Computer Graphics 2013

14. Global Illumination

Hongxin Zhang
State Key Lab of CAD&CG, Zhejiang University

2013-10-30

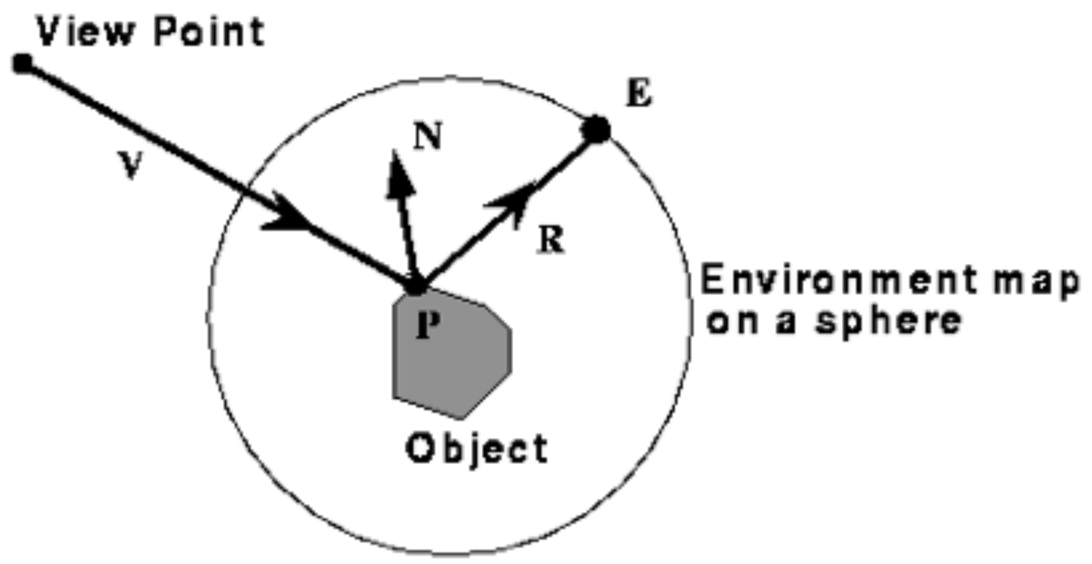
Final examination

- Friday night, 7:30PM ~ 9:00PM, Nov. 8th
- Room 103 (?), CaoGuangBiao West Building
- Bring your textbook and course notes

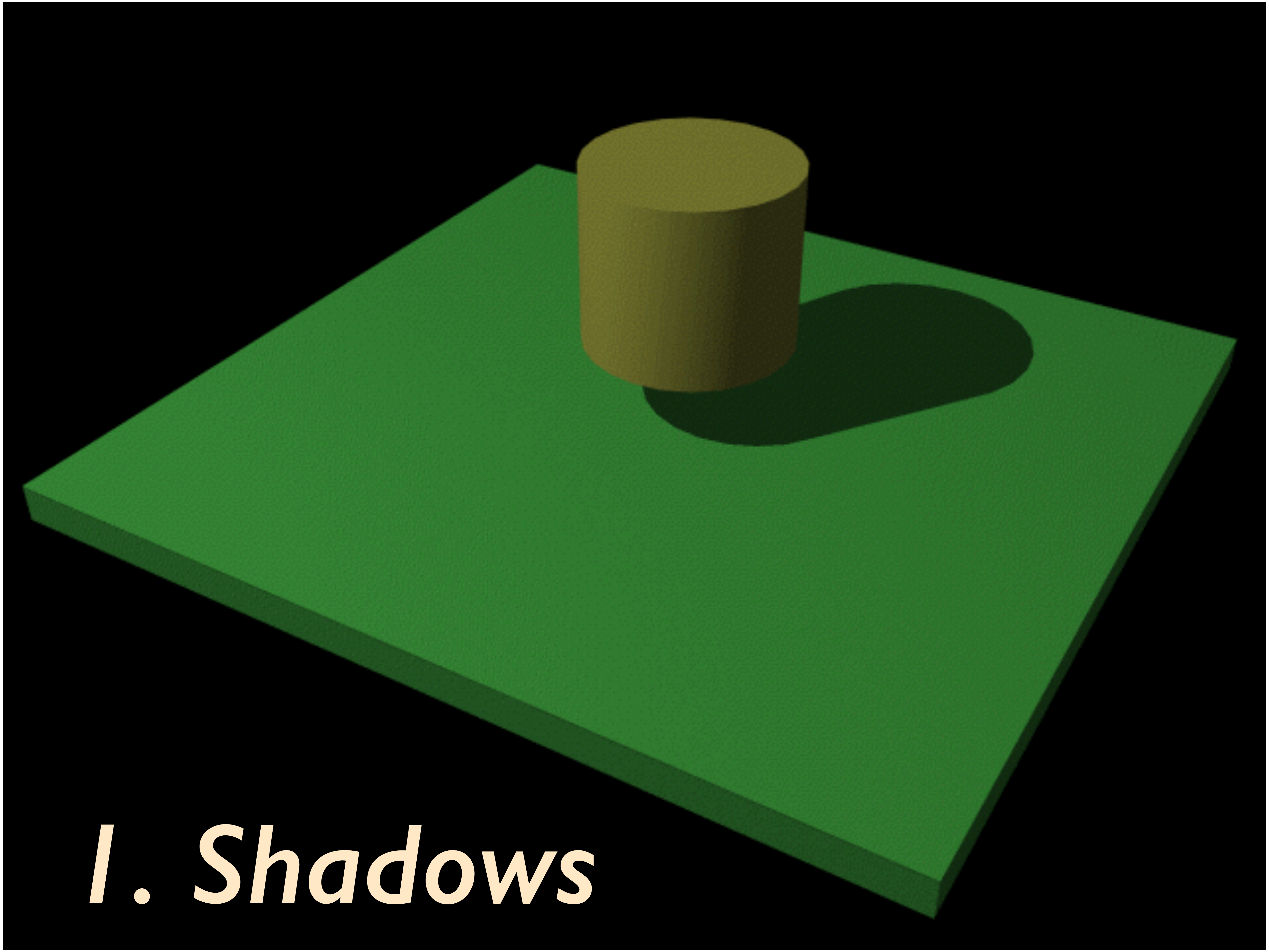
Outline

- Shadows
- Radiosity
- Ray-tracing

Environment Maps

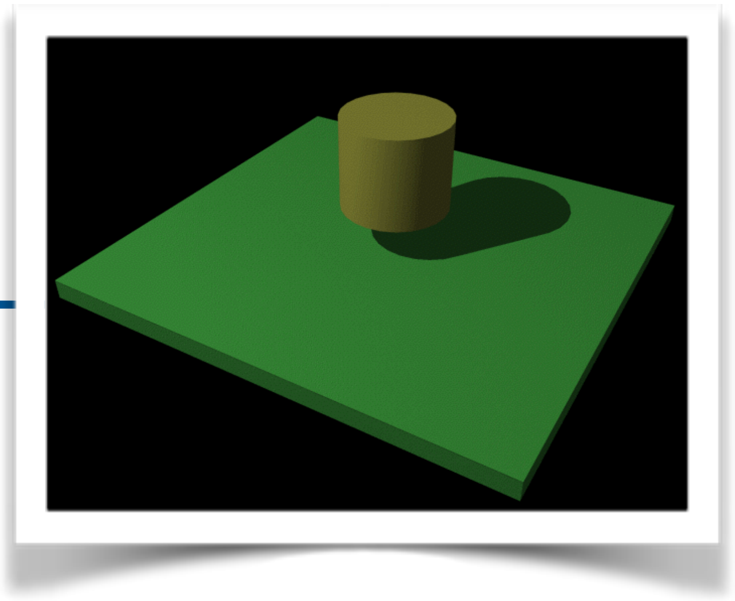


- We use the *direction* of the reflected ray to index a texture map.
- We can simulate reflections. This approach is not completely accurate. It assumes that all reflected rays begin from the same point, and that all objects in the scene are the same distance from that point.



1. Shadows

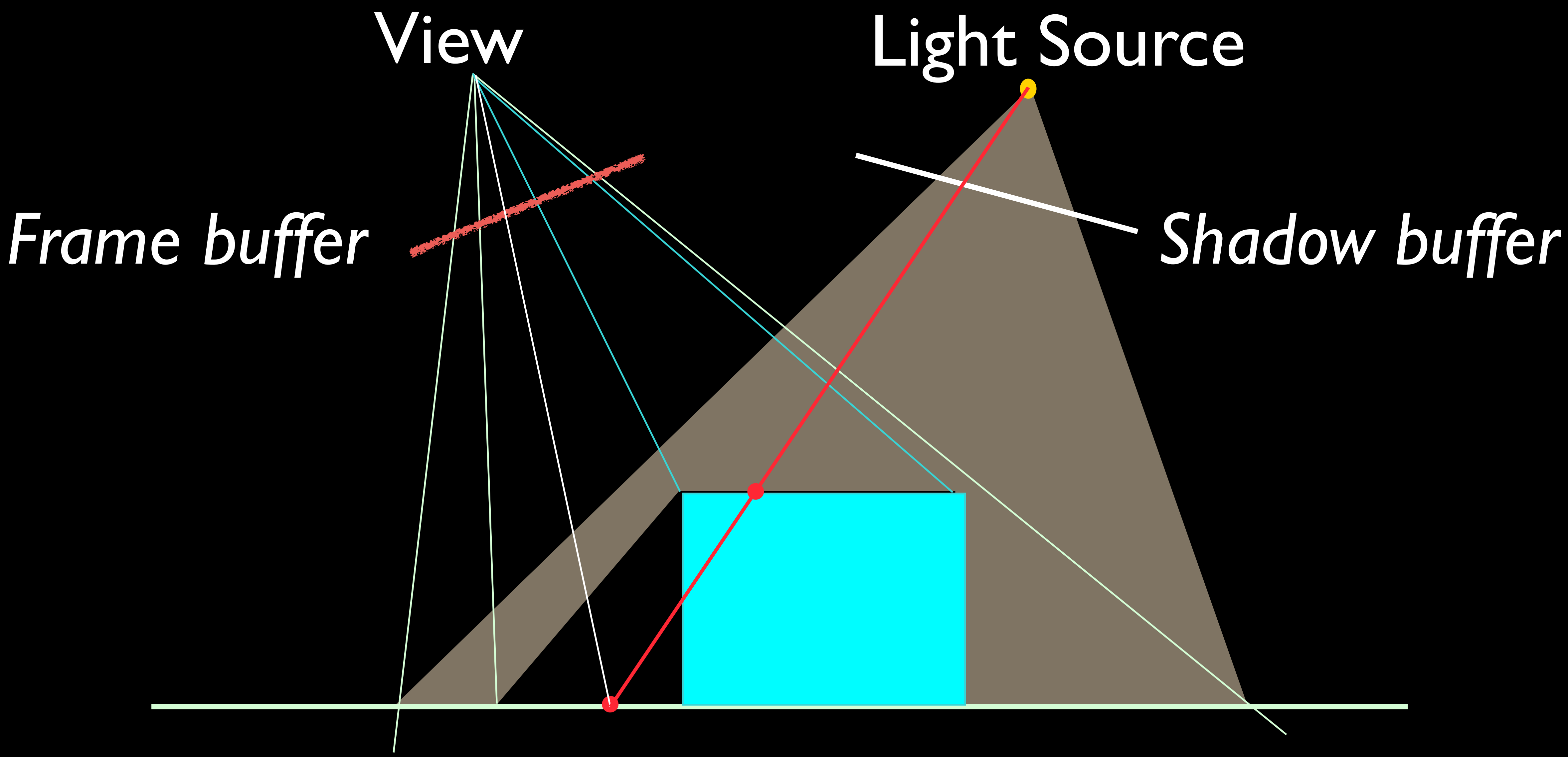
Shadows



- eliminate the perceptual artifacts of objects floating above the ground
- emphasize the changing direction of the light source
- may be *sharp edged* or *soft edge*, can contain both an *umbra* and a *penumbra* depending on the shape of the light source and its distance from the light source

- An illuminated scene without shadows could be confusing and affects realism

Shadow buffer



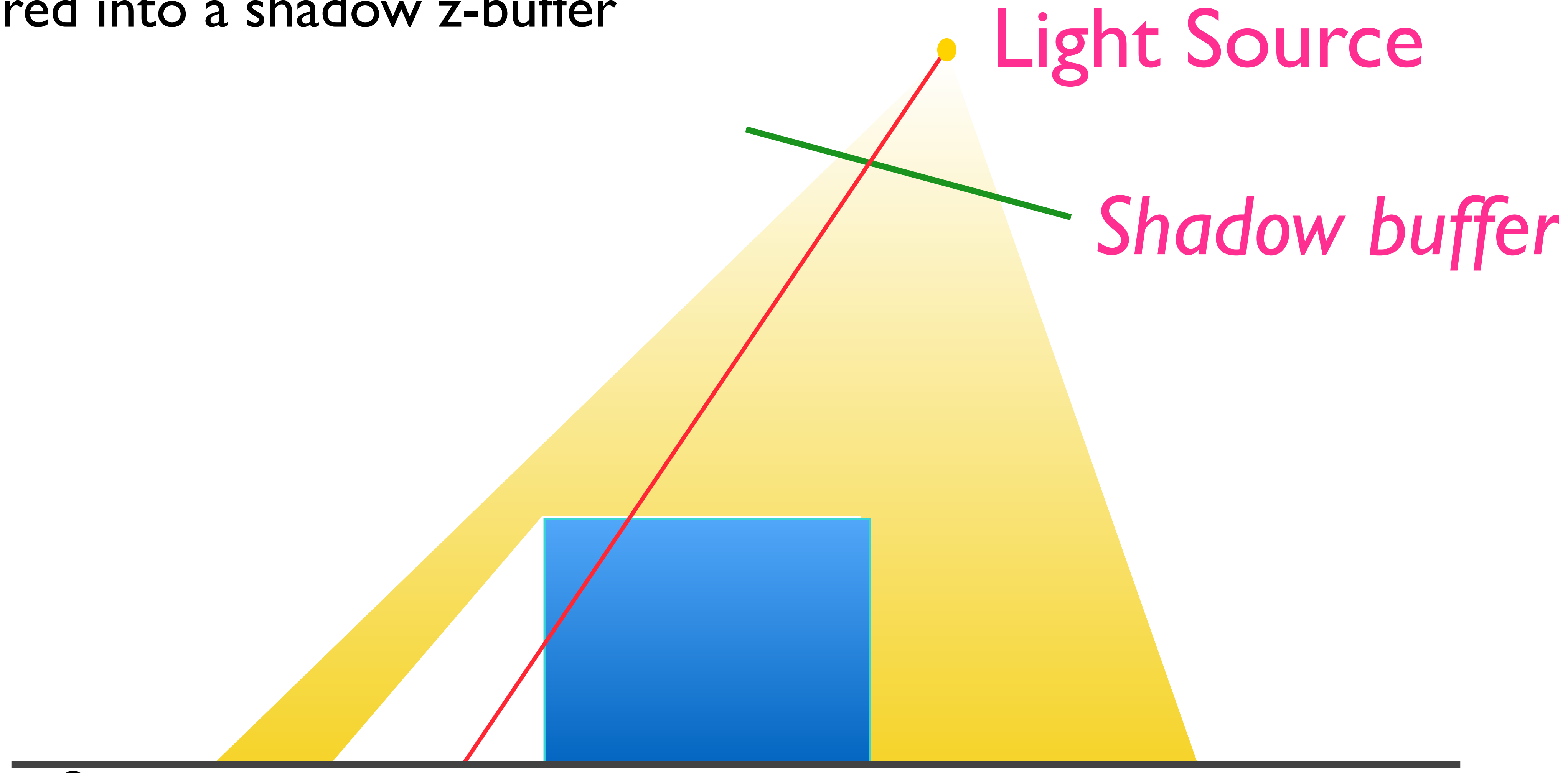
$$I = I_a k_a + \sum_{1 \le i \le m} I_{pi}[k_d(\overline{N} \bullet \overline{L}_i) + k_s(\overline{R}_i \bullet \overline{V})^n]$$

Shadow Z-buffer

- One of the simplest approaches to shadow computation
  - Integrates easily with z-buffer based renderers
- Idea:
  shadows are those parts of objects that are not visible when viewed from the light source

Shadow Z-buffer

- The algorithm has two steps
  - **First step**
    the light source is treated as the eye point and the appropriate view transformation is applied
  - The scene is rendered using z-buffer algorithm, but only depth information is stored into a shadow z-buffer

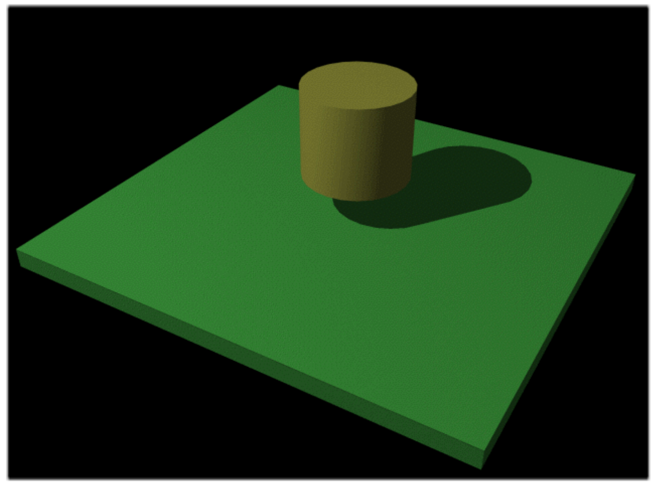


Shadow Z-buffer (Z-buffer enhanced for shadows)

- **Second step**
  the scene is rendered from the actual view with an enhanced z-buffer algorithm
  - if a point (x, y, z) is visible, than the view transformation of the light source is applied to the point which maps it to a point (xl, yl, zl)
  - (xl, yl, zl) is a view of the point (x, y, z) seen from the source
  - (xl, yl) is used to index into the shadow z-buffer and the corresponding depth value is compared with zl
  - if zl is greater than the depth value then the (x, y, z) is in a shadow

Facts about adding shadows

- Algorithms for shadows deal with polygonal meshes
  - ray tracing is an exception
- Most of the shadows generated are hard edged generated by point light sources
  - Shadows generated by area light sources require special handling
- Adding shadows is a computational overhead and hence is not treated as a necessity like shading algorithms
- For 3D animation shadows are important for depth and movement perception



Global Illumination

- Observation:
  light comes from other surfaces, not just designated light sources
- Goal:
  simulate inter-reflection of light in 3D scenes
- Difficulty:
  you can no longer shade surfaces one at a time, since they're now interrelated!

- Two general classes of algorithms:
  - **radiosity methods**: set up a system of linear equations whose solution is the light distribution
  - **ray tracing methods**: simulate motion of photons one by one, tracing photon paths either backwards or forwards

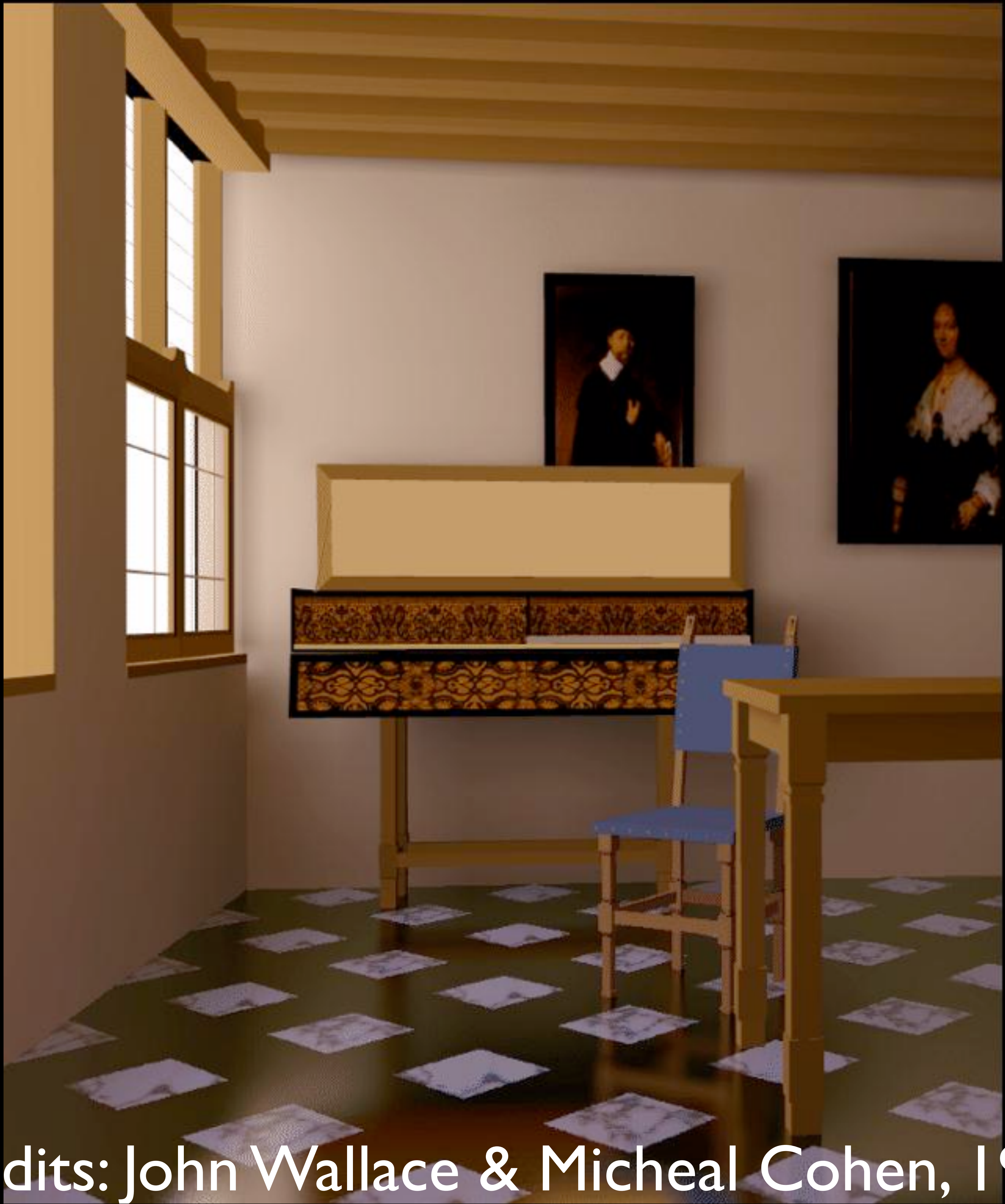
2. *Radiosity*

credits: Stu Feldman & John Wallace, 1987

Radiosity Methods

- Solutions for global diffused interactions
- Object space algorithm
  - solves for intensity on the surface of each object in the environment
  - **view independent.**

- The resulting intensity solution is given to a renderer
  - synthesizes an image for a specific view by removing hidden surfaces
- Excellent for generation of realistic images of interior environments which are collections of non-specular objects



credits: John Wallace & Micheal Cohen, 1987

credits: Dani Lischinski & Filippo Tampieri, 1993

Radiosity Theory

- Radiosity is defined as energy per unit area leaving a surface per second

- Surfaces in the environment are divided into smaller elements called patches

- Each patch has surface properties like reflectivity **R** (positive value less than 1.0) , and emissivity **E**, which is the energy emitted per unit area
- Energy leaving a patch is the sum of reflected energy and emitted energy

- For the $\mathbf{i}^{th}$ patch with *radiosity* $\mathbf{B_i}$ and *area* $\mathbf{A_i}$
- energy leaving the patch is $\mathbf{B_i A_i}$

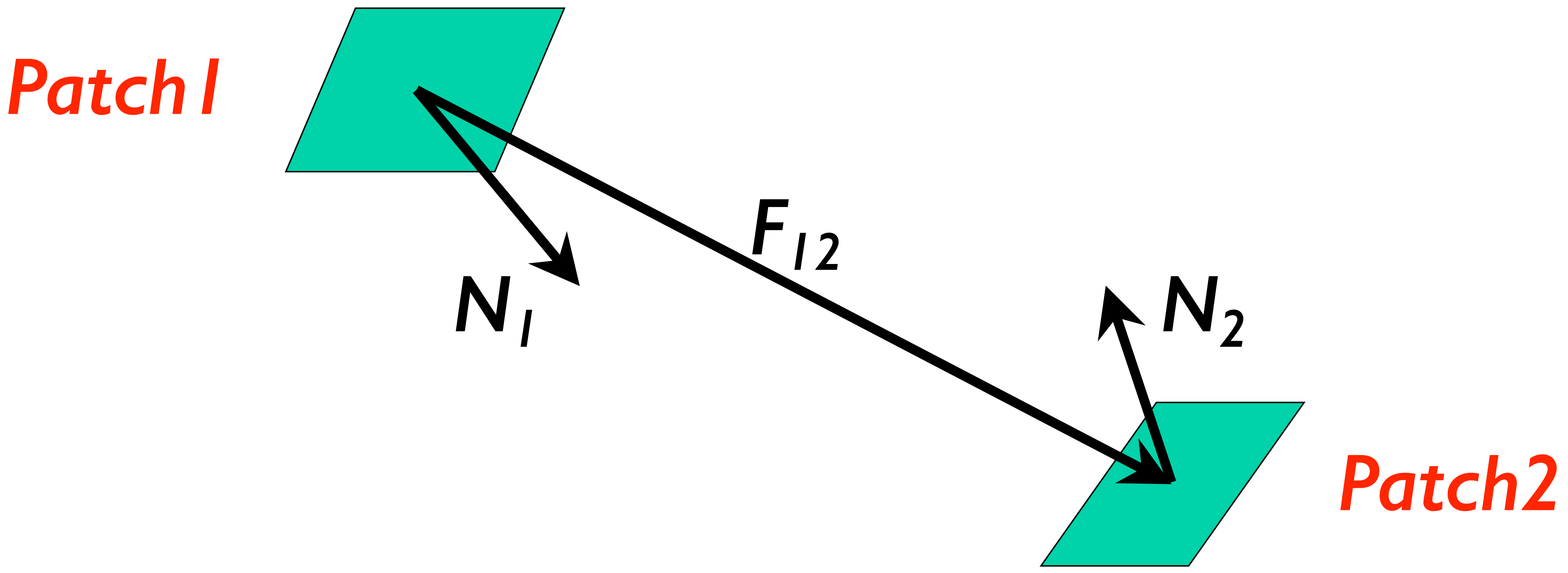
B_iA_i = ***emitted energy*** + ***reflected energy***

emitted energy = E_iA_i

reflected energy = R_i x (***Incident energy***)

- *Incident energy* is the energy that arrives at the i^{th} patch from all the other patches in the environment
- If the $\mathbf{B_j A_j}$ is the energy leaving the $\mathbf{j}^{th}$ patch, then the amount of that energy reaching the $\mathbf{i}^{th}$ patch is denoted as $\mathbf{F_{ji} B_j A_j}$
- $\mathbf{F_{ji}}$ is called the **form factor**

Form Factor Calculation



Form Factor Calculation

- The radiosity equation can now be written as

$$B_i A_i = E_i A_i + R_i \sum_{j=1}^{n} B_j F_{ji} A_j$$

- Form factor $\mathbf{F_{ji}}$ is a constant and depends on the geometric relationship between the patches
- A reciprocity relationship gives $\mathbf{F_{ij}A_i=F_{ji}A_j}$
- Substituting this relationship in the above radiosity equation and dividing throughout by $\mathbf{A_i}$

$$B_i = E_i + R_i \sum_{j=1}^{n} B_j F_{ij}$$

The Radiosity Matrix

$$B_i - R_i \sum_{j=1}^{n} B_j F_{ij} = E_i$$

Such an equation exists for every patch in the scene

$$B_1 - R_1 F_{11} B_1 - R_1 F_{12} B_2 - R_1 F_{13} B_3 \cdots = E_1$$
$$B_2 - R_2 F_{21} B_1 - R_2 F_{22} B_2 - R_2 F_{23} B_3 \cdots = E_2$$
$$B_3 - R_3 F_{31} B_1 - R_3 F_{32} B_2 - R_3 F_{33} B_3 \cdots = E_3$$
$$\vdots$$
$$B_n - R_n F_{n1} B_1 - R_n F_{n2} B_2 - R_n F_{n3} B_3 \cdots = E_n$$

The Radiosity Matrix

This set simultaneous of equations can be written in matrix form:

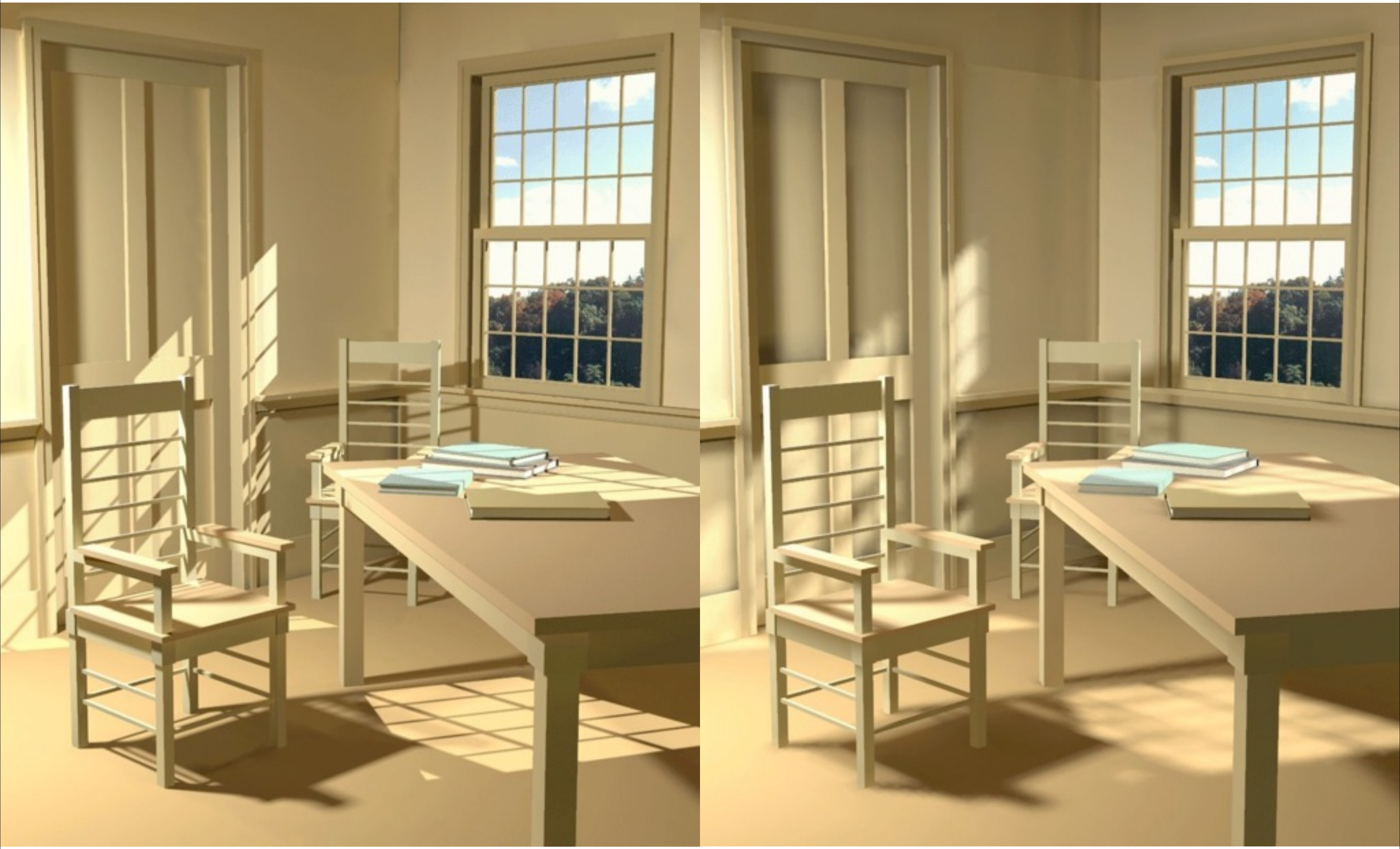
$$\begin{bmatrix} 1 - R_1 F_{11} & -R_1 F_{12} & \mathrm{L} & -R_1 F_{1n} \\ -R_2 F_{21} & 1 - R_2 F_{22} & \mathrm{L} & -R_2 F_{2n} \\ \mathrm{M} & \mathrm{M} & & \mathrm{M} \\ -R_n F_{n1} & -R_n F_{n2} & \mathrm{L} & 1 - R_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \mathrm{M} \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \mathrm{M} \\ E_n \end{bmatrix}$$

E_i is non-zero only for patches of light sources

R_i is the reflectivity of the patch

F_{ij} is the form factor which can be calculated

The matrix equation is solved to calculate B_i's



credits: Dani Lischinski & Filippo Tampieri, 1993

Reference

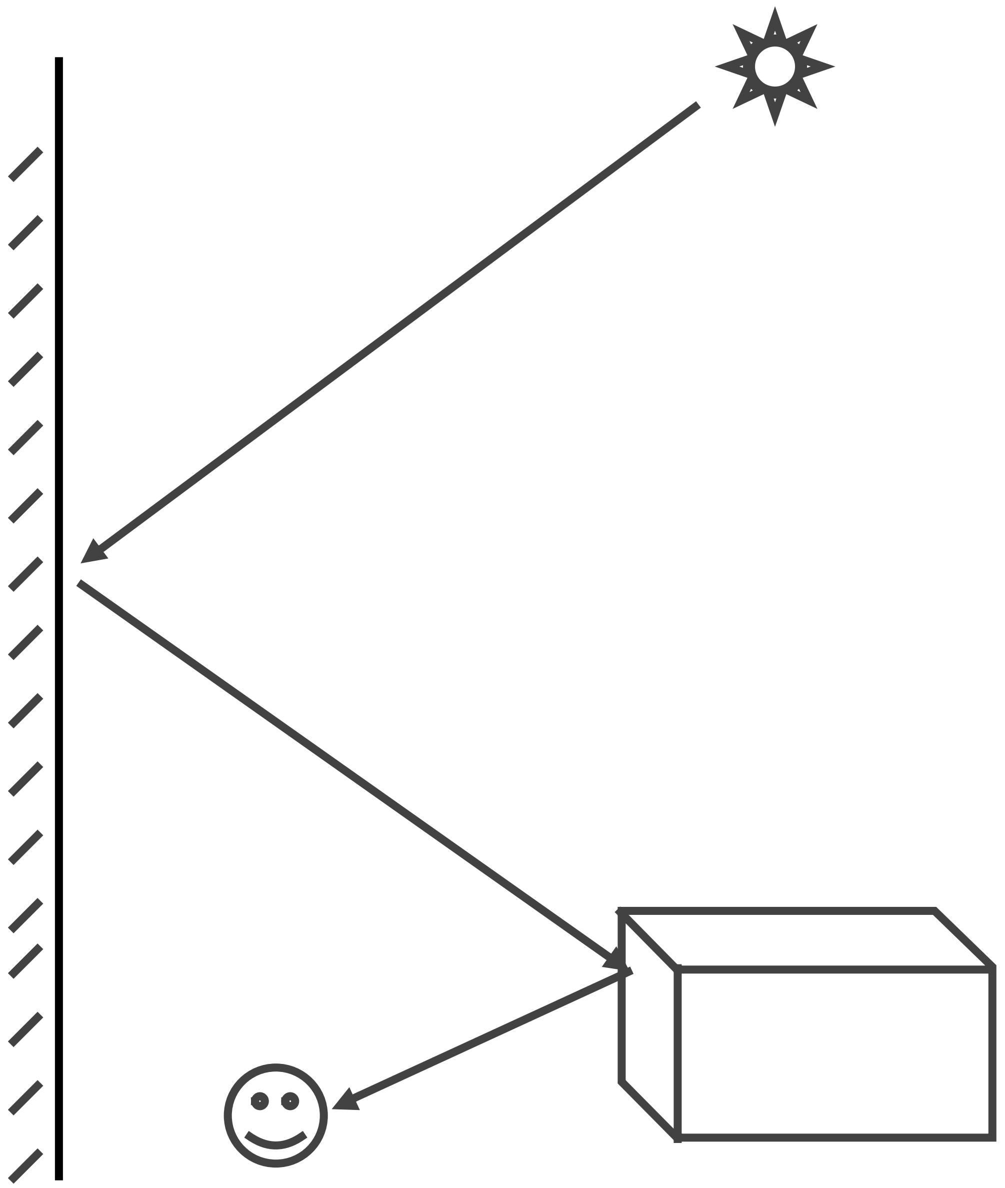
- http://dudka.cz/rrv

DIAMANTS - DENIS OLIVIER 6/95 © ChromaGraphics - POV-Ray 2.2 - POVLAB 2.0

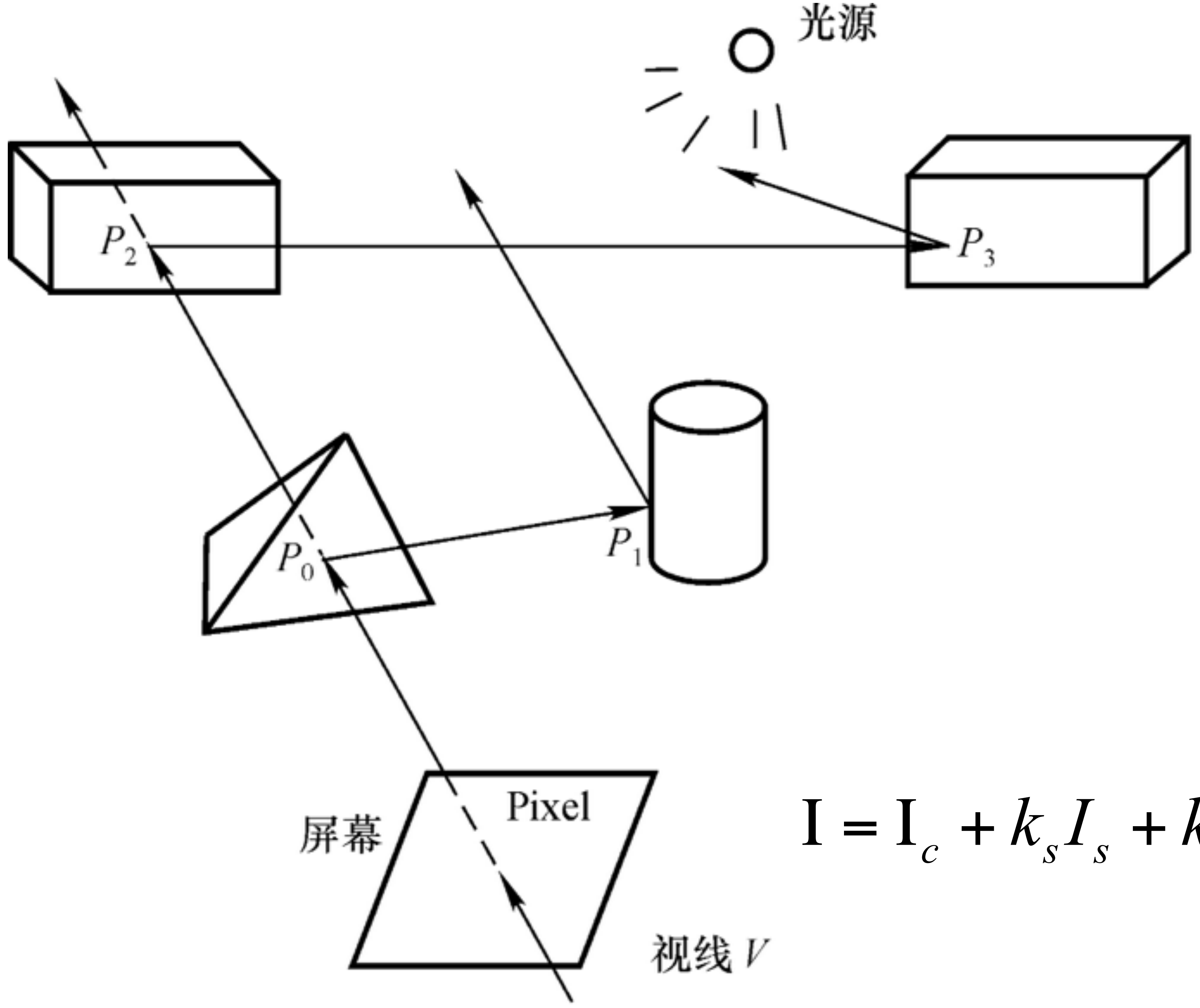
3. *Ray Tracing*

Introduction

- Ray Tracing is a technique for image synthesis
  Helps create a 2D picture of a 3D world
- An algorithm for visible surface determination, which combines following factors in a single model
  - hidden surface removal
  - shading due to direct illumination
  - shading due to global illumination
  - shadows



Witted model: $\mathrm{I} = \mathrm{I}_c + k_s I_s + k_t I_t$

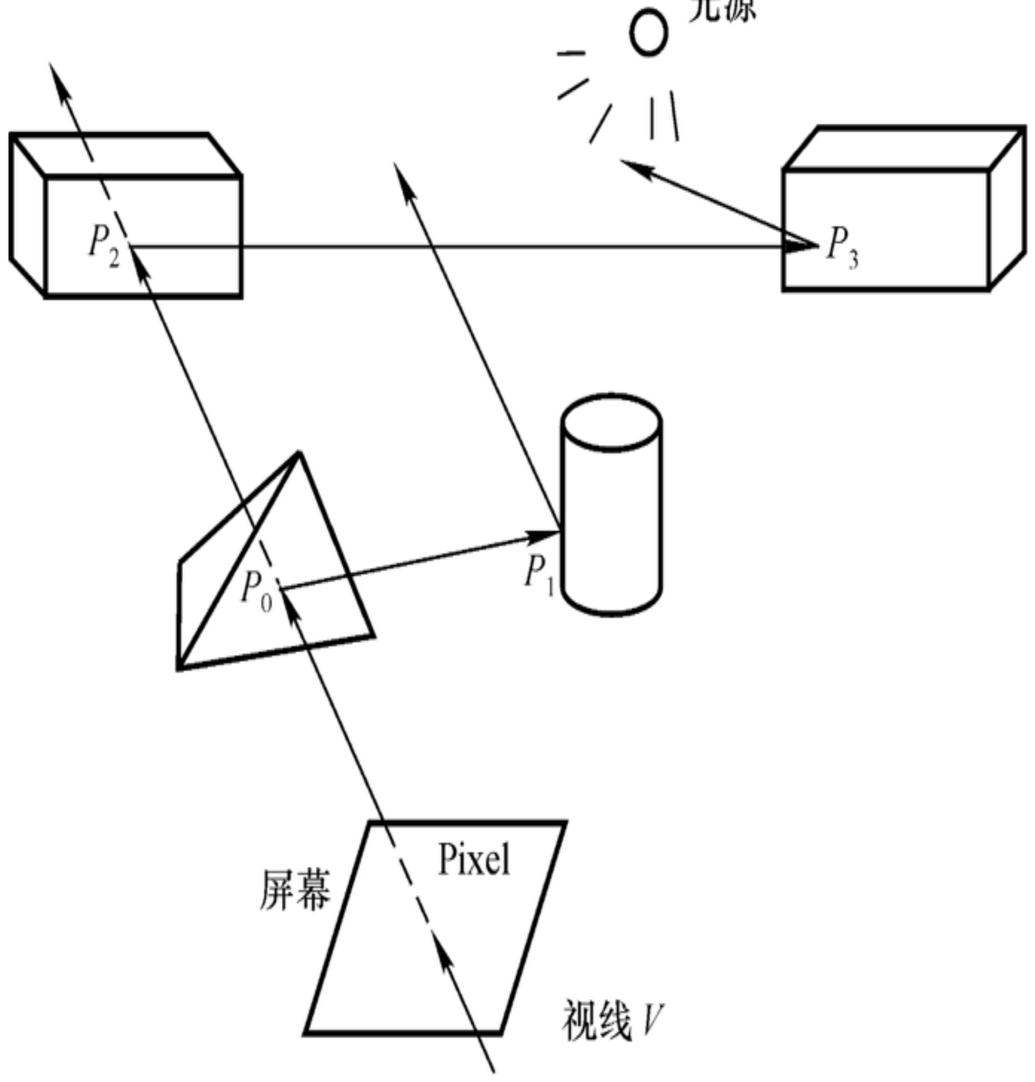


$$\mathrm{I} = \mathrm{I}_c + k_s I_s + k_t I_t$$

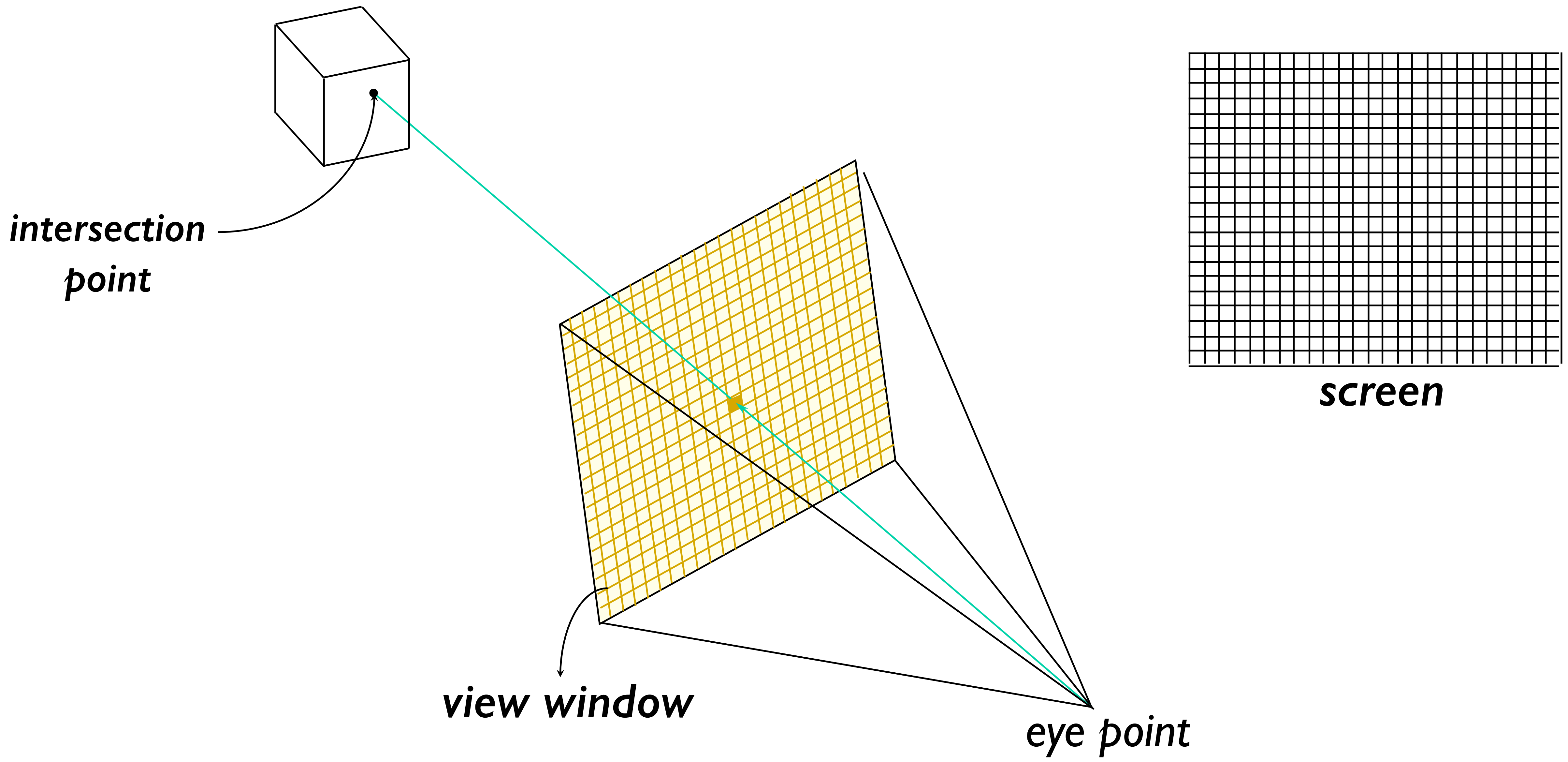
```
void a_raytrace(Vector3D ray, int level, Color *I)
{
    Color   Ilocal, Is, It;
    Int Is_inter;
   Vector3D vl, vr, p, Normal;

    Inter_scene(ray, scene, &Is_inter, &p, &Normal, &vl, &vr, face);
    if (Is_inter) {
        Calculate_Local_I(&Ilocal, face, p, Normal);
        if (Level<plvl) {
            a_raytrace(vl, int level+1, &Is);
            a_raytrace(vr, int level+1, &It);
            *I=Ilocal+face->ks*Is + face->kt*It;
        }
        else
            *I=Ilocal;
    }
    else
      *I = Background;
 }
```

$$\mathrm{I} = \mathrm{I}_c + k_s I_s + k_t I_t$$

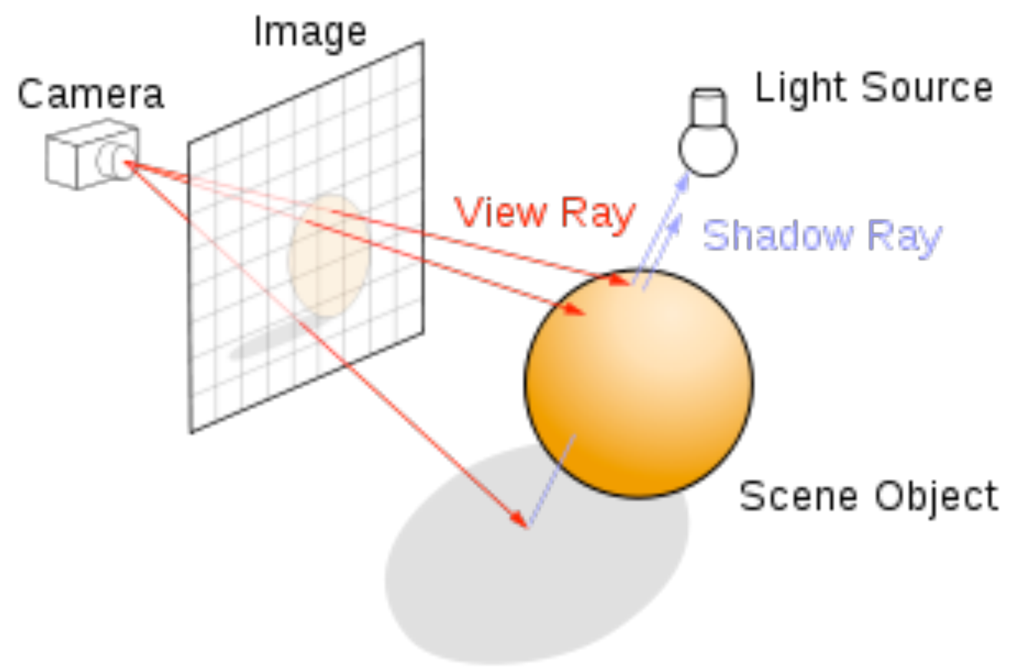


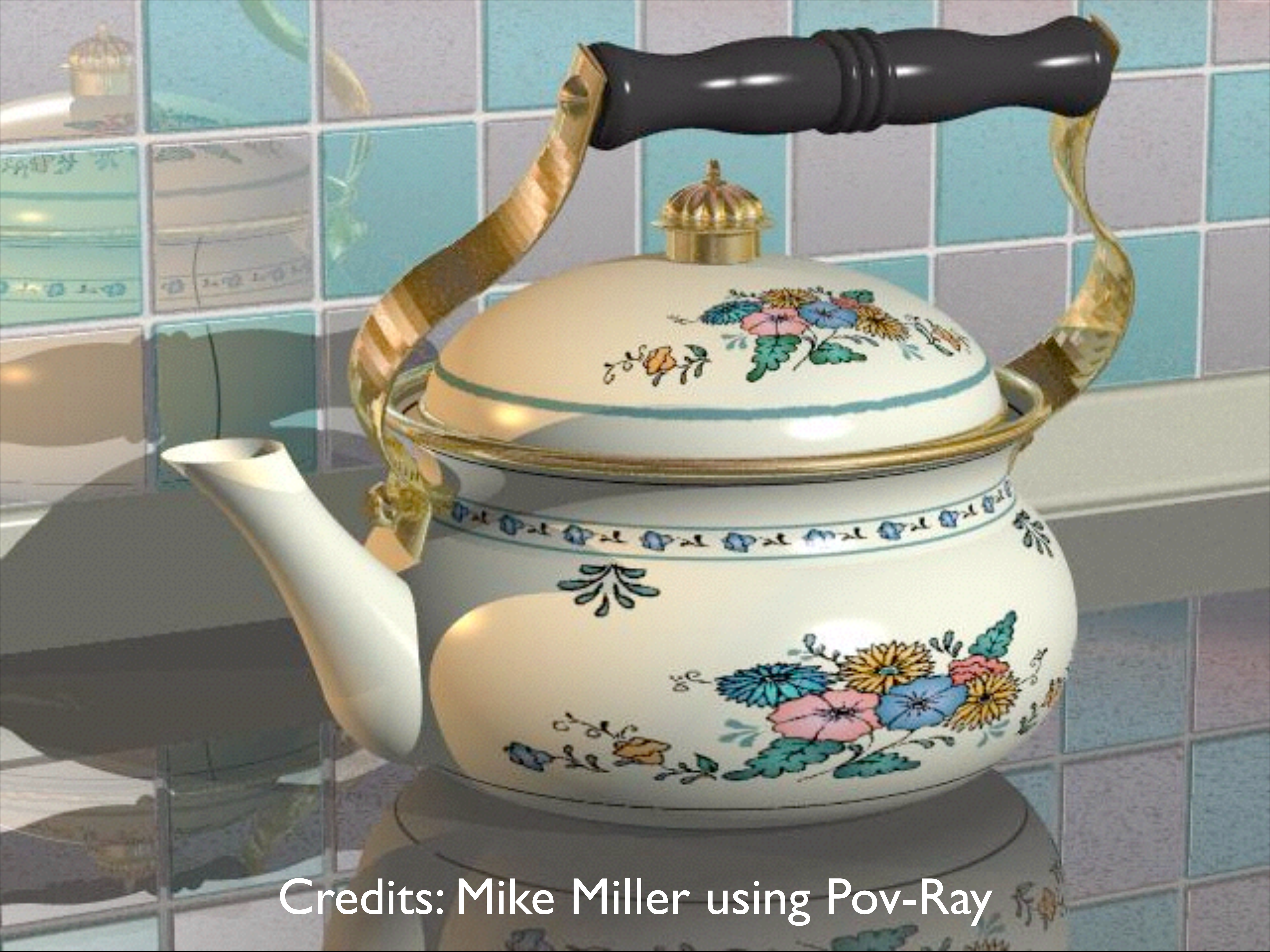
Ray Tracing



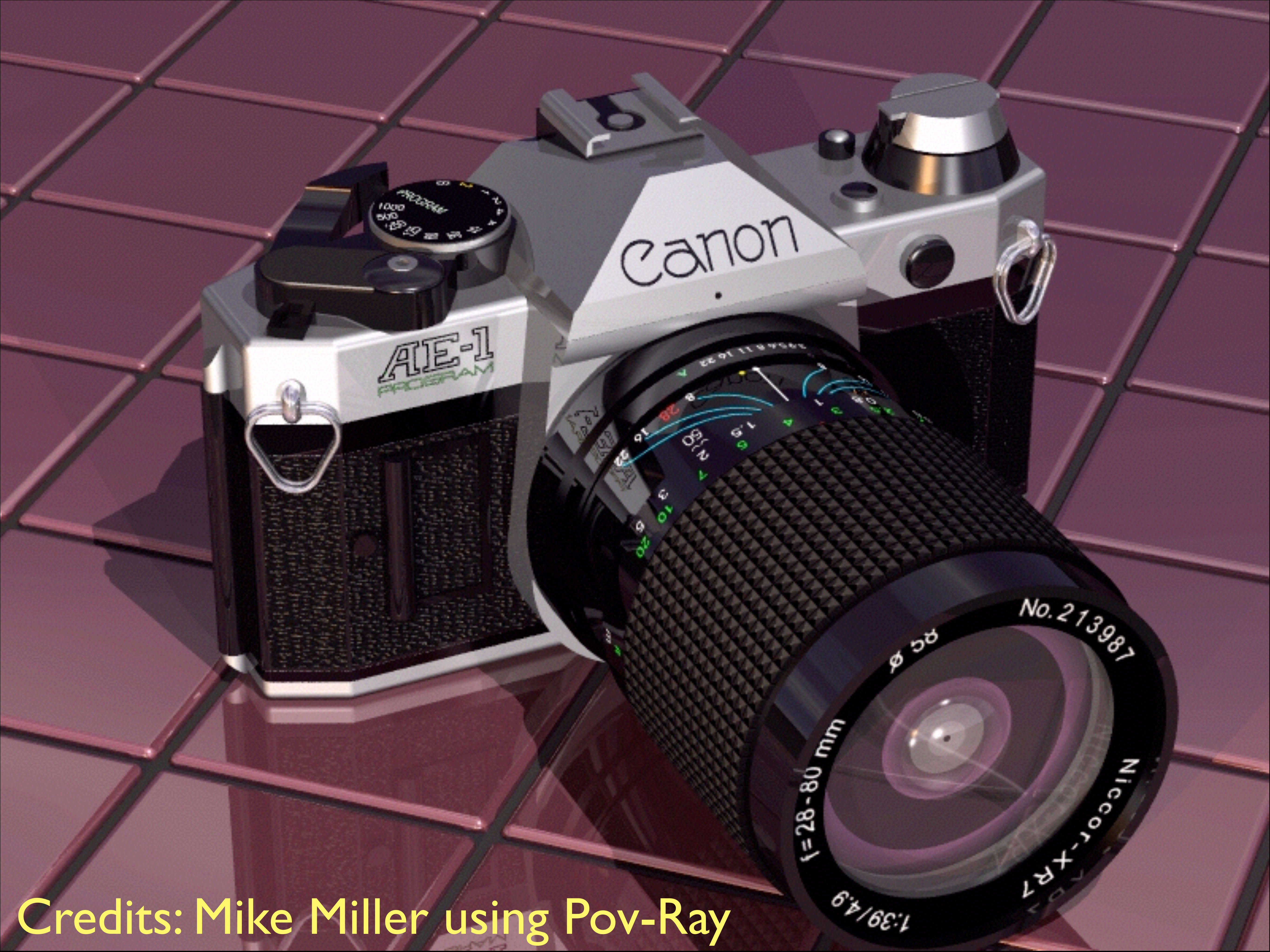
Features

- Best known for handling shadows, reflections and refraction
- It is an algorithm that works entirely in object space, hence accurate
- Partial solution to global illumination problem and is the most complete simulation of an illumination-reflection model in computer graphics
- Ray tracing has produced some of the most realistic images in computer graphics





Credits: Mike Miller using Pov-Ray



Credits: Mike Miller using Pov-Ray

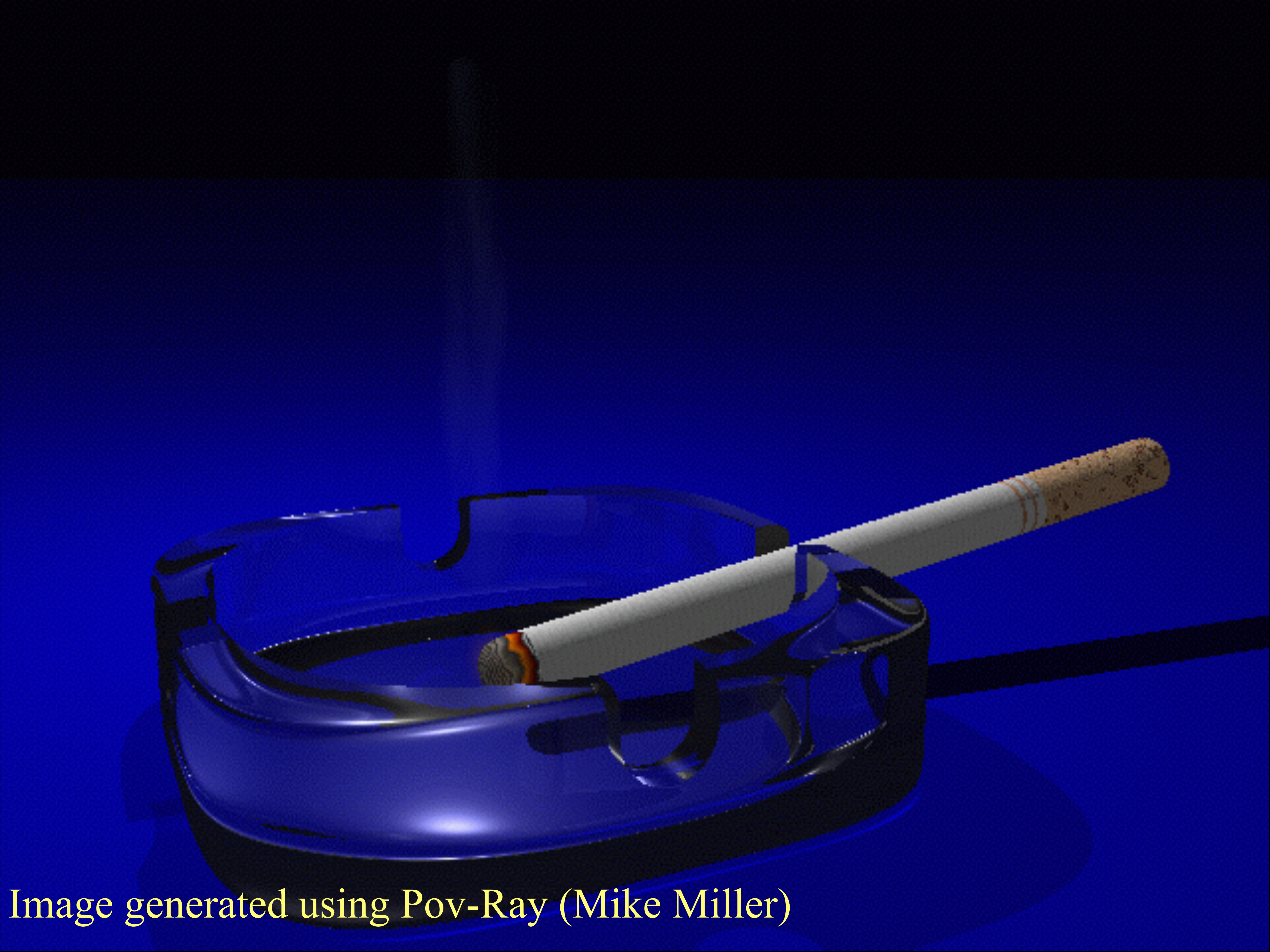


Image generated using Pov-Ray (Mike Miller)

Representing a Ray

- Ray tracing is based on ray-object intersection algorithms

Representing a ray becomes essential:

A point *P* on a ray is given by the parametric equation

$$P = O + t \bullet D \quad , \quad for \quad t > 0$$

where *O* is the ray origin, *D* is the ray direction

If the direction *D* is normalized then *t* is the distance of the point from the origin

…Representing a Ray

- Given a ray with

  origin **O(x_o, y_o, z_o)** and *direction* **D(x_d, y_d, z_d)**

  any point on the ray is given as

$$P\left(x_o + t \bullet x_d, \quad y_o + t \bullet y_d, \quad z_o + t \bullet z_d\right)$$

- This equation forms the basis of calculating intersections with some of the common primitives like sphere, plane etc..

Ray-Sphere Intersection

- Sphere Representation:
  - center **C(x_c, y_c, z_c)** , radius **r**
- Equation of the sphere is

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2$$

- Substituting the ray equation into the sphere equation we have

$$(x_o + t \bullet x_d - x_c)^2 + (y_o + t \bullet y_d - y_c)^2 + (z_o + t \bullet z_d - z_c)^2 = r^2$$

- This is a quadratic equation of the form

$$
\begin{aligned}
&A \bullet t^2 + B \bullet t + C = 0 \\
&\textit{where,} \\
&A = x_d{}^2 + y_d{}^2 + z_d{}^2 = 1 \\
&B = 2 \bullet \left(x_d \bullet \left(x_o - x_c\right) + y_d \bullet \left(y_o - y_c\right) + z_d \bullet \left(z_o - z_c\right)\right) \\
&C = \left(x_o - x_c\right)^2 + \left(y_o - y_c\right)^2 + \left(z_o - z_c\right)^2 - r^2
\end{aligned}
$$

- the two roots are given by

$$
t_1 = \frac{-B - \sqrt{B^2 - 4 \bullet C}}{2} \qquad t_2 = \frac{-B + \sqrt{B^2 - 4 \bullet C}}{2}
$$

- The smallest positive *t* value gives the nearest point of intersection

Ray-Plane Intersection

- The plane is represented by the equation

$$a \bullet x + b \bullet y + c \bullet z + d = 0$$

- Substituting the ray equation into the plane equation we have

$$a \bullet (x_o + t \bullet x_d) + b \bullet (y_o + t \bullet y_d) + c \bullet (z_o + t \bullet z_d) + d = 0$$

- Solving for t

$$t = \frac{-(a \bullet x_o + b \bullet y_o + c \bullet z_o + d)}{(a \bullet x_d + b \bullet y_d + c \bullet z_d)}$$

Ray-Polygon Intersection

- Involves two steps
  - Find the point of intersection of the ray with the plane of the polygon
  - Check if the point is inside or outside the polygon (even-odd rule)

Efficiency in Ray Tracing

- 95% of the time is spent in ray-object intersection
- So to increase speed
  - write faster intersection algorithms
  - reduce number of intersection calculations
- Intersection algorithms are always written to work efficiently. Reducing the number of intersection calculation is the key to increase speeds

Some Observations of Ray tracing

- computationally intensive
  - may take hours to generate a scene of reasonable complexity
- view dependent
  - For every change in view the image has to be recomputed

- Ray tracing in real-time is a challenge even today
  - GPU based ~ or Cloud based ~
  - Use of parallel machines and dedicated ray tracing chips are some methods being investigated to do real-time ray tracing

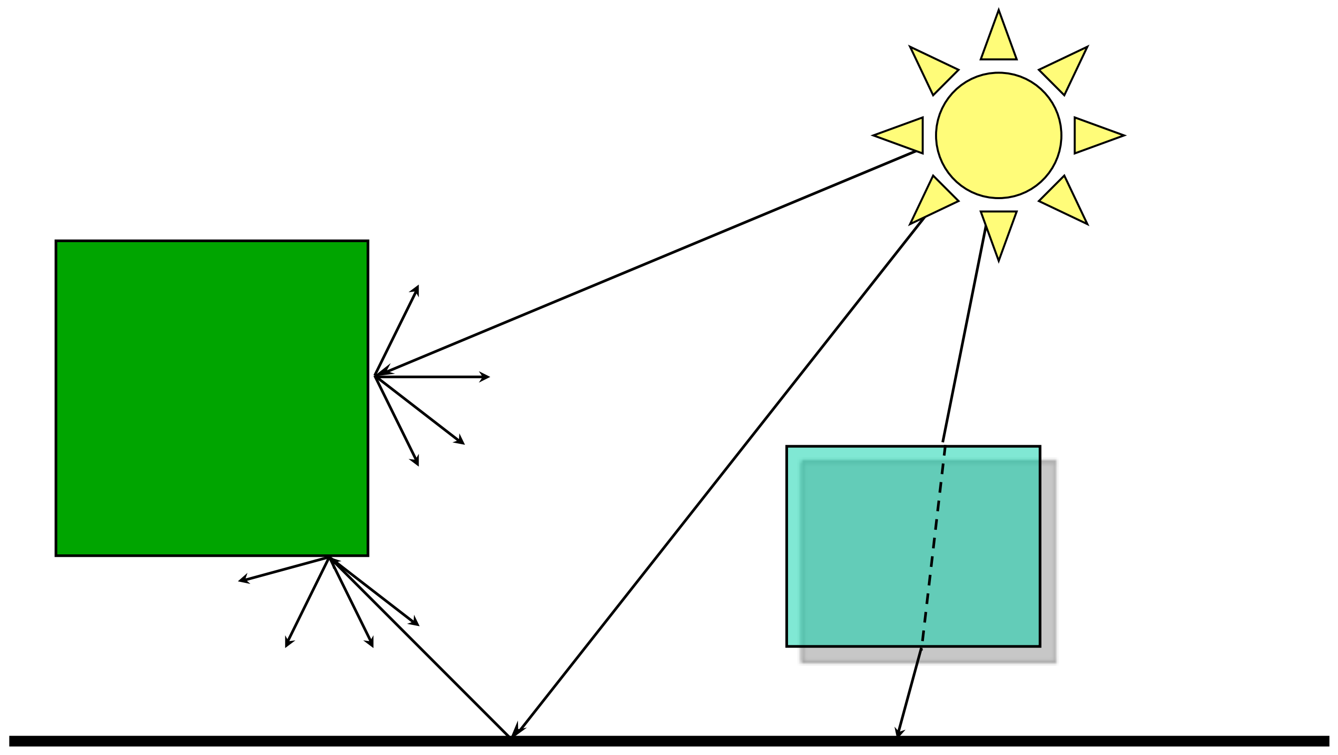
- Ray tracing does not handle in a natural way some behavior of light like
  - diffused inter-reflections, bleeding of colored light from a dull red file cabinet on to a white carpet, giving the carpet a pink tint
  - caustics, focussed light like the shimmering waves at the bottom of the swimming pool

Radiosity v.s. Ray Tracing

- Area light sources	- Point light sources
- Diffuse Reflections	- Specular reflections
- Color Bleeding	- Refraction effects
- Soft Shadows	- Sharp shadows
- View-independent	- View-dependent

Particle/Path Tracing

- Global Illumination Method
- Particle Model of Light
- Monte Carlo Simulation

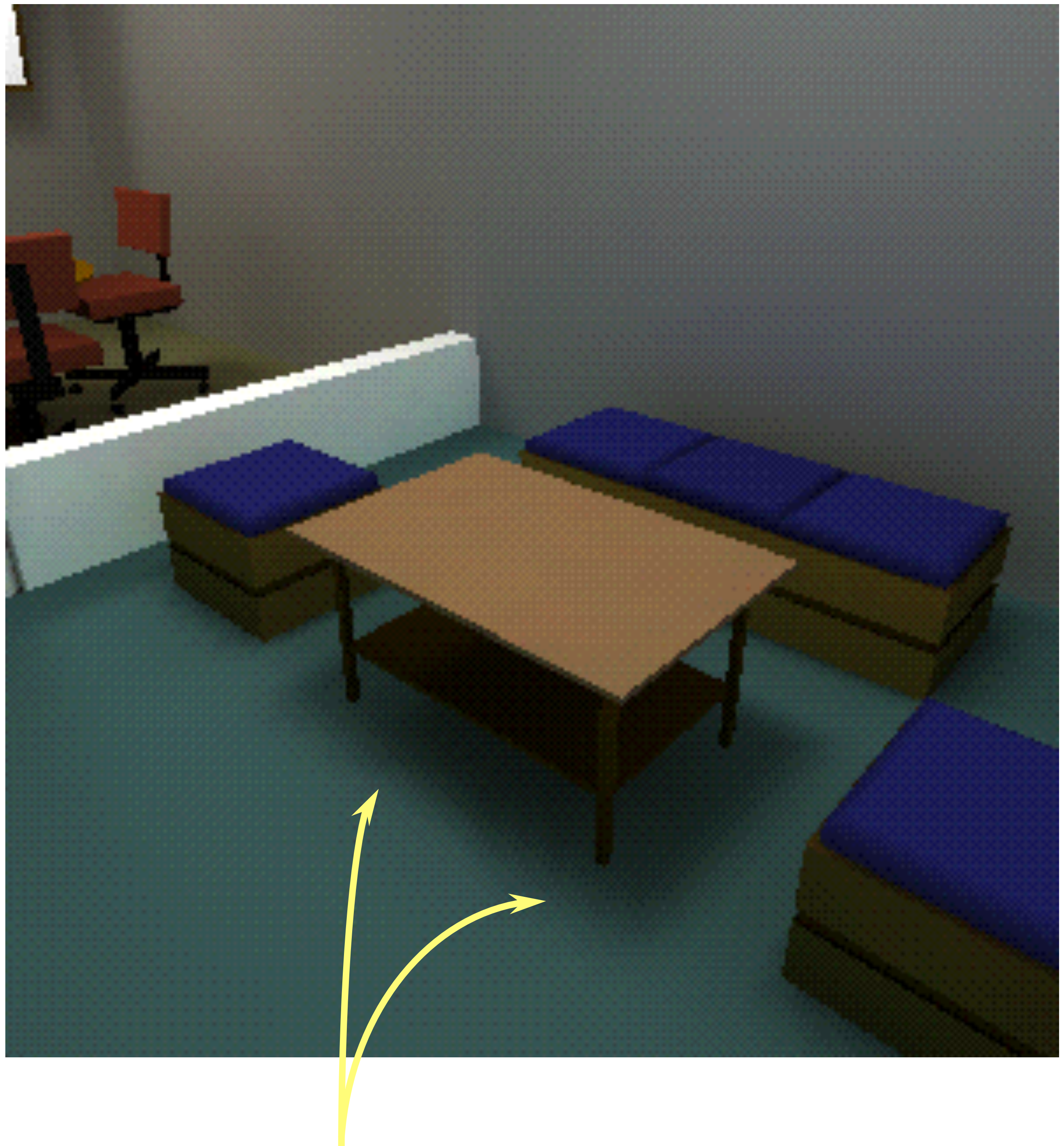


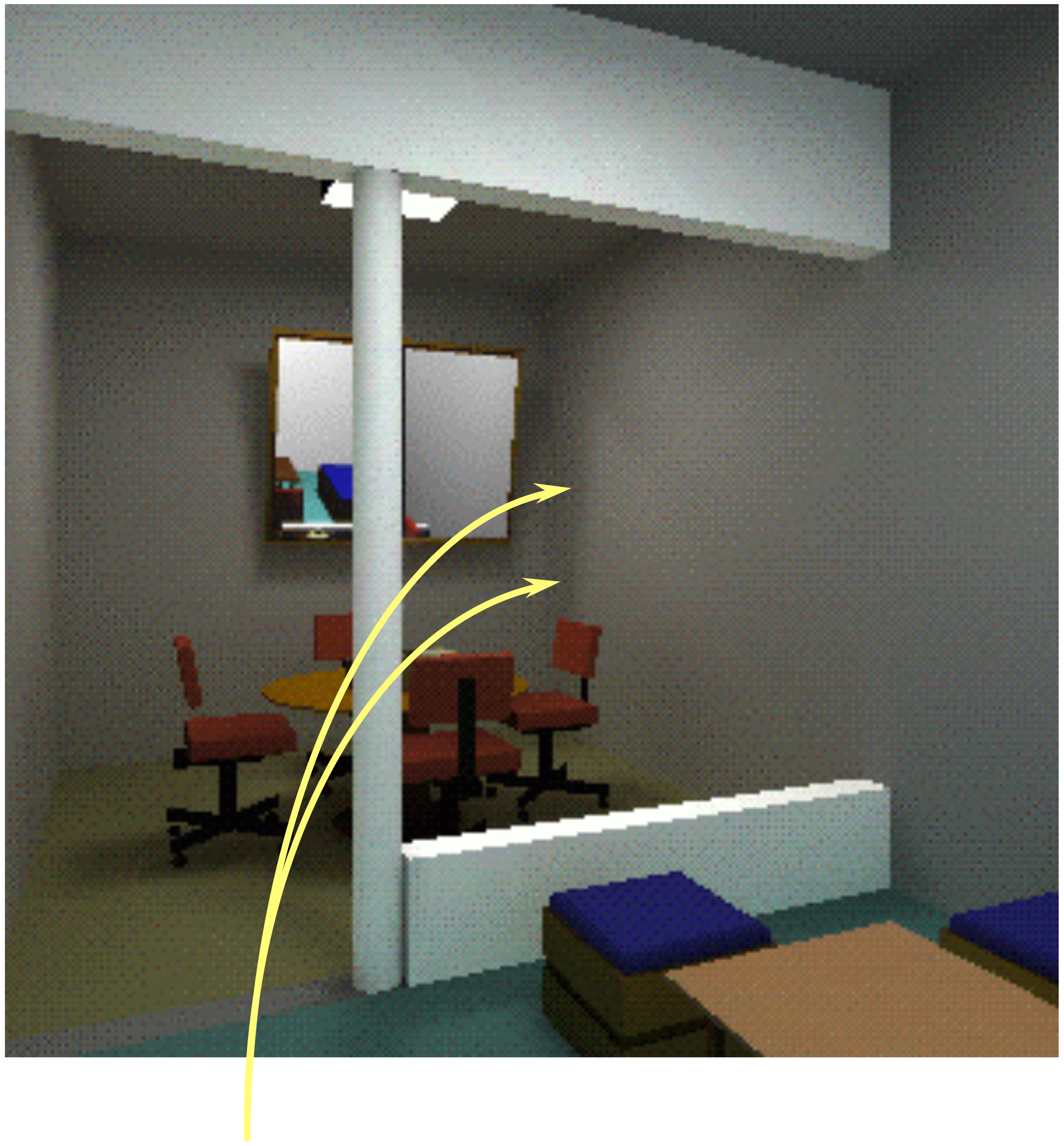
luxrender.net

- LuxRender is a physically based and unbiased rendering engine. Based on state of the art algorithms, LuxRender simulates the flow of light according to physical equations, thus producing realistic images of photographic quality.

Particle Tracing

- *Particle Tracing* is a view-independent technique for global illumination computation
- It is based on the *Monte Carlo* simulation of particle model of light
- *Particle Tracing* computes illumination for surfaces as well as volumes
- Computation is done only once for static scenes





Particle Tracing with Reflections

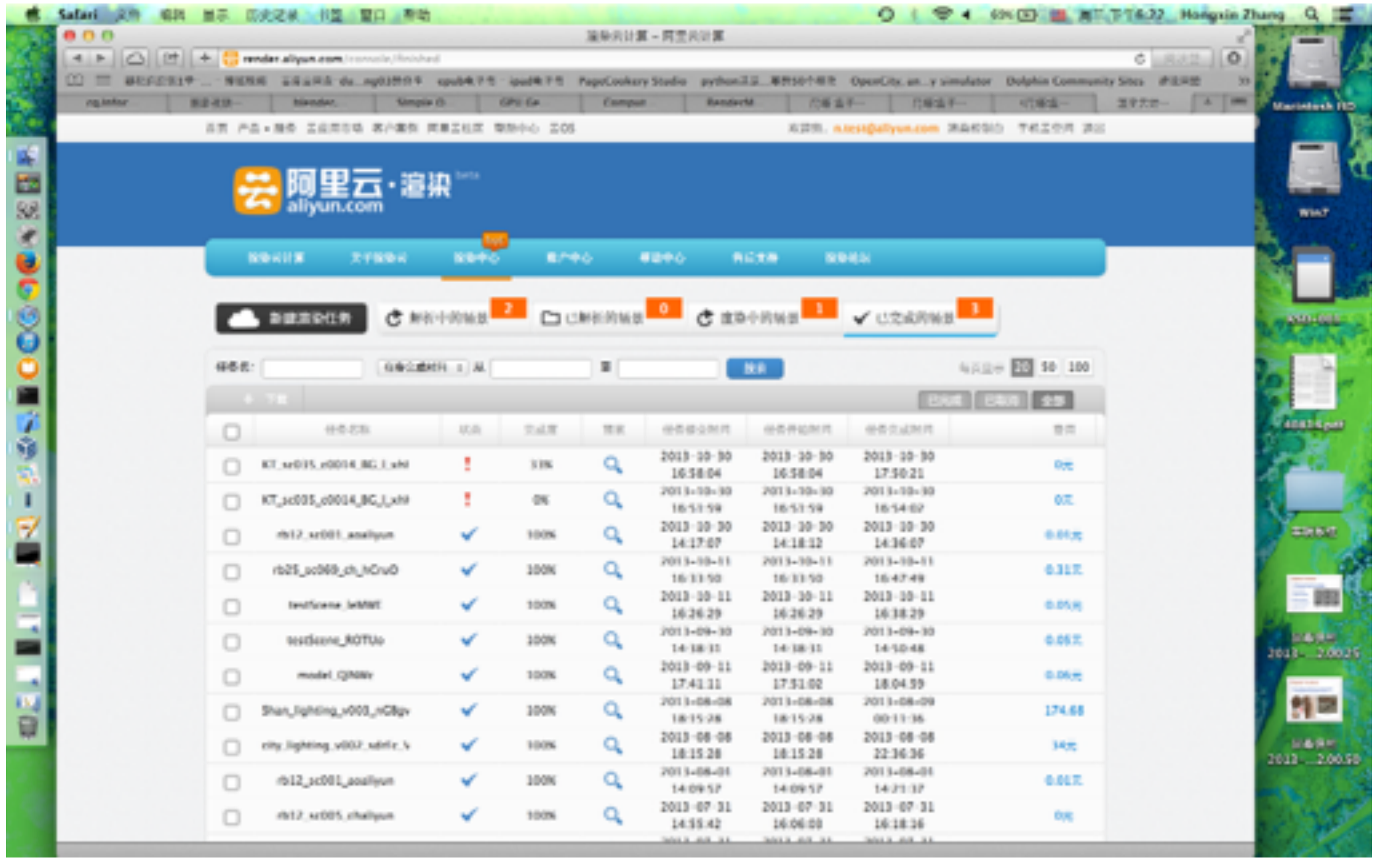
Hybrid Techniques - Particle Tracing and Ray Tracing

Fantastic work from CAD Lab

- RenderAnts Pro (GPU based)
  - http://www.gaps-zju.org/project/renderants.html

Fantastic work from CAD Lab

- Rendering Cloud System (cloud based @ aliyun)
  - http://render.aliyun.com



[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
KT_sc035_c0014_BG_l_xhl	!	33%	🔍	2013-10-30 16:58:04	2013-10-30 16:58:04	2013-10-30 17:50:21	0元
KT_sc035_c0014_BG_l_xhl	!	0%	🔍	2013-10-30 16:51:59	2013-10-30 16:51:59	2013-10-30 16:54:02	0元
rb12_sc001_aoaliyun	✓	100%	🔍	2013-10-30 14:17:07	2013-10-30 14:18:12	2013-10-30 14:36:07	0.01元
rb25_sc069_ch_hCruD	✓	100%	🔍	2013-10-11 16:33:50	2013-10-11 16:33:50	2013-10-11 16:47:49	0.31元
testScene_IeMWt	✓	100%	🔍	2013-10-11 16:26:29	2013-10-11 16:26:29	2013-10-11 16:38:29	0.05元
testScene_ROTUo	✓	100%	🔍	2013-09-30 14:38:11	2013-09-30 14:38:11	2013-09-30 14:50:48	0.05元
model_QJNWv	✓	100%	🔍	2013-09-11 17:41:11	2013-09-11 17:51:02	2013-09-11 18:04:59	0.06元
Shan_lighting_v003_nGBgv	✓	100%	🔍	2013-08-08 18:15:28	2013-08-08 18:15:28	2013-08-09 00:11:36	174.68
city_lighting_v002_sdnfr_k	✓	100%	🔍	2013-08-08 18:15:28	2013-08-08 18:15:28	2013-08-08 22:36:36	34元
rb12_sc001_aoaliyun	✓	100%	🔍	2013-08-01 14:09:57	2013-08-01 14:09:57	2013-08-01 14:21:17	0.01元
rb12_sc005_chaliyun	✓	100%	🔍	2013-07-31 14:55:42	2013-07-31 16:06:00	2013-07-31 16:18:16	0元

up to 6700 computing node

Fantastic work from CAD Lab

- Rendering Cloud System (cloud based @ aliyun)
  - http://render.aliyun.com

Homework 05

- Render your dream car
  - Target: render a car
  - Software: Povray
  - Resolution: > 640x480
  - Contraints:
    - Algorithm:ray tracing
    - effects: mirror / transparent / (soft) shadow

THANK YOU