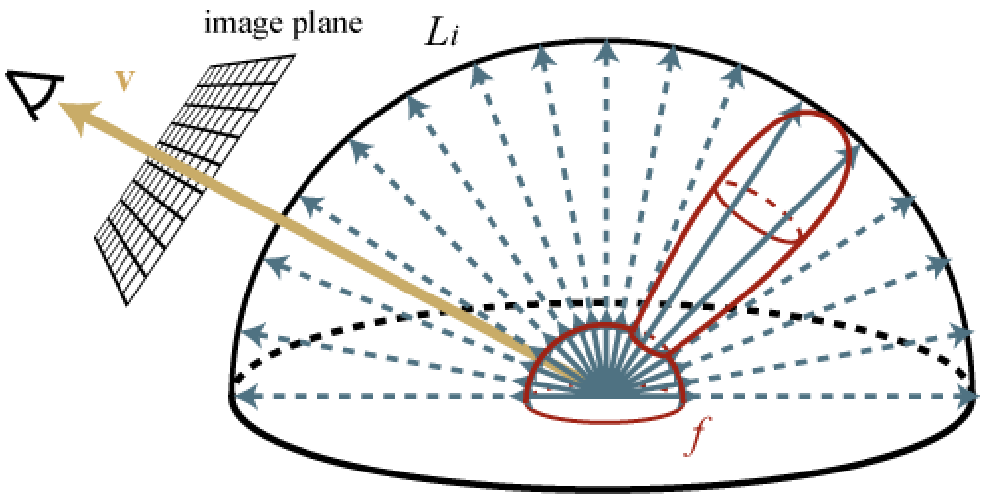




Global =
direct +
indirect



Review: Reflection equation



$$L_{\text{refl}}(x, \omega_{\text{out}}) = \int_{H(\mathbf{x})} L_{\text{in}}(x, \omega_{\text{in}}) \cdot f_r(x, \omega_{\text{in}} \to \omega_{\text{out}}) \cdot \cos\theta_{\text{in}} \;\, \mathrm{d}\omega_{\text{in}}$$

Review: Reflection equation

- If the shading point x itself is on an emitting surface (i.e. if x is on a light source):

Emitted radiance

$$L_{\mathrm{out}}(x, \omega_{\mathrm{out}}) = L_{\mathrm{e}}(x, \omega_{\mathrm{out}}) + L_{\mathrm{refl}}(x, \omega_{\mathrm{out}})$$

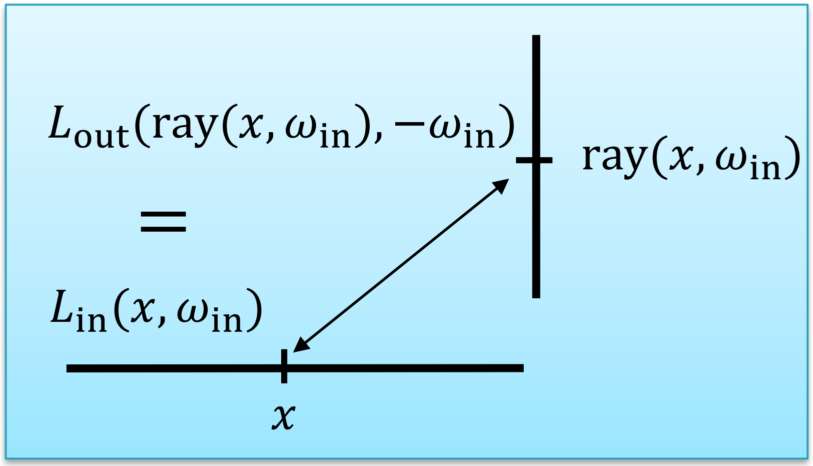
Reflected radiance (previous slide)

Total outgoing radiance

From local reflection to global light transport

- Where does the incoming radiance $L_{\text{in}}(x, \omega_{\text{in}})$ come from, really?
  - From other places in the scene!

$$\boxed{L_{\text{in}}(x,\omega_{\text{in}}) = L_{\text{out}}(\underbrace{\text{ray}(x,\omega_{\text{in}})}_{\text{Ray casting function}}, -\omega_{\text{in}})}$$


$L_{\text{out}}(\text{ray}(x,\omega_{\text{in}}), -\omega_{\text{in}})$
$=$
$L_{\text{in}}(x,\omega_{\text{in}})$
$\text{ray}(x,\omega_{\text{in}})$
x

From local reflection to global light transport

- Plug for L_{in} into the reflection equation

$$L_{\mathrm{out}}(x, \omega_{\mathrm{out}}) = L_{\mathrm{e}}(x, \omega_{\mathrm{out}}) +$$

$$\int_{H(\mathbf{x})} L_{\mathrm{out}}(\mathrm{ray}(x, \omega_{\mathrm{in}}), -\omega_{\mathrm{in}}) \cdot f_r(x, \omega_{\mathrm{in}} \to \omega_{\mathrm{out}}) \cdot \cos\theta_{\mathrm{in}} \, \mathrm{d}\omega_{\mathrm{in}}$$

- Incoming radiance L_{in} drops out

- Outgoing radiance L_{out} at x is described in terms of L_{out} at other points in the scene

Rendering equation

$$L(x, \omega_{\mathrm{out}}) = L_{\mathrm{refl}}(x, \omega_{\mathrm{out}}) + \int_{H(\mathbf{x})} L(\mathrm{ray}(x, \omega_{\mathrm{in}}), -\omega_{\mathrm{in}}) \cdot f_r(x, \omega_{\mathrm{in}} \to \omega_{\mathrm{out}}) \cdot \cos\theta_{\mathrm{in}} \; \mathrm{d}\omega_{\mathrm{in}}$$

- Removed the subscript "out" from the outgoing radiance for brevity
- The RE describes the steady state = **energy balance** in the scene
- **Rendering** = calculate $L(x, \omega_{\mathrm{out}})$ for all points visible through pixels, such that it fulfils the rendering equation

Reflection equation vs. Rendering equation

Similar form – different meaning

- **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

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

Rendering Equation – Kajiya 1986

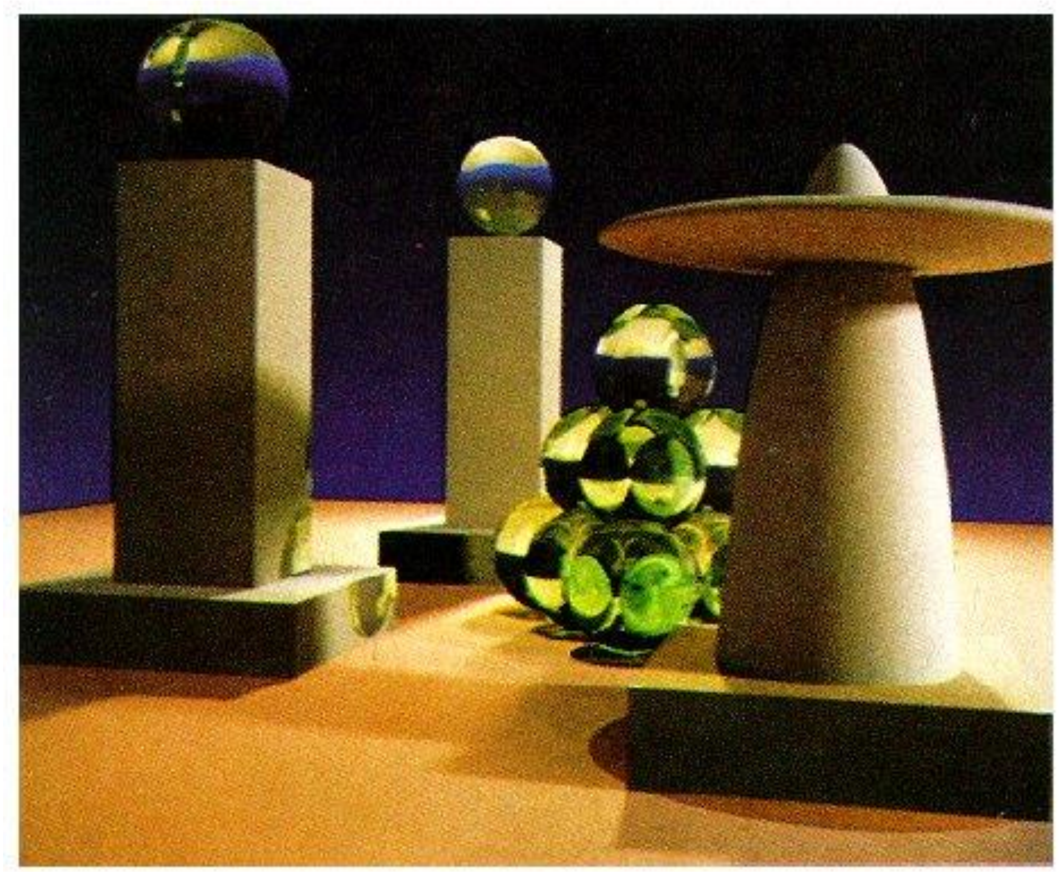
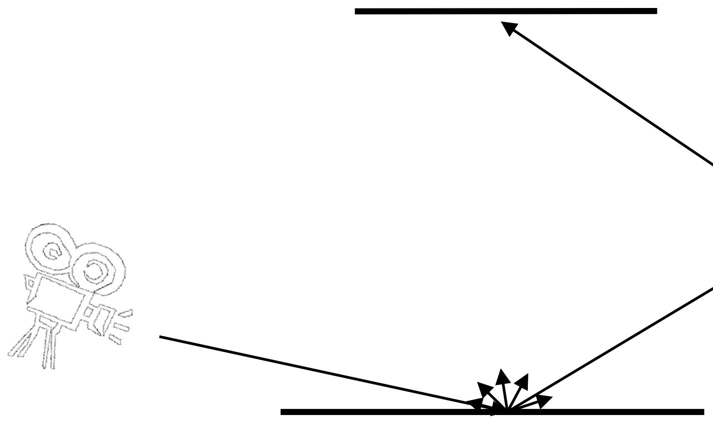


Figure 6. A sample image. All objects are neutral grey. Color on the objects is due to caustics from the green glass balls and color bleeding from the base polygon.

Path tracing sketch

Recursive unwinding of the RE

- To calculate $L(x, \omega_{out})$, we need to calculate $L(\text{ray}(x, \omega_{in}), -\omega_{in})$ for all directions ω_{in} around x.
- At each intersected point, we need to do the same recursively.



Path tracing, v. 0.1

```
estimateLin ($x$, $\omega$):                 // radiance incident at $x$ from direction $\omega$
    $y$ = findNearestIntersection($x$, $\omega$)
    if (no intersection)
        return backgroud.getLe ($y$, $-\omega$) // emitted radiance from envmap
    else
        return   getLe ($y$, $-\omega$) +        // emitted radiance (if $y$ is on a light)
                 estimateLrefl ($y$, $-\omega$)// reflected radiance

estimateLrefl($x$, $\omega_{out}$):
    [$\omega_{in}$ , pdf] = genRandomDir($x$, $\omega_{out}$);          // e.g. BRDF imp. sampling
    return estimateLin($x$, $\omega_{in}$) * brdf(x, $\omega_{in}$, $\omega_{out}$) * dot($\mathbf{n}_x$, $\omega_{in}$) / pdf
```

- Let’s now argue more rigorously why the recursive path tracing actually solves the rendering equation…

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

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

unknown function

known functions

equation "kernel"

Rendering equation:

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

Linear operators

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

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

- 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

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

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

Transport operator

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

- Rendering equation

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

Solution of the RE in the operator form

- Rendering equation

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

- Formal solution

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

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

  - unusable in practice – the inverse cannot be explicitly calculated

Expansion of the rendering equation

- Recursive substitution L

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

- 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

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

- Proof

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

Rendering equation

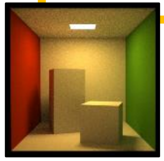
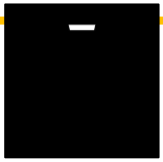
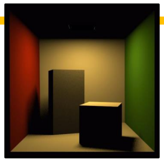
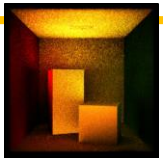
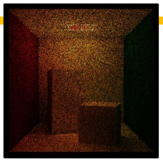
$$\boxed{L = L_\mathrm{e} + T \circ L}$$

$=$

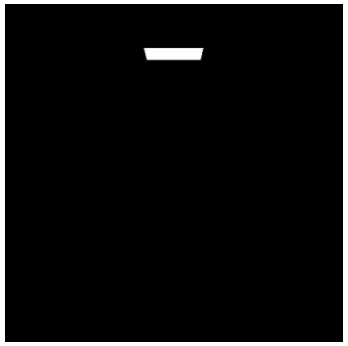
$+ \quad T \circ$

- **Solution**: Neumann series

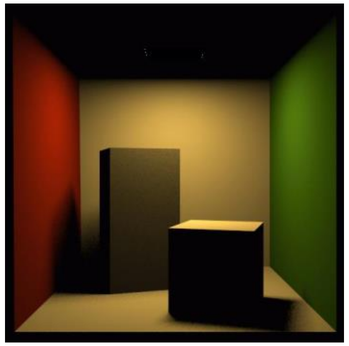
$$\boxed{L = L_\mathrm{e} + T L_\mathrm{e} + T^2 L_\mathrm{e} + T^3 L_\mathrm{e} + \ldots}$$

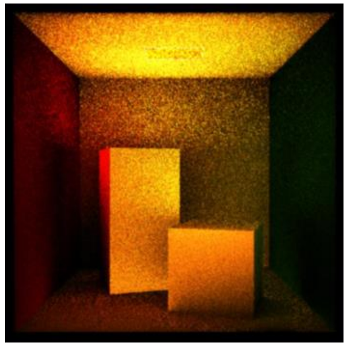
Progressive approximation



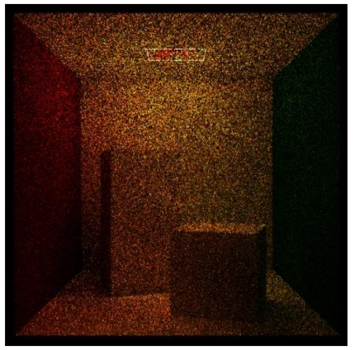
L_e



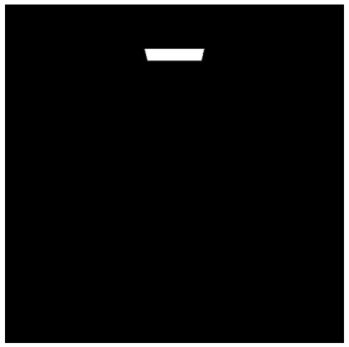
$T \circ L_e$



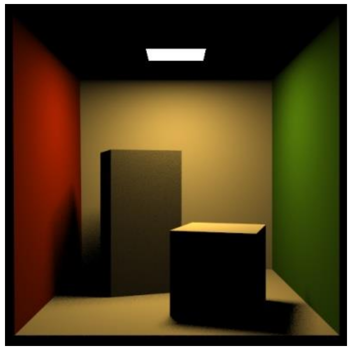
$T \circ T \circ L_e$



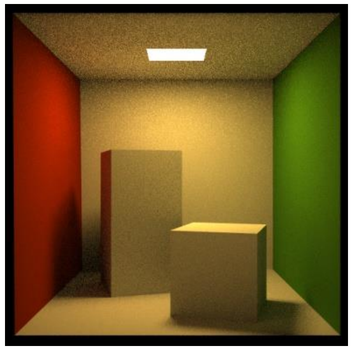
$T \circ T \circ T \circ L_e$



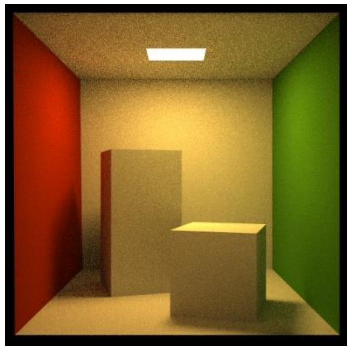
L_e



$L_e + T \circ L_e$



$L_e + TL_e + T^2 L_e$



$L_e + \ldots + T^3 L_e$

Progressive approximation

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

$$L = \underbrace{\boxed{\underbrace{L_\mathrm{e}}_{\text{emission}} + \underbrace{TL_\mathrm{e}}_{\text{direct illumination}}}}_{\text{OpenGL shading}} + \underbrace{T^2L_\mathrm{e}}_{\text{one-bounce indirect illumination}} + \underbrace{T^3L_\mathrm{e}}_{\text{two-bounce indirect illumination}} + \ldots$$

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

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

Solution through the Neumann series

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

- 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

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

- Paths in a high-dimensional path space

$$L = L_\mathrm{e} + TL_\mathrm{e} + T^2L_\mathrm{e} + T^3L_\mathrm{e} + \ldots$$

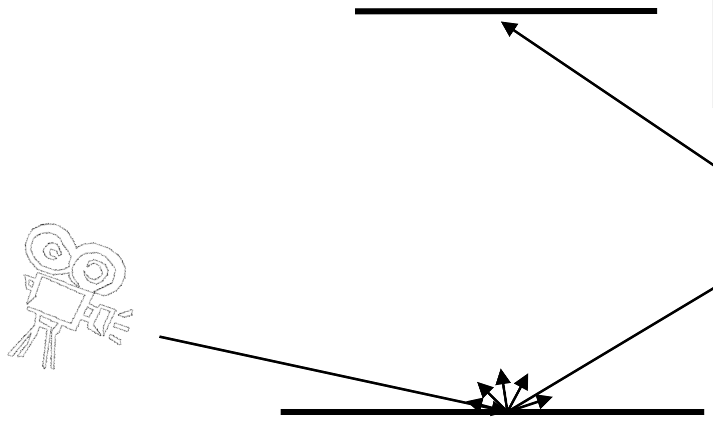
- Recursive solution of a series of nested (hemi)spherical integrals:

$$L = L_\mathrm{e} + T(L_\mathrm{e} + T(L_\mathrm{e} + T(L_\mathrm{e} + \ldots$$

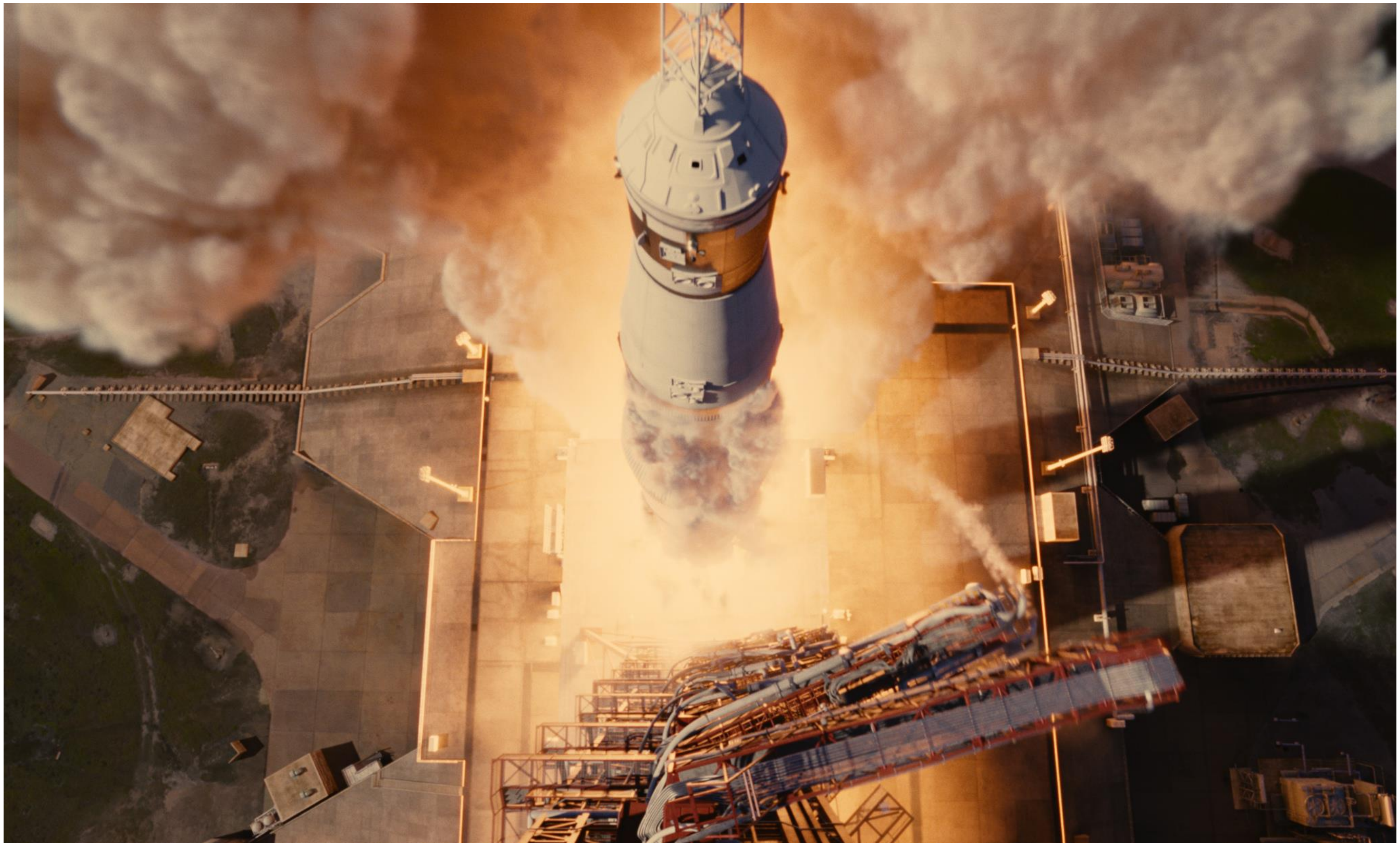
Recursive unwinding of the RE

- To calculate $L(x, \omega_{out})$, we need to calculate $L(\mathrm{ray}(x, \omega_{in}), -\omega_{in})$ for all directions ω_{in} around x.
- At each intersected point, we need to do the same recursively.

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



Path tracing, v. 2012, Arnold Renderer



corona

Angular and Area form of the rendering equation

Angular integral form of the RE

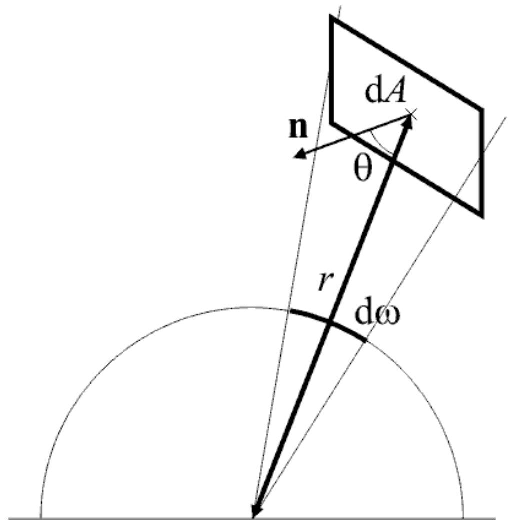
- Integral over the hemisphere in incoming directions

$$L(x,\omega_{\text{out}}) = L_{\text{refl}}(x,\omega_{\text{out}}) + \int_{H(\mathbf{x})} L(\text{ray}(x,\omega_{\text{in}}),-\omega_{\text{in}}) \cdot f_r(x,\omega_{\text{in}} \to \omega_{\text{out}}) \cdot \cos\theta_{\text{in}} \; \mathrm{d}\omega_{\text{in}}$$

Going from angular to area form

- Change-of-variables applied to the angular form

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



Area integral form of the RE

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

$$L(x,\omega_{\text{out}}) = L_{\text{e}}(x,\omega_{\text{out}}) + \int_M L(y \to x) \cdot f_r(y \to x \to \omega_{\text{out}}) \cdot G(x \leftrightarrow y) \cdot V(x \leftrightarrow y)\, \mathrm{d}A_y$$

scene surface

geometry term

$$G(x \leftrightarrow y) = \frac{\cos\theta_x \cdot \cos\theta_y}{\|x-y\|^2}$$

visibility

1 … y visible from x

0 … otherwise

Angular integral form

- Sum up radiance contributions from all directions
- For each direction, find the nearest surface by ray tracing
- 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.

Area integral form

- Sum up contributions from all other points on the scene surface
  - Contribution added only if visible
- Implementation in stochastic path tracing:
  - Generate randomly point y on scene geometry (e.g. on a light source). 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

(Optional material)

From the RE to radiosity

- Start from the area integral form of the RE
- The Radiosity method– **assumptions**
  - Only diffuse surfaces (BRDF constant in ω_{in} and ω_{out})
  - Radiosity (i.e. radiant exitance) is spatially constant (flat) over the individual elements

From the RE to radiosity

- **Diffuse surfaces only**
  - The BRDF is constant in ω_{in} and ω_{out}

$$L(x,\ \omega_{\text{out}}) = L_{\text{e}}(x,\ \omega_{\text{out}}) + \frac{\rho(x)}{\pi} \int_{M} L(y \to x) \cdot G(x \leftrightarrow y) \cdot V(x \leftrightarrow y)\ \mathrm{d}A_y$$

  - **Outgoing radiance is independent of** ω_{out} . It is equal to radiosity *B* divided by π

$$B(x) = B_{\text{e}}(x) + \rho(x) \cdot \int_{M} B(y) \cdot \underbrace{\frac{G(x \leftrightarrow y) \cdot V(x \leftrightarrow y)}{\pi}}_{G'(x \leftrightarrow y)}\ \mathrm{d}A_y$$

From the RE to radiosity

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

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

Radiosity of the *j*-th element

Geometry factor between surface element *j* and point *x*

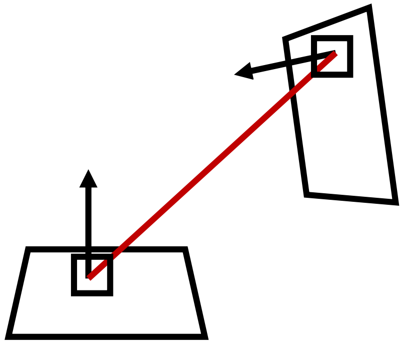
From the RE to radiosity

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

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

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

F_{ij} … form factor



Classic radiosity equation

- System of linear equations

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

- Form factors

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

- **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 anymore**
  - Scene subdivision -> sensitive to the quality of the geometry model (but in reality, models are always broken)
  - High memory consumption, complex implementation