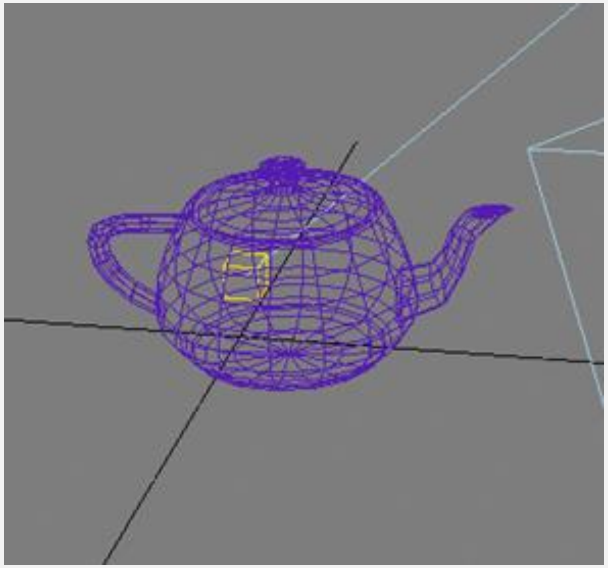
Recollection

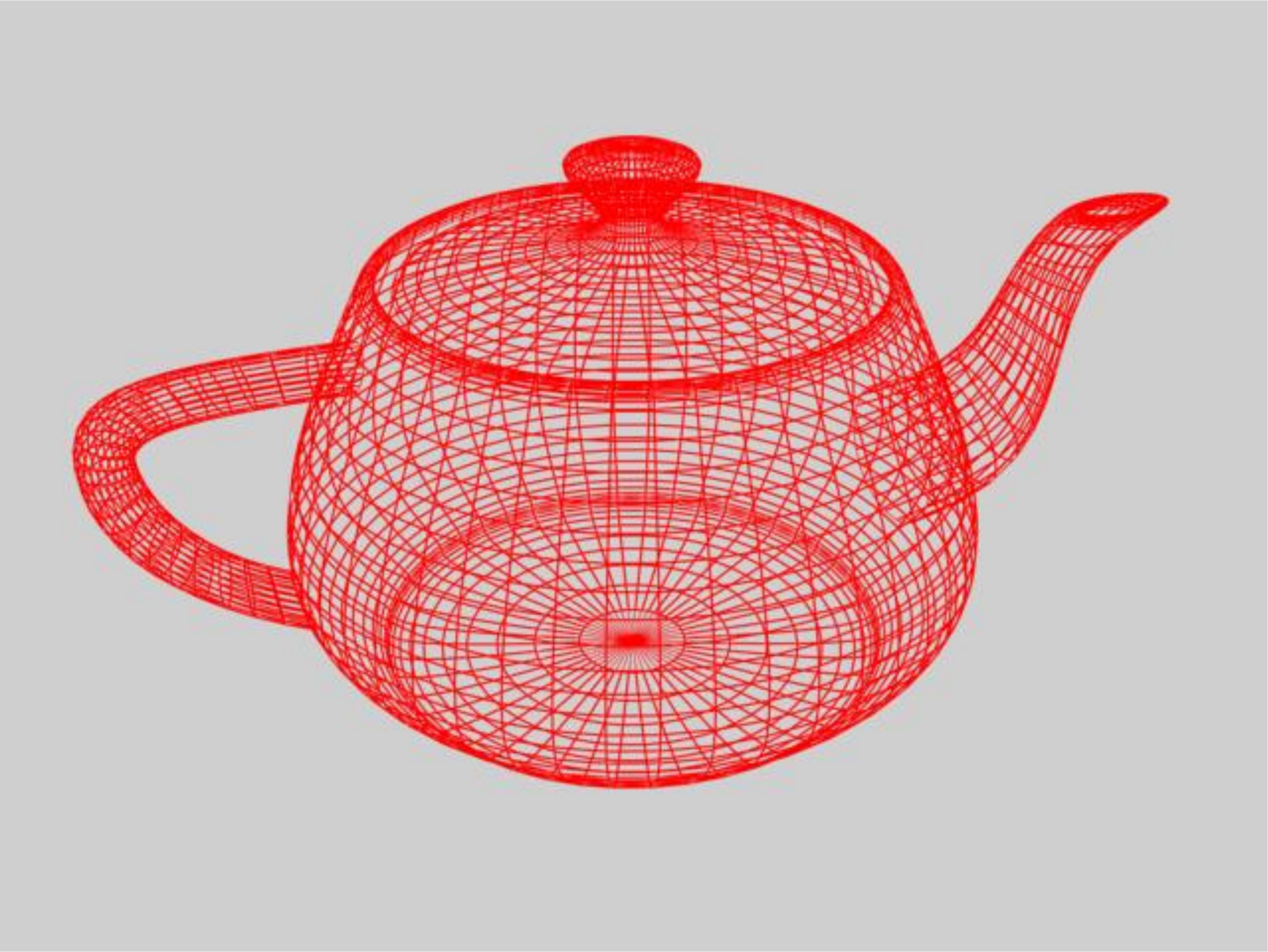
- Models → Pixels

- Model transformation
- Viewport transformation
- Clipping
- Rasterization
- Texturing
- + Lights & shadows
  – Can be computed in different stages



So far we came to…

Geometry model

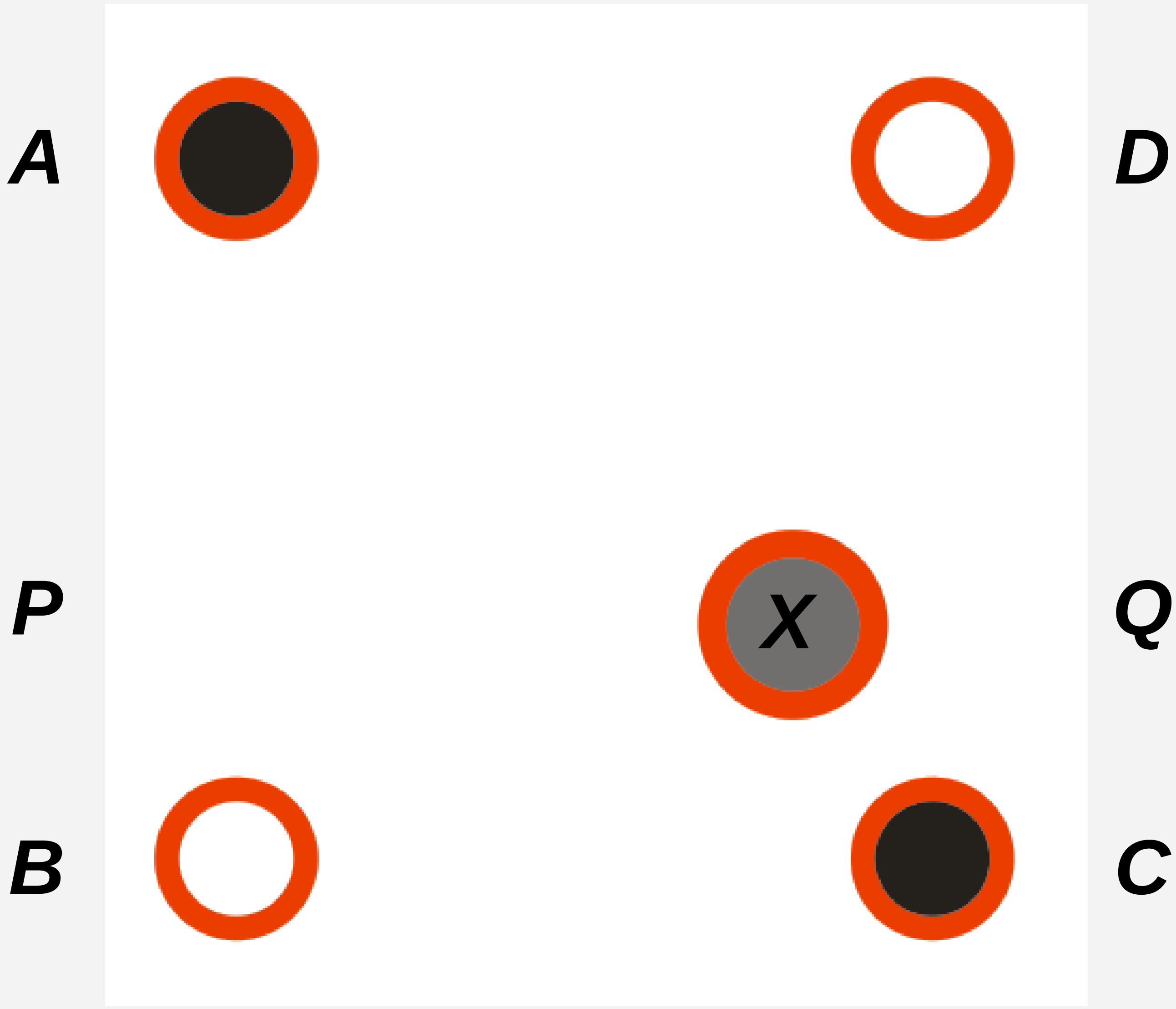


Surface color

Now: Shading

Important recollections

Bilinear interpolation



Bilinear interpolation

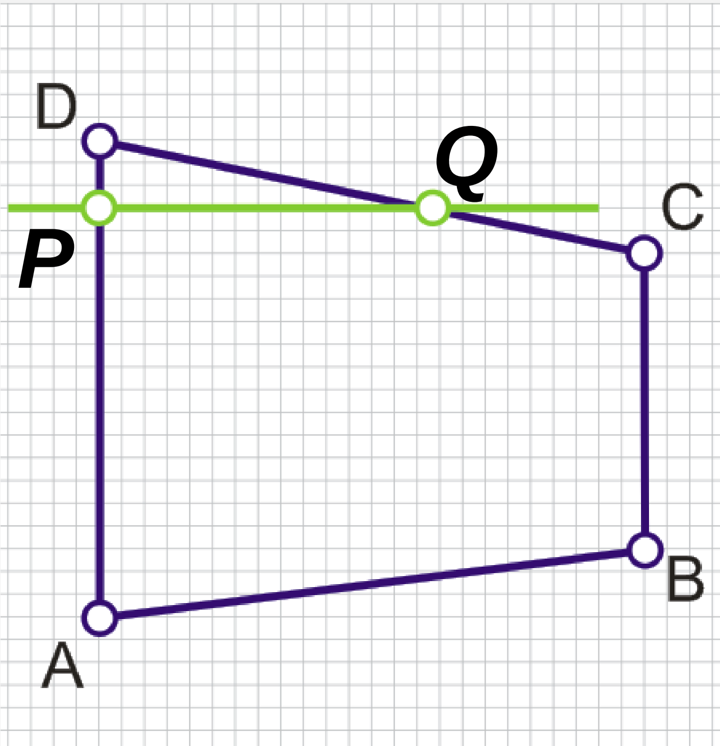
- 4 corner points ***A,B,C,D*** with known values
- 1 internal point ***X*** with unknown value
- ***P = A + u.(B-A), Q = D + u.(C-D)***
- ***X = P + v.(Q-P)***

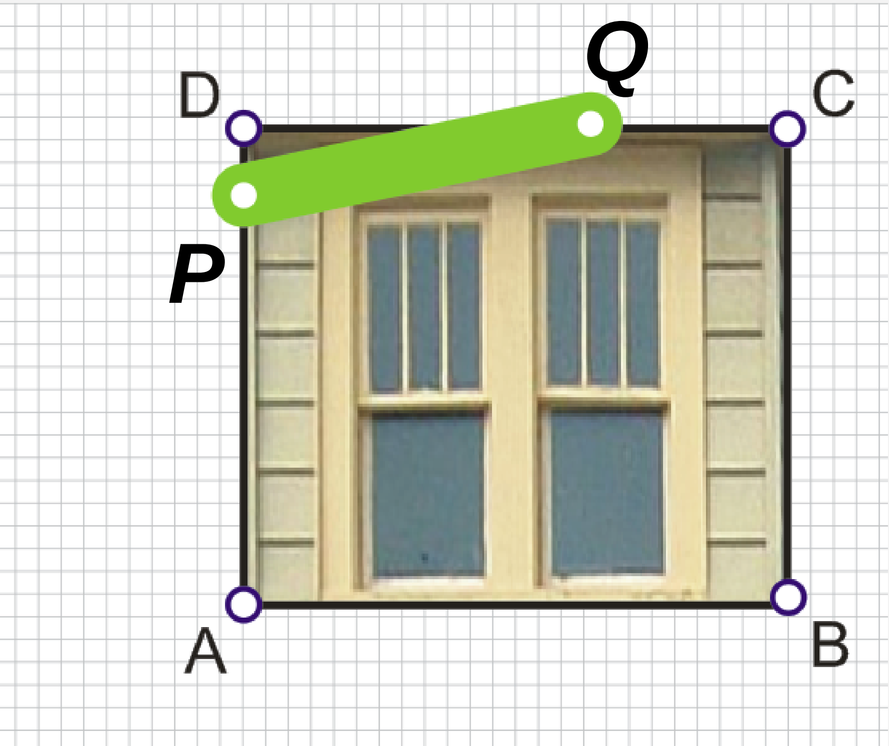
- Matrix representation

$$X = (1-u, u)\begin{pmatrix} A & D \\ B & C \end{pmatrix}\begin{pmatrix} 1-v \\ v \end{pmatrix} \qquad u \in \langle 0,1 \rangle, v \in \langle 0,1 \rangle$$

Application: texture mapping

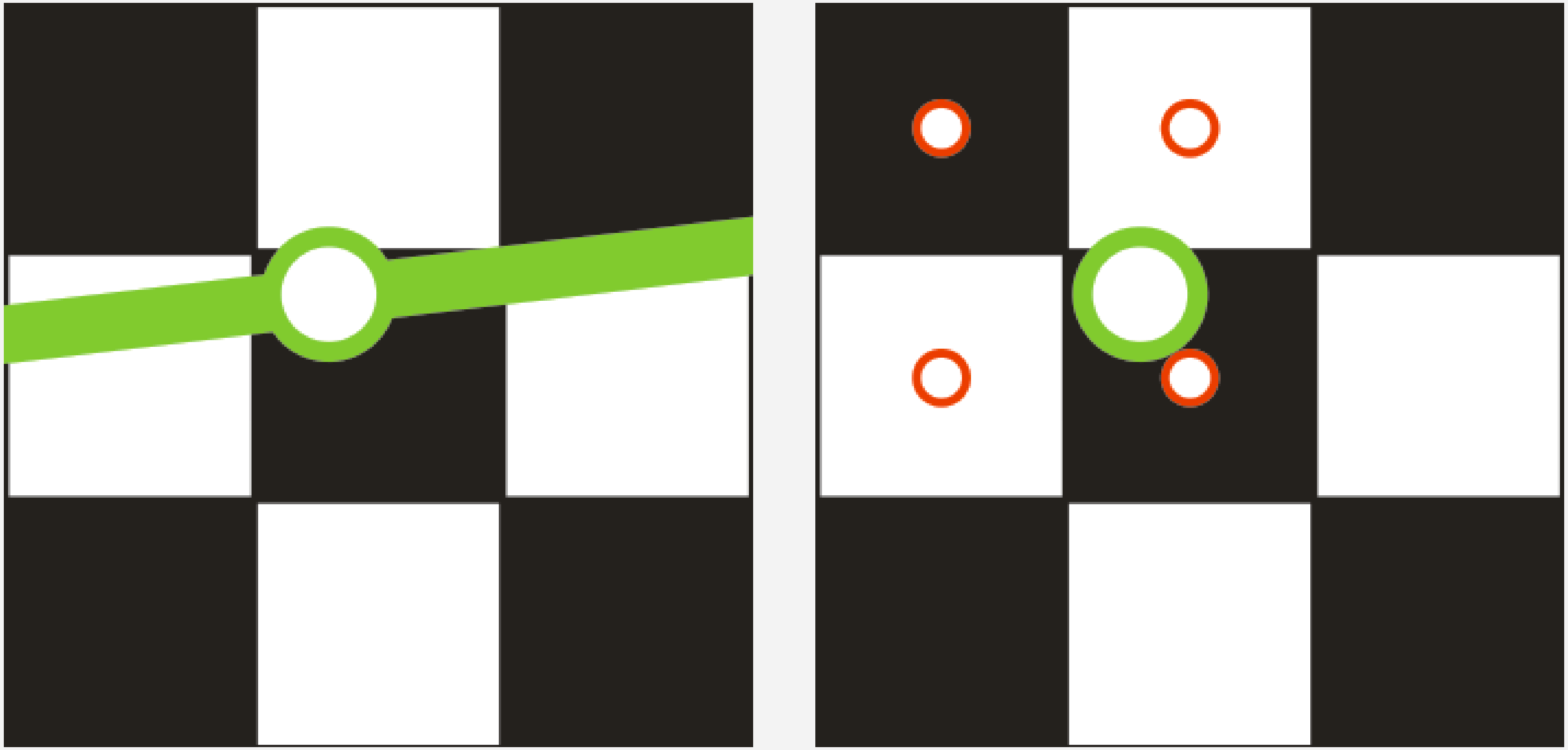
- Interpolate D↔A = P, D↔C = Q, P↔Q = X





Application: texture filtering

- Consider 4 neighboring texels
- Weighted average



Lighting and shading

General problem

- For a point in space, calculate lighting conditions and modulate the inherent object color to produce final pixel color

Lighting and shading

- What is lighting
  - Computing amount of radiance (per wavelength) reflected from object towards the camera

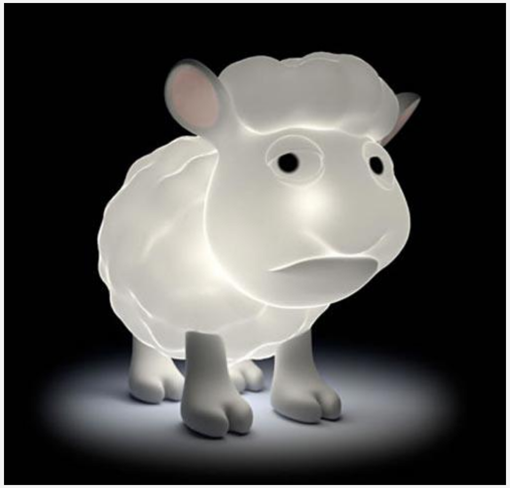
- What is shading
  - Creating illusion of space in planar images
  - Usually uses lighting but other options are available too – e.g. depth shading

Lighting

Light source types

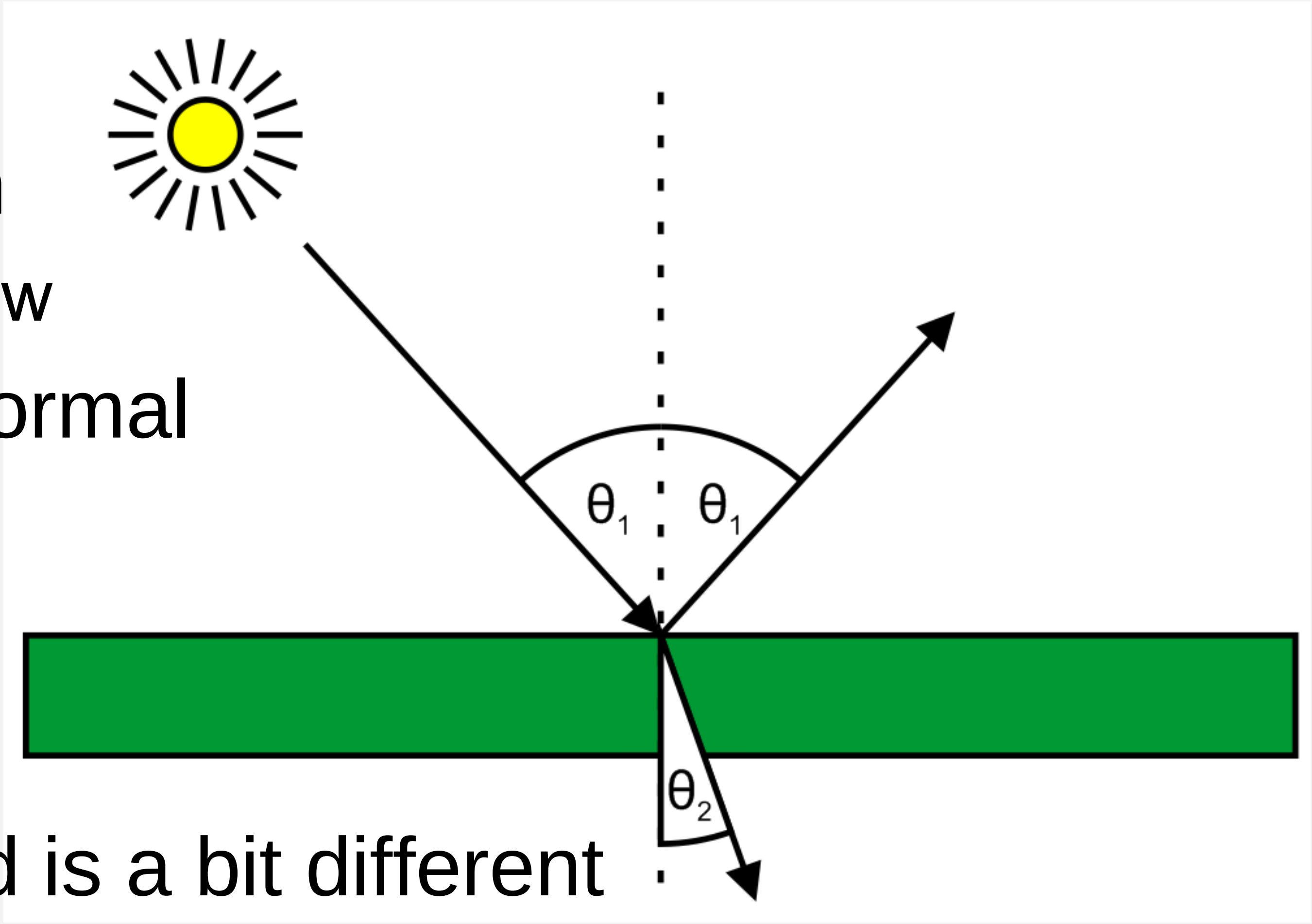
- Omnidirectional
- Spotlight
- Area
- Directional
- Object

- what are the differences?



Elementary theory

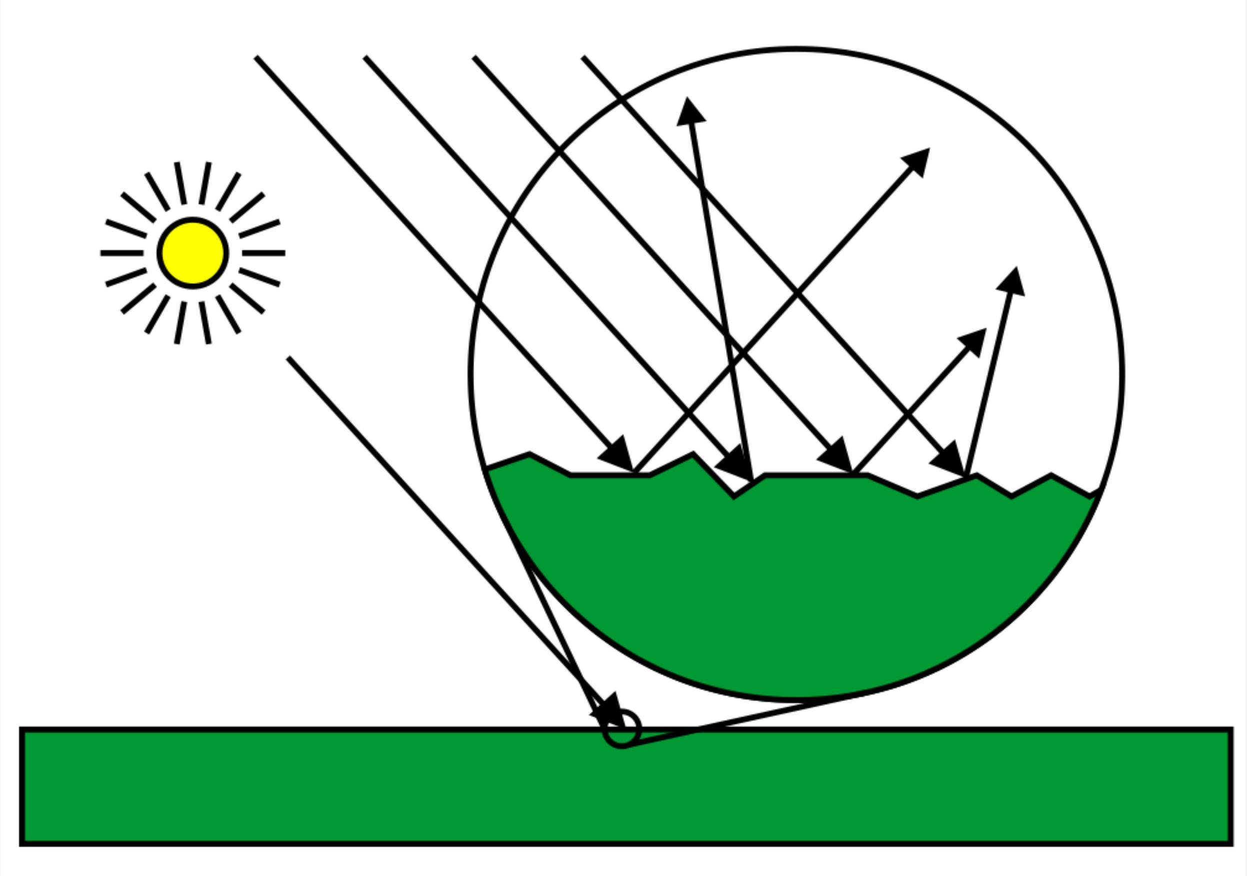
- Light-surface interaction
- Reflection
- Refraction
  - Snell's law
- Surface normal vector



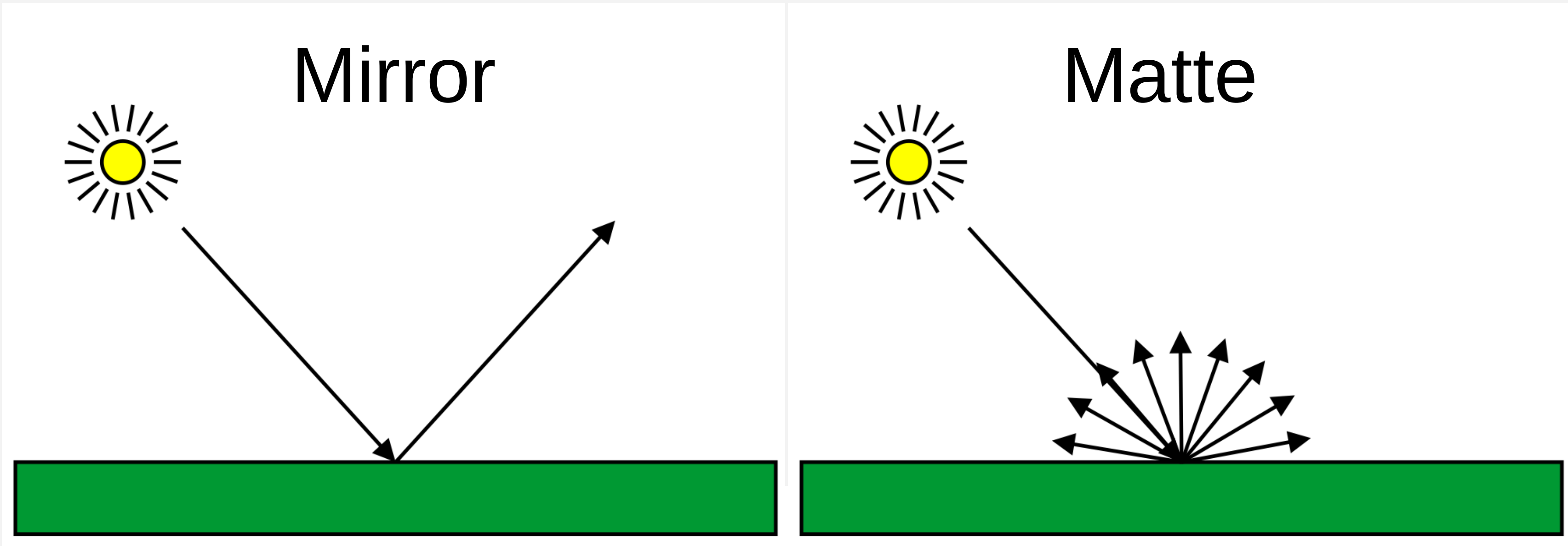
- Real world is a bit different

Surface types

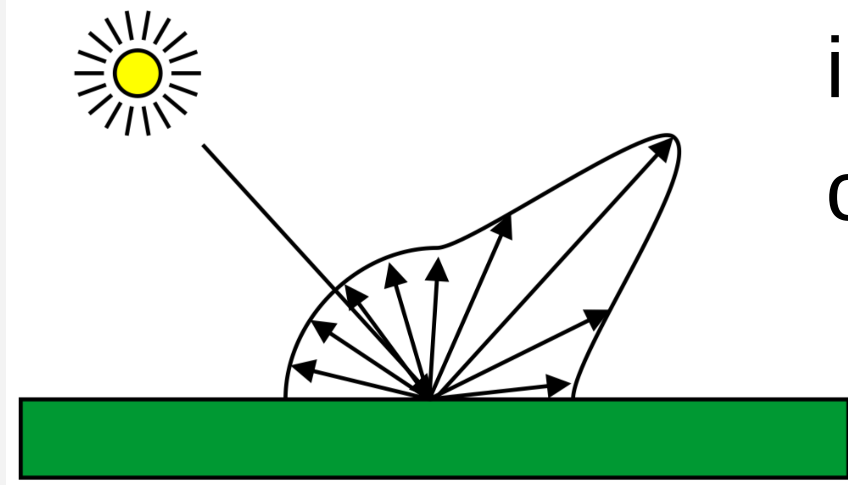
- Reflective
- Diffuse – Lambertian
- Both



Light reflection distribution



directional component



indirectional component

Lighting models

- Empiric – e.g. Phong lighting model
  - cheap computation
  - physically incorrect
  - visually plausible

- Physically-based
  - energy transfer, light propagation
  - closer to real-world physics
  - expensive

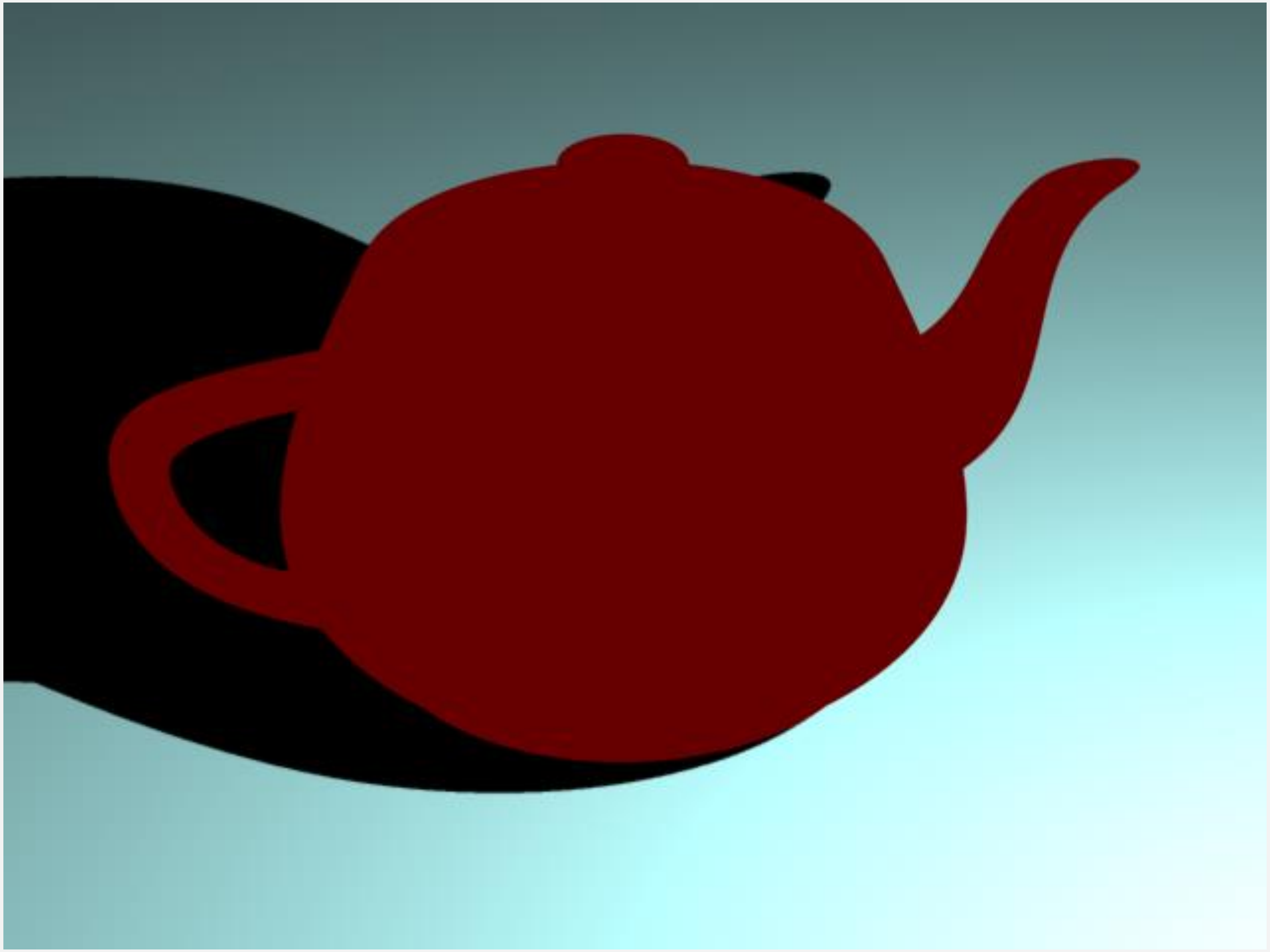
Local illumination models

- Fast but inaccurate
- Ignore other objects (i.e. it’s not global)
- Empirical (no physical background)
- Many physical effects are impossible to achieve

- Computer games, real-time rendering

Diffuse light

Ambient light

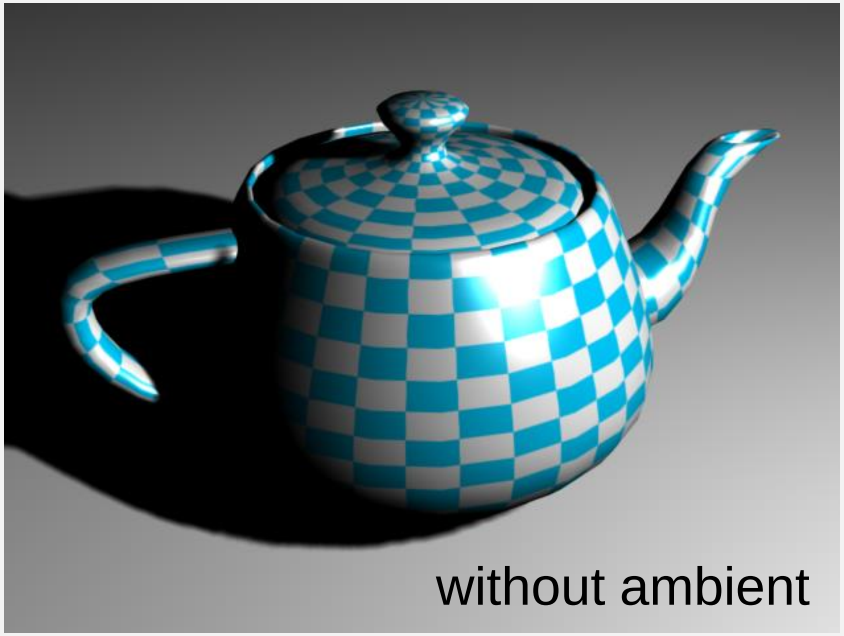


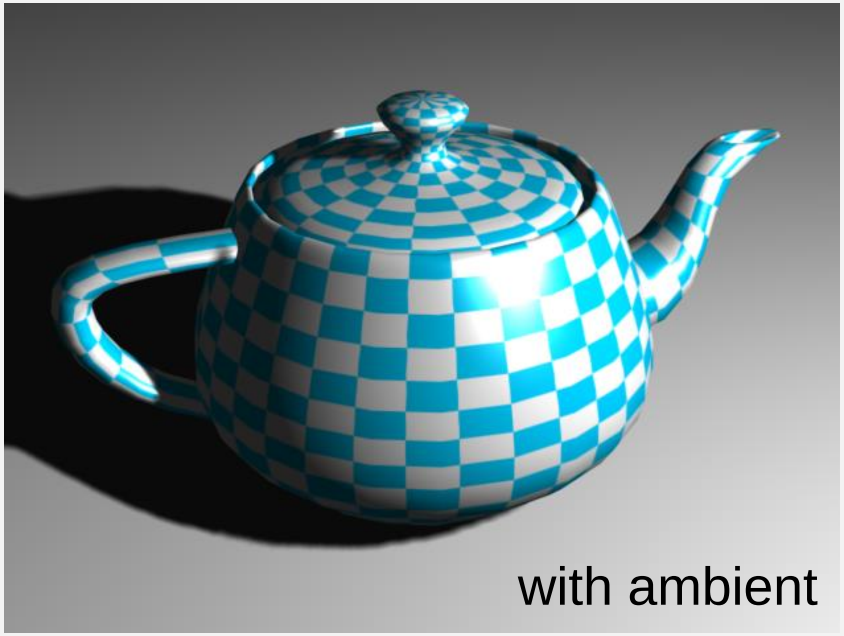
Diffuse + ambient

Diffuse + ambient + specular

Phong lighting model

- **Ambient** + Diffuse + Specular components

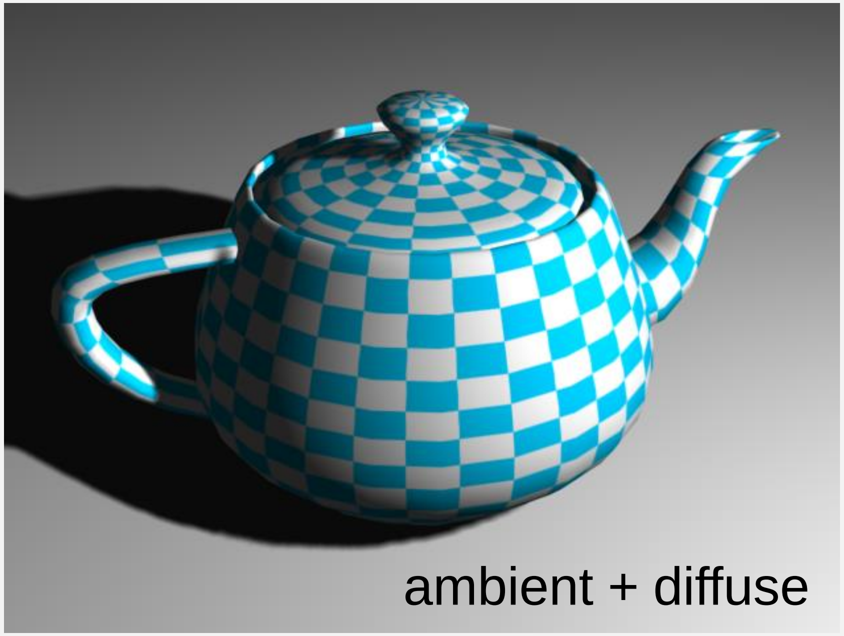


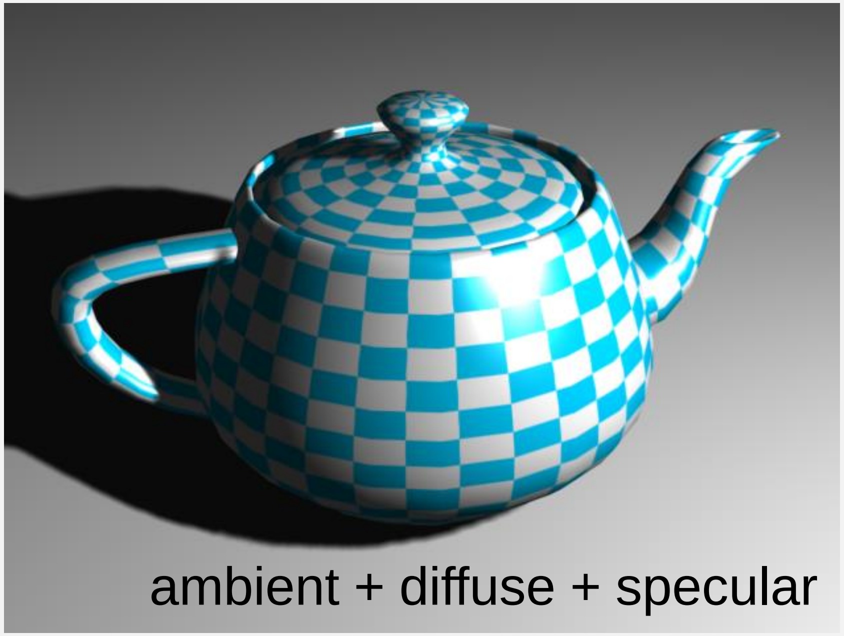


- Simulates global light scattered in the scene and reflected from other objects

Phong lighting model

- Ambient + **Diffuse** + Specular components



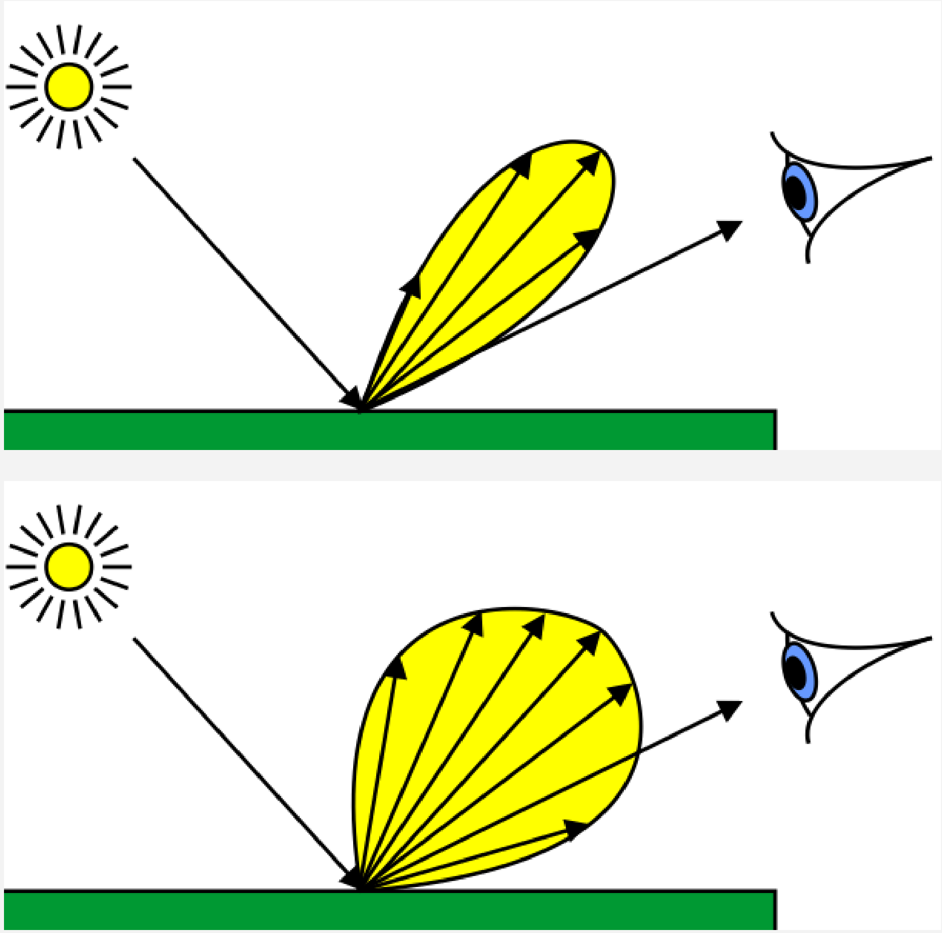


- Lambert law $I_d = \vec{n} \cdot \vec{l}$

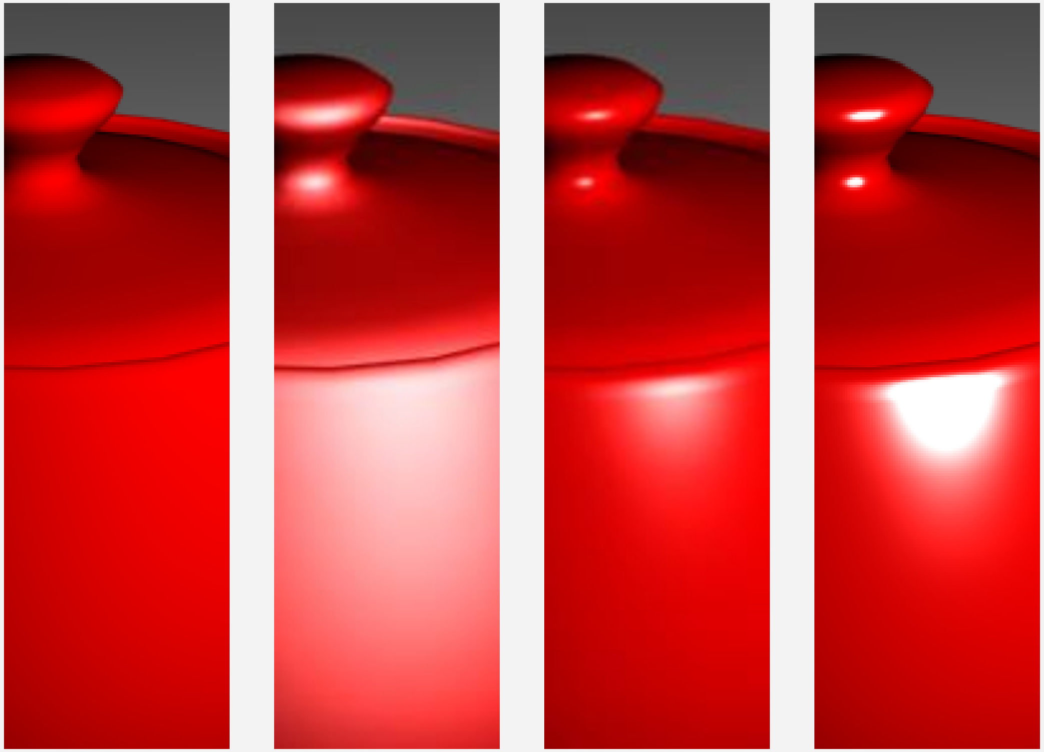
Phong lighting model

- Ambient + Diffuse + **Specular** components
- Directional
  - – view vector

$$I_s = \left(\vec{r} \cdot \vec{v}\right)^{k_{shine}}$$



Specular component



- **r.v** = |r|.|v|.cos(rv)
- absolute parameter k_s
- exponential parameter shininess (gloss)

Phong lighting model

$$I = k_a I_a + \sum_{\forall lights} \left(k_d I_d + k_s I_s \right)$$

- k, I coefficients can depend on wavelength

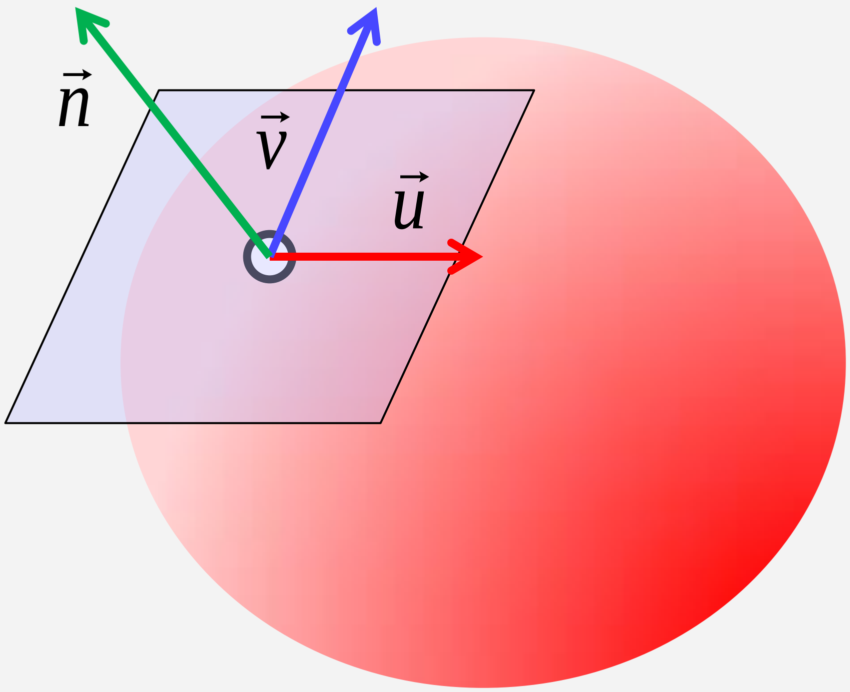
- what defines surface lighting properties?
  - k_a, k_d, k_s, k_{shine}

Other lighting models

- Blinn-Phong
  - generalization of Phong's model
- Cook-Torrance
  - microfacets
- Oren-Nayar
  - rough surfaces
- Anisotropic microfacet distribution

Surface normal vector

- Perpendicular to the surface at the point
- Computation:
  - Usually from tangent vectors
  - Vector product $\vec{n} = \vec{u} \times \vec{v}$
  - Depends on the object representation



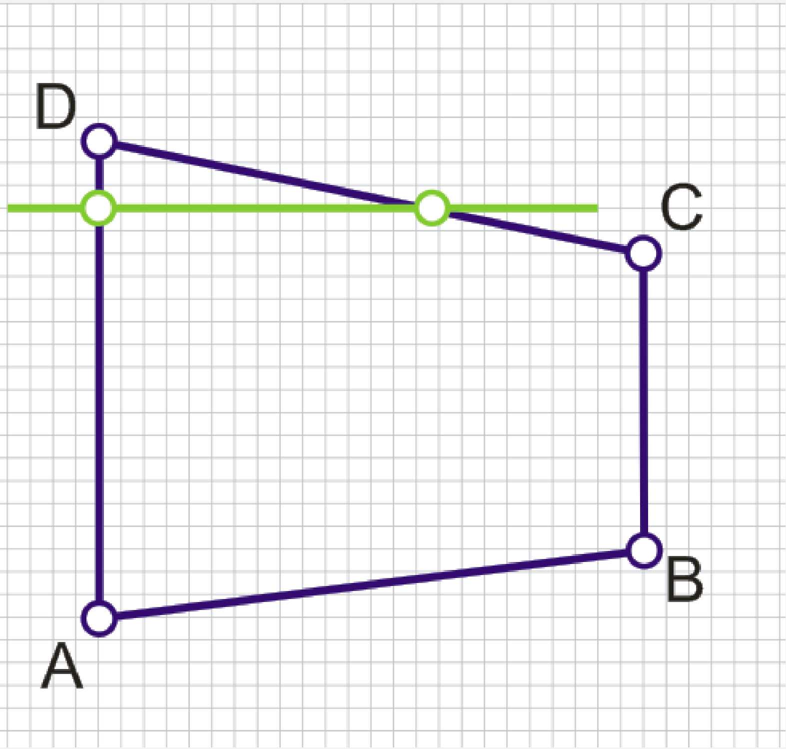
- Vector normalization $$\hat{n} = \frac{\vec{n}}{|\vec{n}|}$$

Tangent vectors

- Parametric representation
    - $X = x(u,v)$
    - $Y = y(u,v)$
    - $Z = z(u,v)$
  - Partial derivation by $u,v \rightarrow$ vectors $t_{u,}$ t_u
- Polygonal representation
  - Tangent vectors are edge vectors
  - Mind the orientation!

Lighting a polygon

- Scanline rasterization
- For each pixel – evaluate lighting model
  - compute normal vector, view vector, light vector
  - get surface parameters
  - evaluate formula

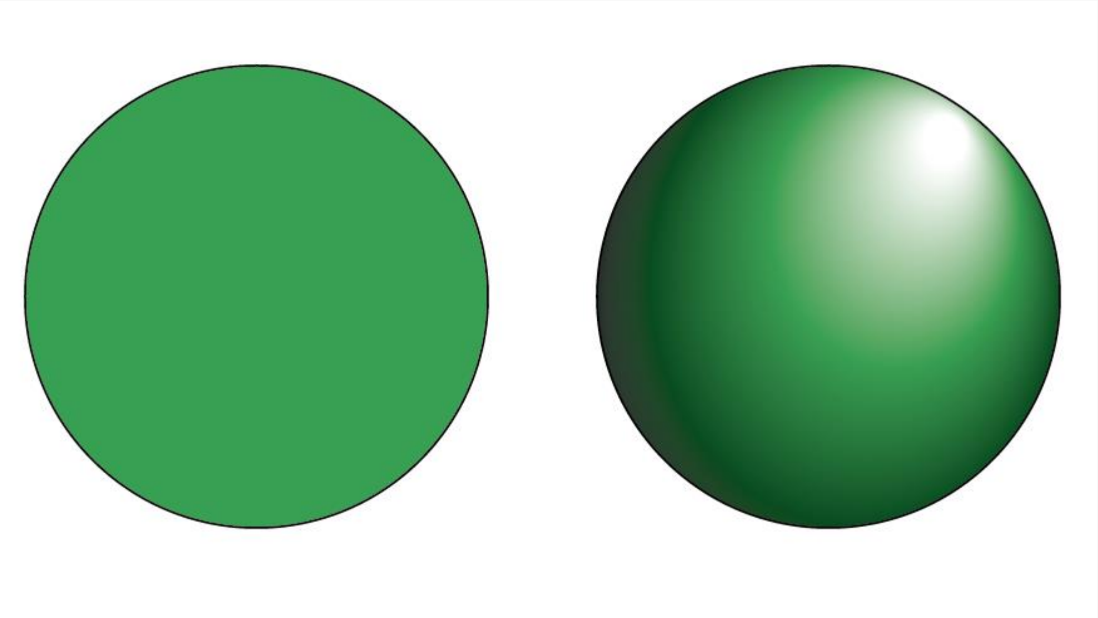


- Expensive
  - therefore: shading

Shading

Shading

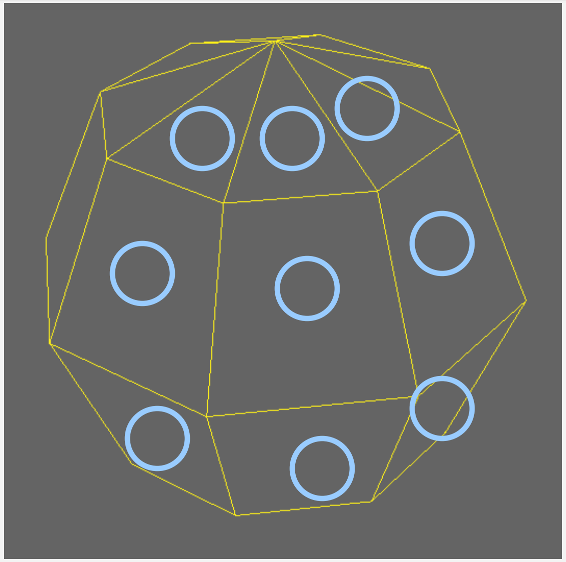
- Object color is altered to give impression of light and depth



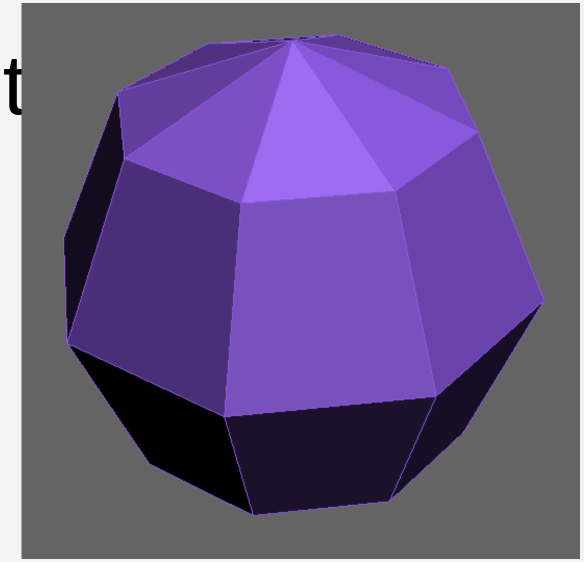
- Usually incorporates lighting
- Often only an approximation of real physics

Two stages of lighting

1. Evaluate illumination for some object
   = LIGHTING

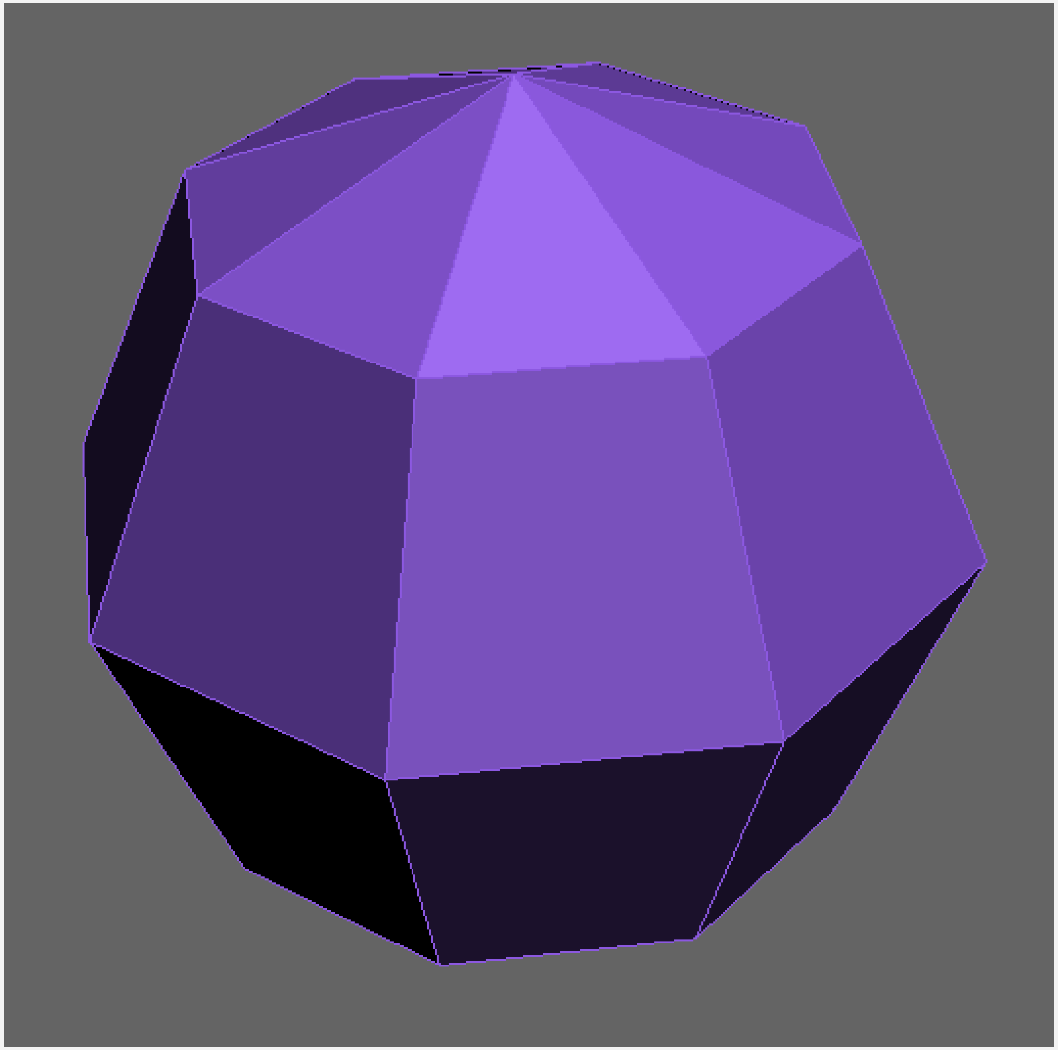


2. Use results from (1) to comput of the rest of the object
   = SHADING



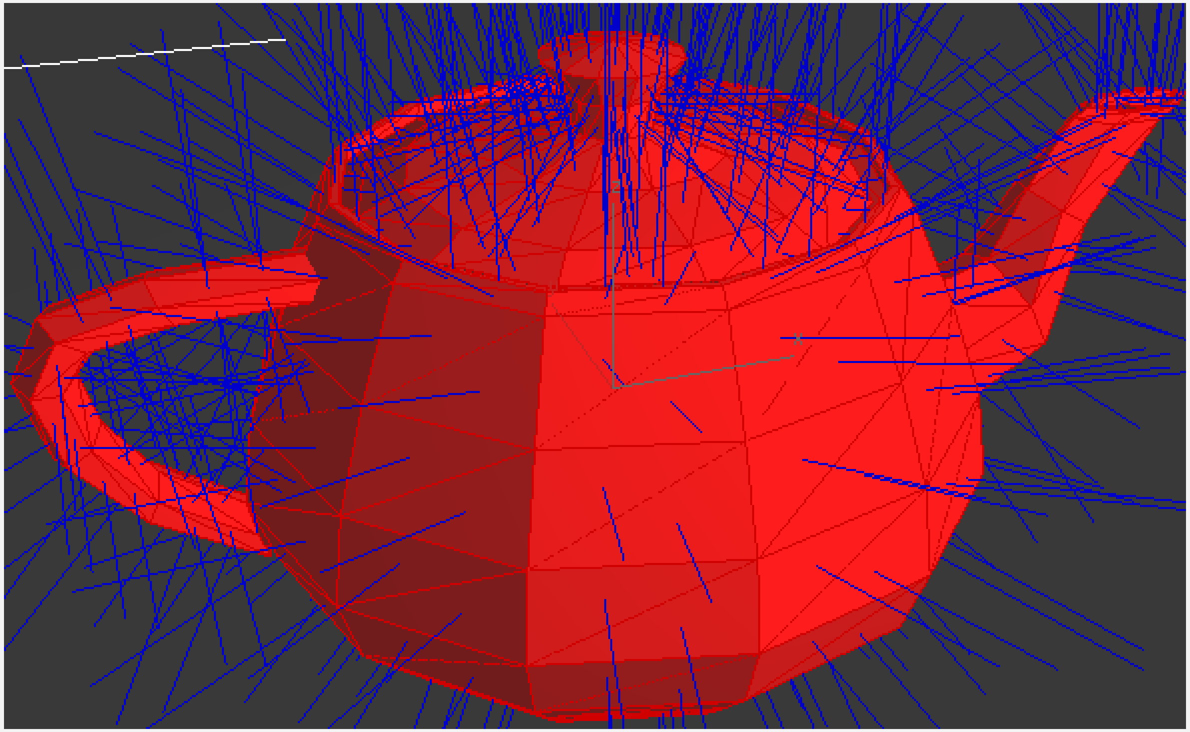
Flat shading

- One normal per face
- Entire face = one color



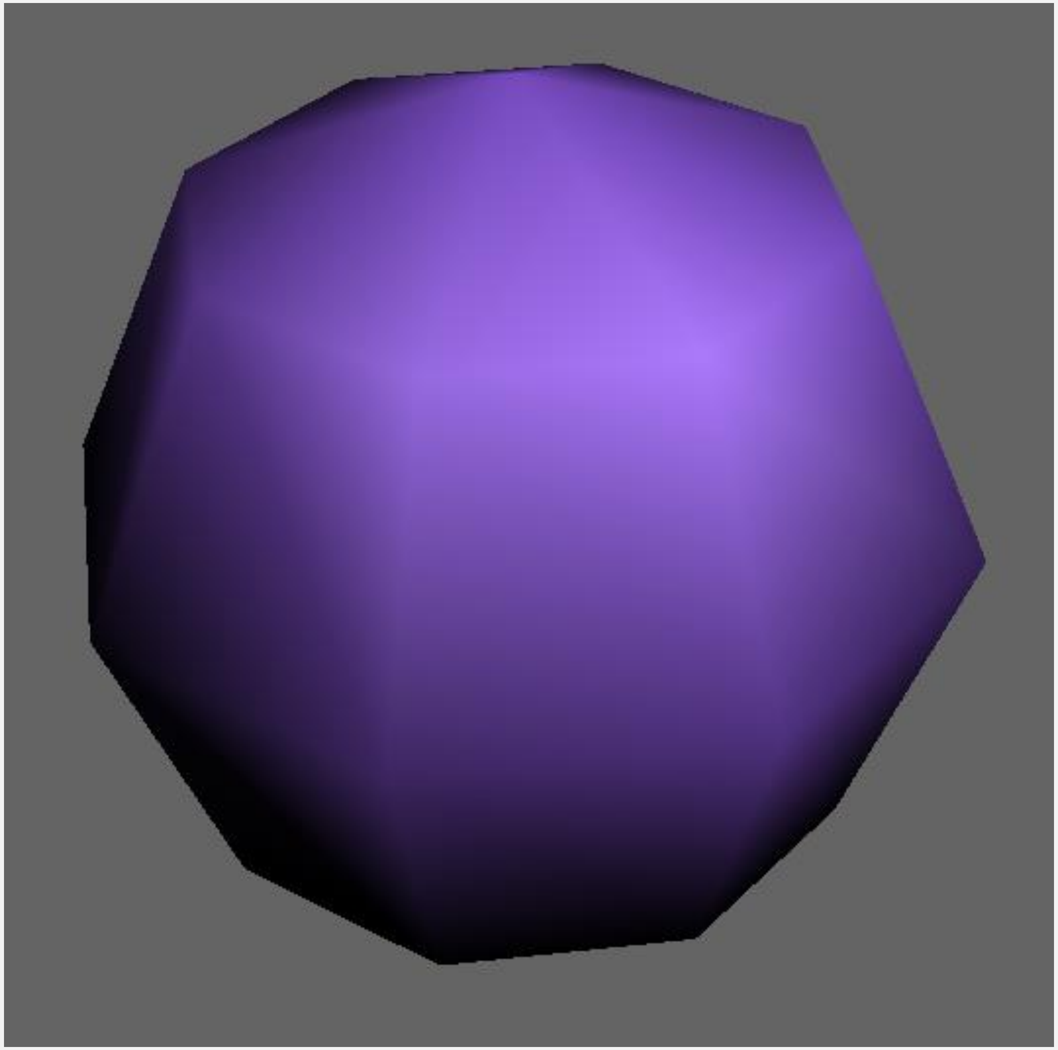
Flat (constant) shading

- 1 normal vector per object face (polygon)
- 1 lighting value per object face
- Entire polygon = 1 color



Gouraud shading

- Per-vertex lighting
- Color is interpolated over the face

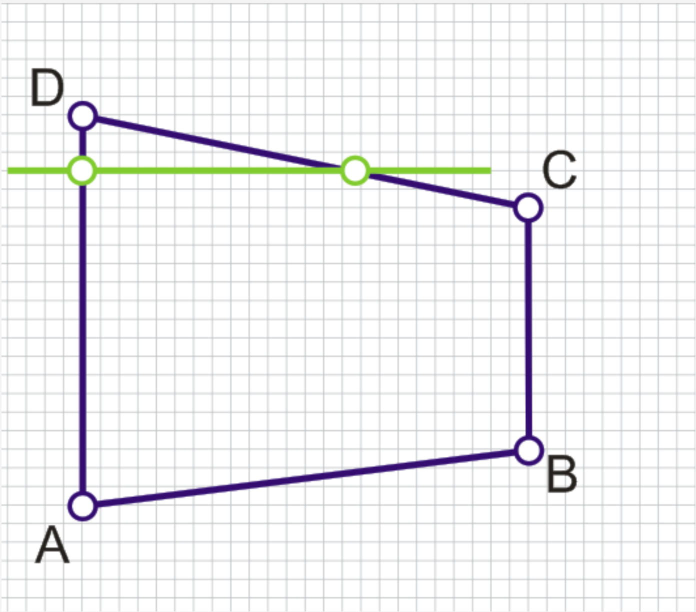


Gouraud shading

- 1 normal vector per 1 surface vertex
  - i.e. 4 lighting values / quad, 3 values / triangle
- Rest of the polygon – lighting value interpolation
- Bands
- Chance of missing specular
- Realtime

Example

- A[0, 0]   = 80% intensity
- B[10, 6] = 20%
- C[10, 9] = 80%
- D[0, 10] = 40%



- Interpolate light intensity at S[5, 8]
- HINT: Bilinear interpolation

  A…D => P   B…C => Q   P…Q => S

Phong shading

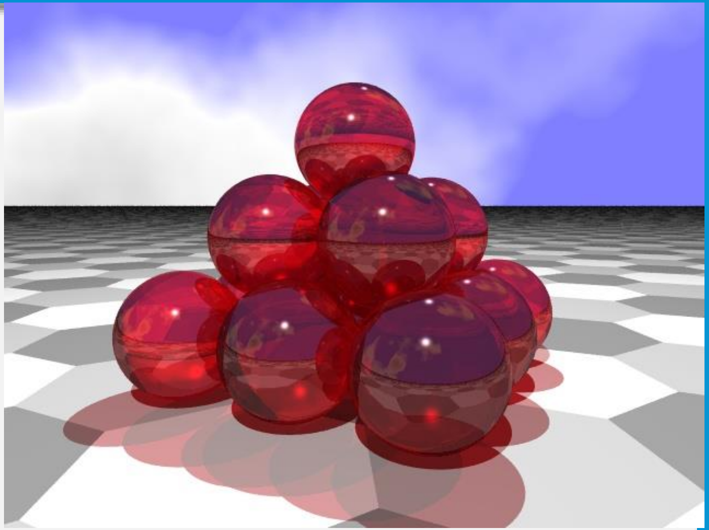
- NOT Phong lighting model
- Entire surface normal is interpolated instead of interpolating only the lighting value
- Per pixel lighting

- Slower

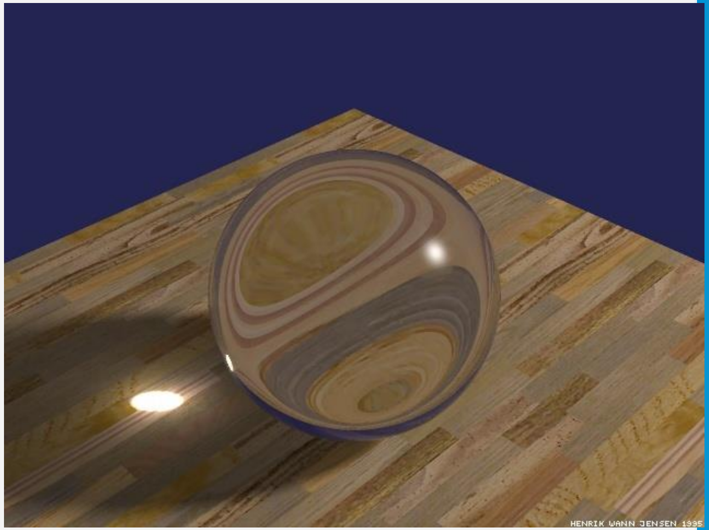
Towards photorealism

Real world effects

- light refraction
- mutual object reflection
- caustics
- chromatic aberration
- color bleeding
- (soft) shadows

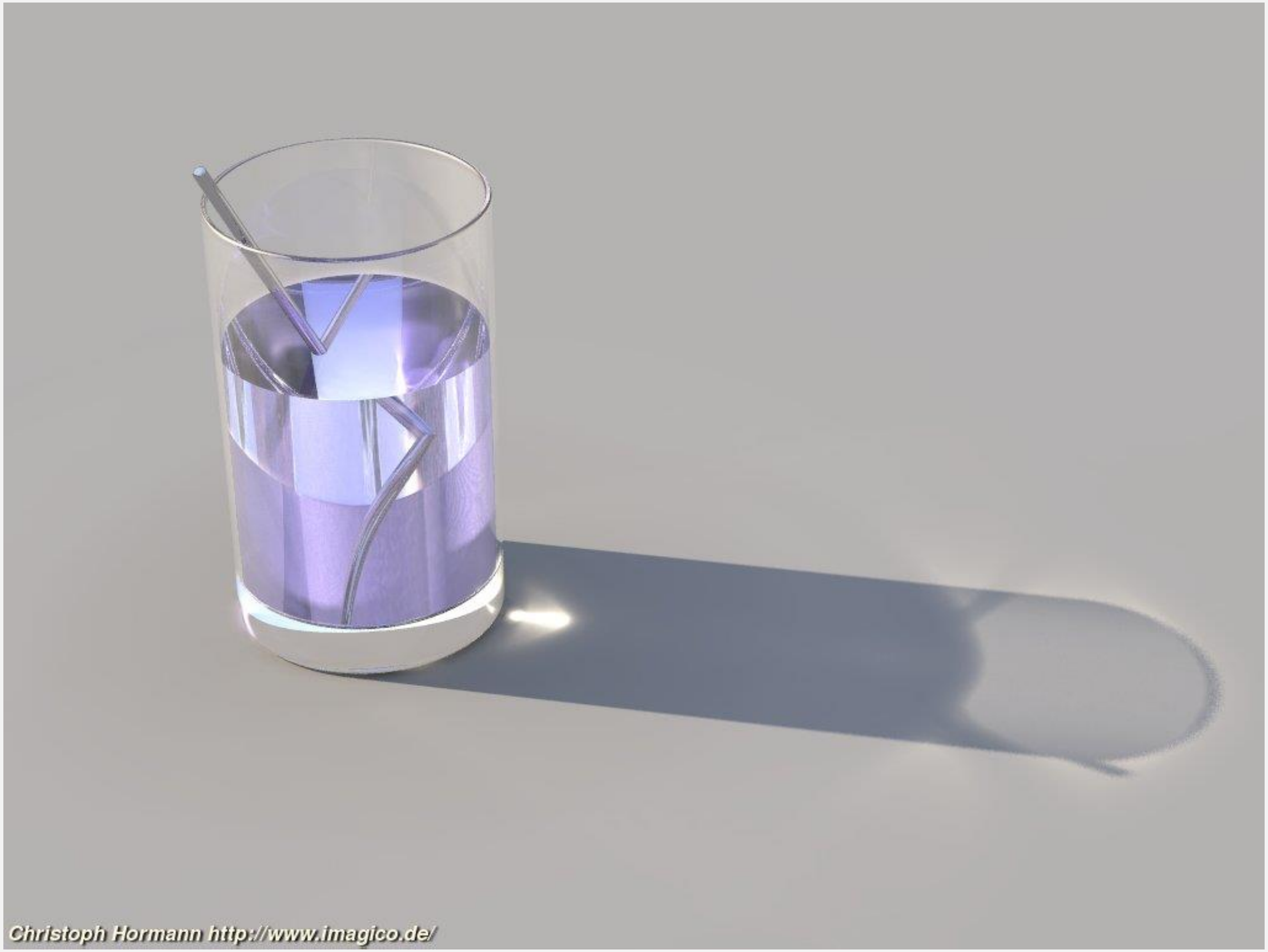


http://math.hws.edu/eck

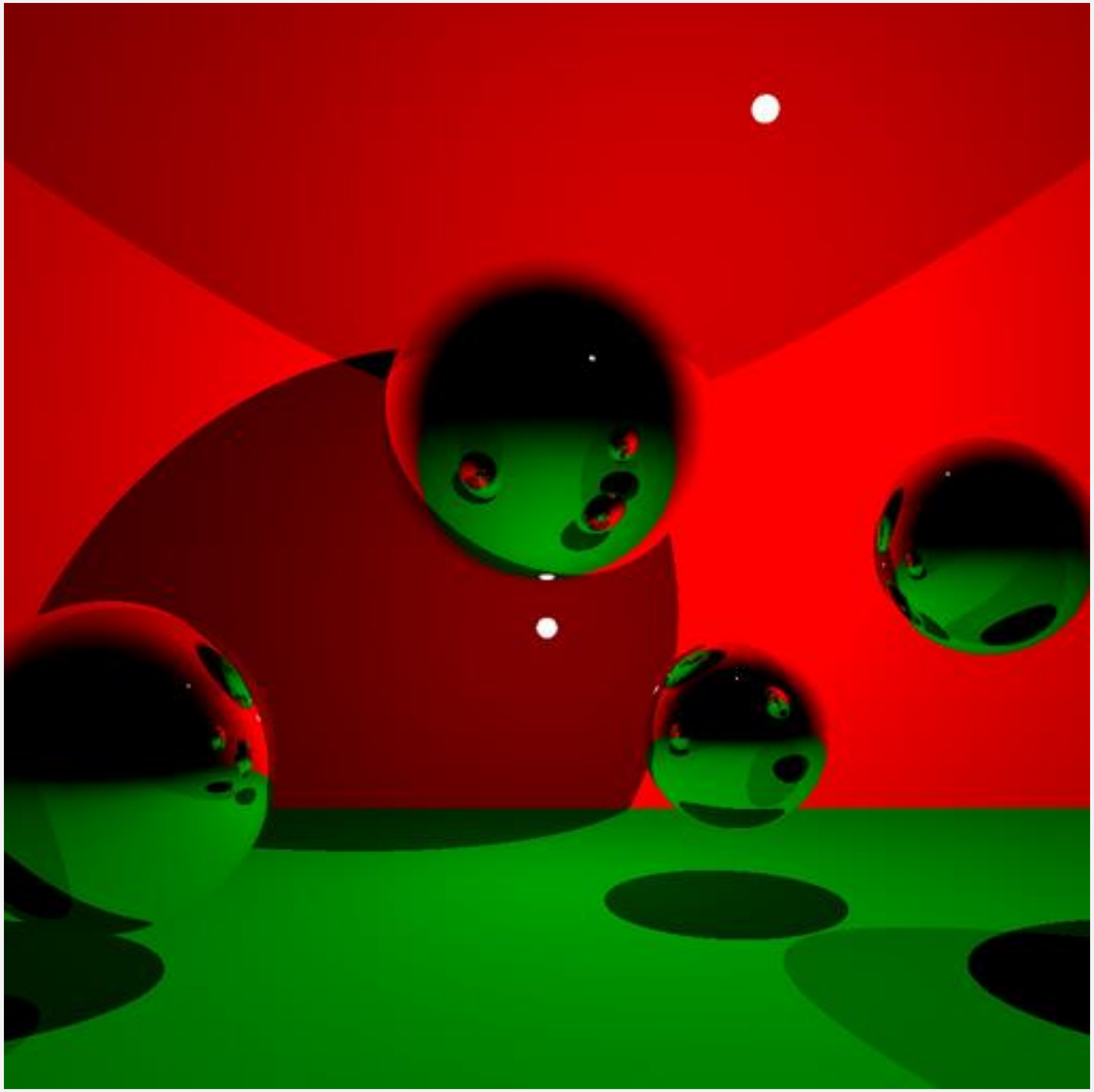


http://graphics.ucsd.edu/~henrik/

Refraction, caustics

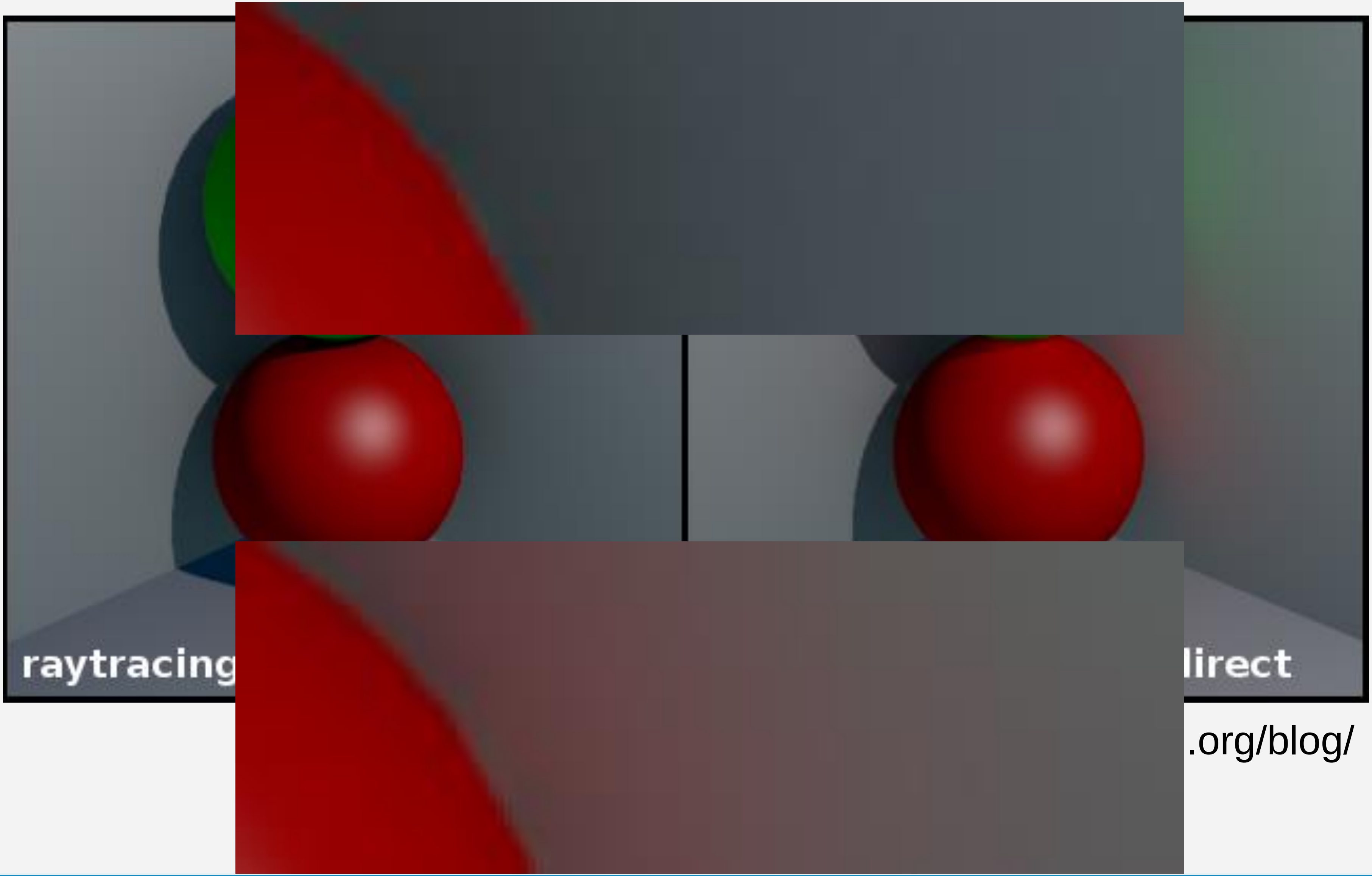


Reflections



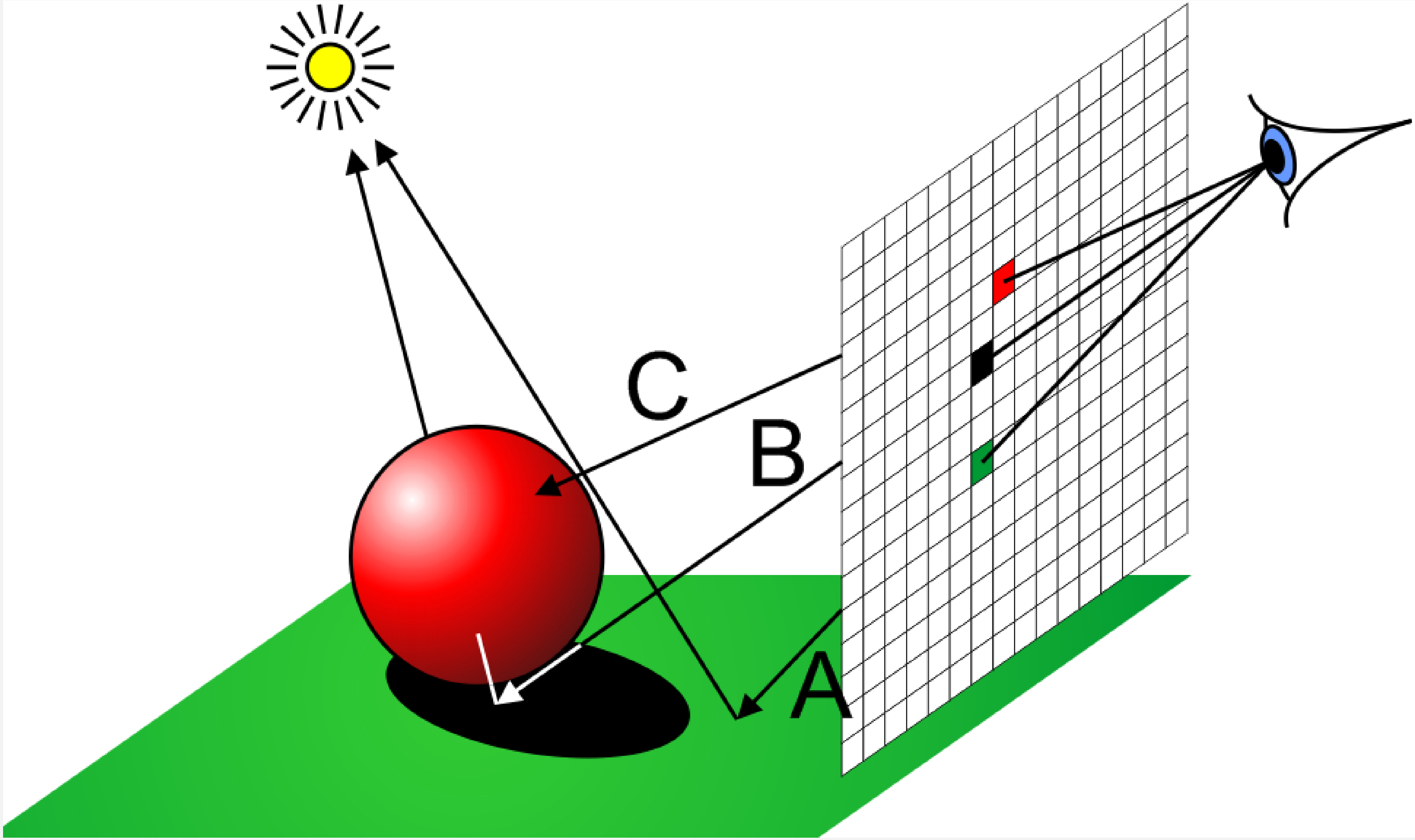
Chromatic aberration

Color bleeding



.org/blog/

Raytracing



Raytracing

- Tracing a beam from viewer’s eye through each screen pixel.
- Find first beam intersection with objects
- Compute local lighting
- Trace reflected and refracted beams
- Combine the results with local result
    - recursively

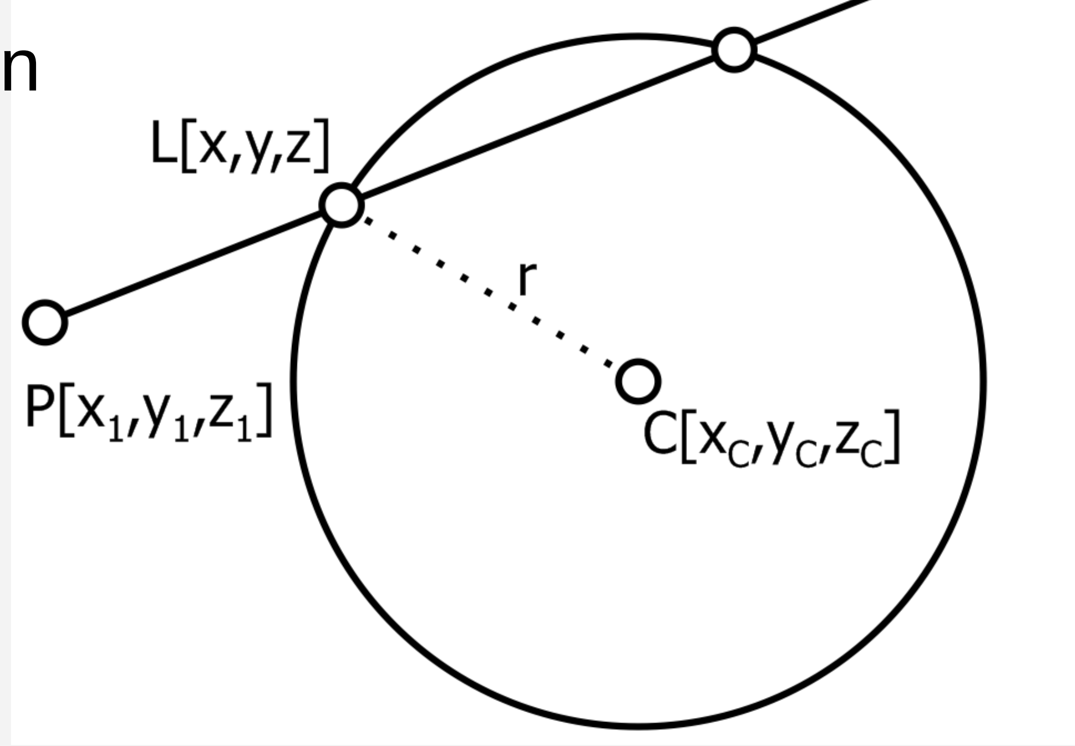
Raytracing – what's inside

- Line-object intersection
  - expensive computation
  - speed-up by e.g. scene subdivision (octree) or bounding volumes
  - take the nearest intersection

- Example intersections:
  - Sphere
  - Triangle

Line-sphere intersection

- Line A,*p* : $\mathbf{L = P + t} * \boldsymbol{p}$
- Sphere C,r : $\mathbf{(S - C)^2 = R^2}$
- Intersect: $\mathbf{(P + t} * \boldsymbol{p} \mathbf{- C)^2 = R^2}$
  - Quadratic equation
  - 0,1,2 roots

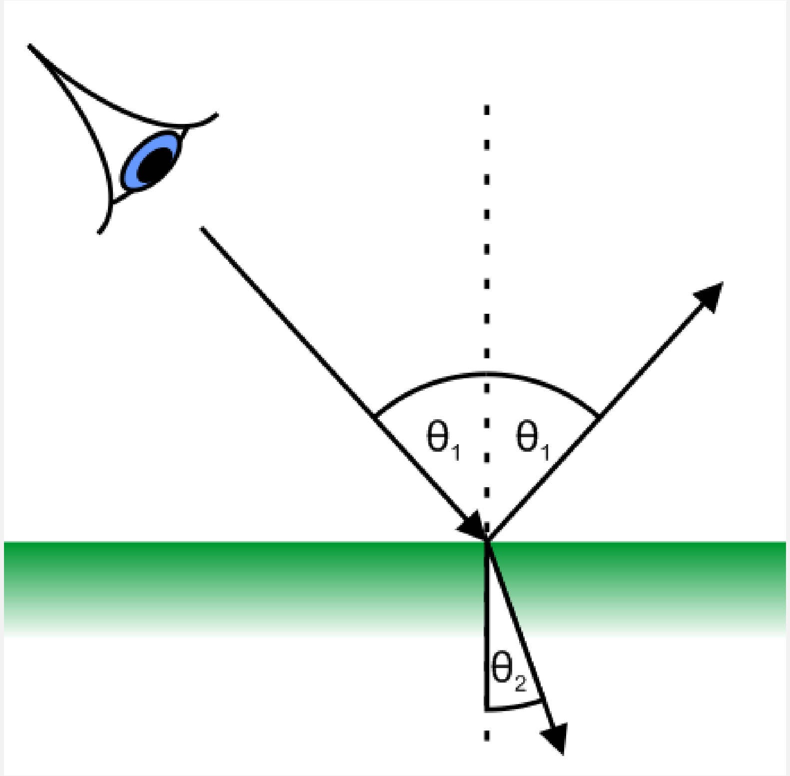


Line-triangle intersection

- Line P,*p* : **L = P + t * *p***
- Triangle K,L,M: **T = K + u*(L-K) + v*(M-K)**
- Line-plane intersection
  - Plane: **ax + by + cx + d = 0**
- Check if intersection is inside triangle

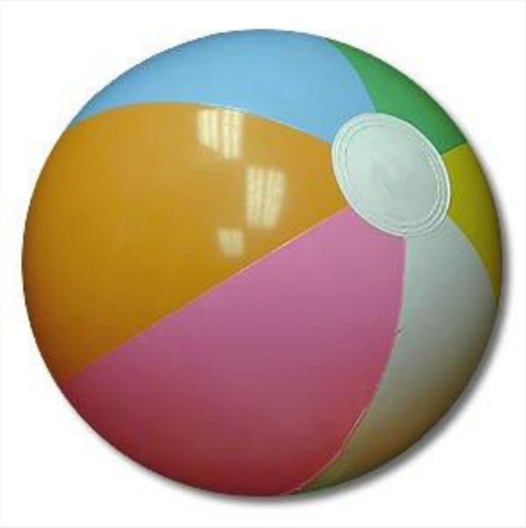
Raytracing – what's inside

- Compute reflected and refracted rays
  - evaluate light coming from their direction
- Combine with the local lighting result



$$C = comb(l.C_L, r.C_R, t.C_T)$$

Let's think the combinations



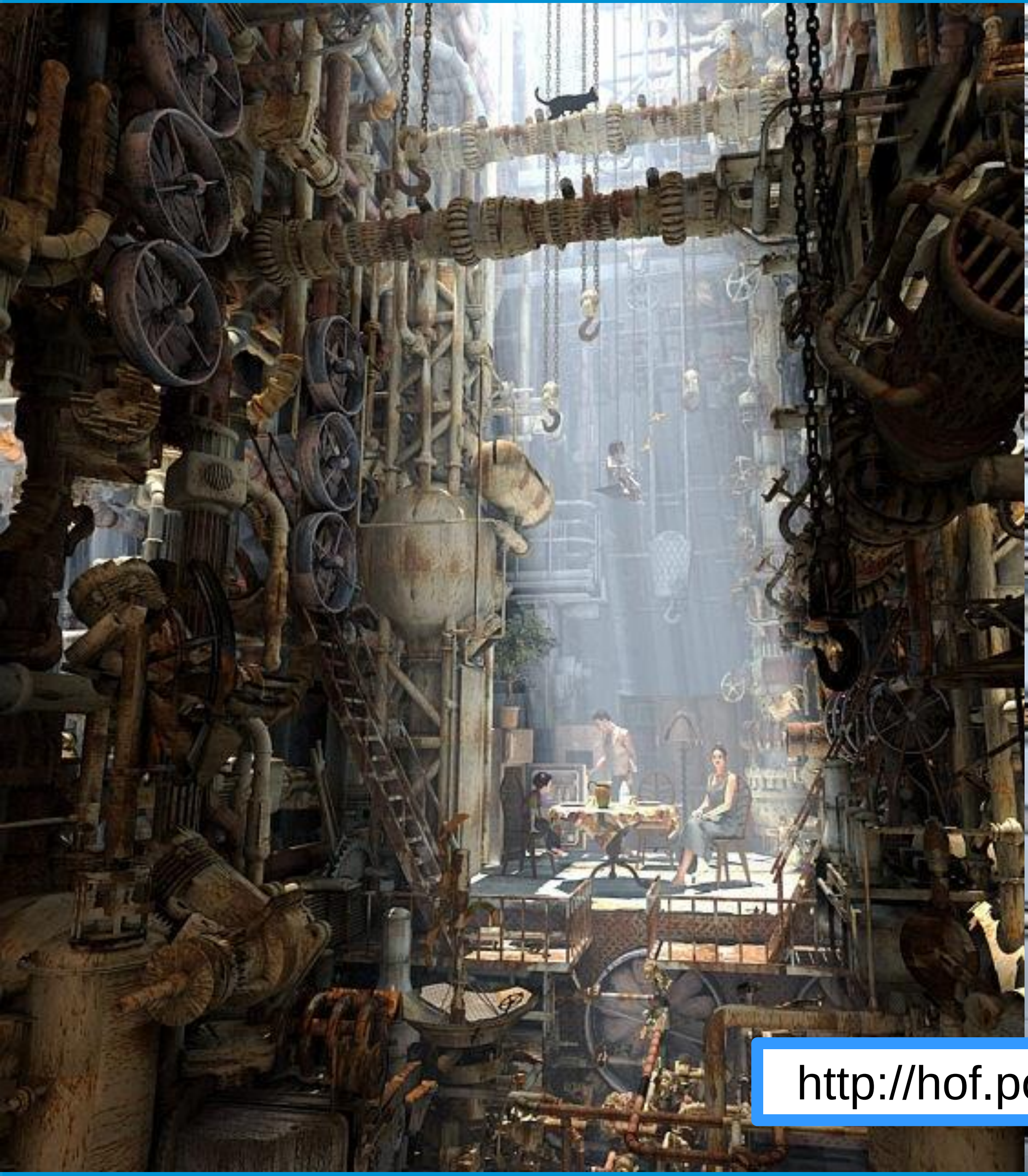
$$C = comb(l.C_L, r.C_R, t.C_T)$$

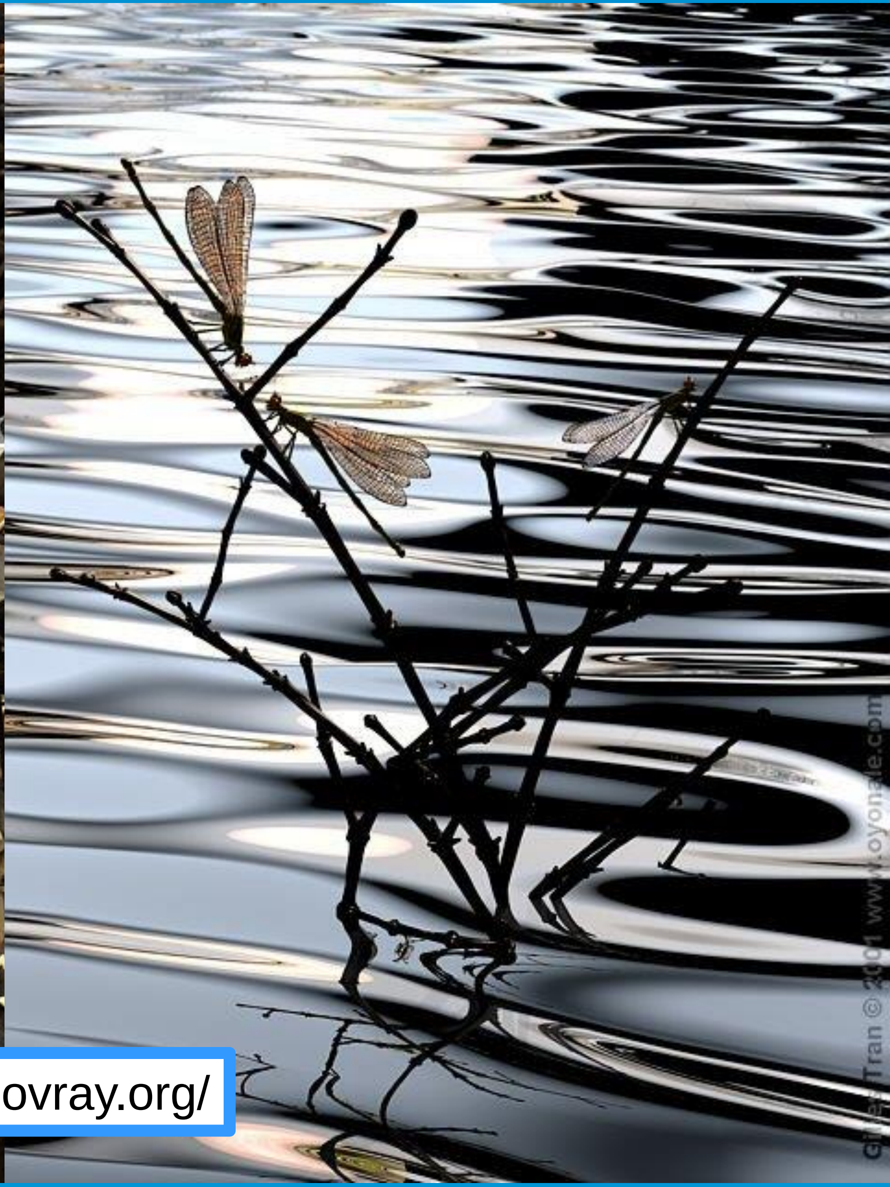
Raytracing – pros and cons

- No need for polygonal representation
  - works with both volume and boundary rep.
  - works with CSG objects, F-reps, meshes...
- No explicit rasterization takes place

- Computationally expensive
- Does not compute soft shadows

Examples: POVRAY



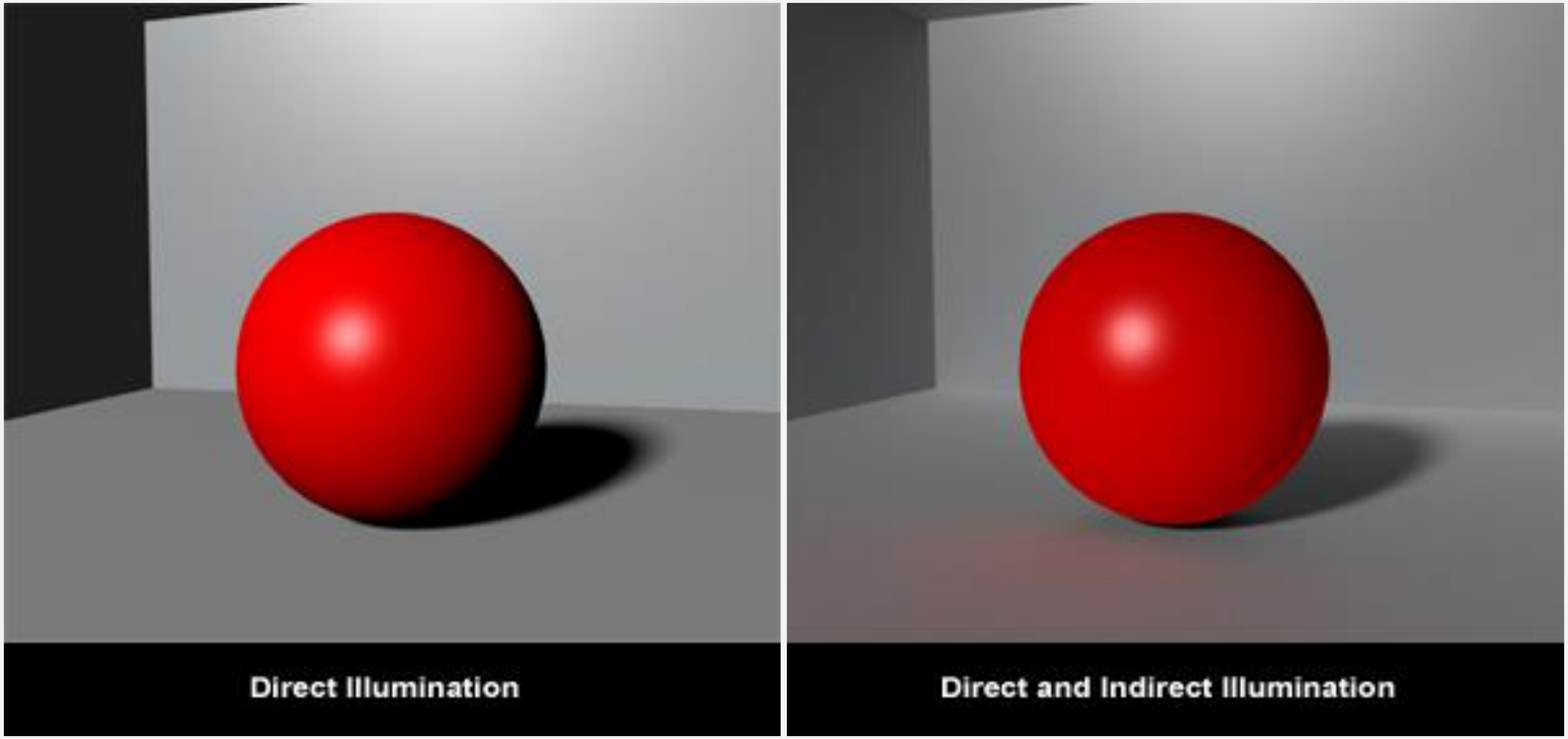


http://hof.povray.org/

Radiosity

- Object hit by light is a new light source
- Energy (light) exchange between objects
- Indirect illumination

http://www.bxhdesigns.com/



Real world radiosity

- Light reflectors in photography

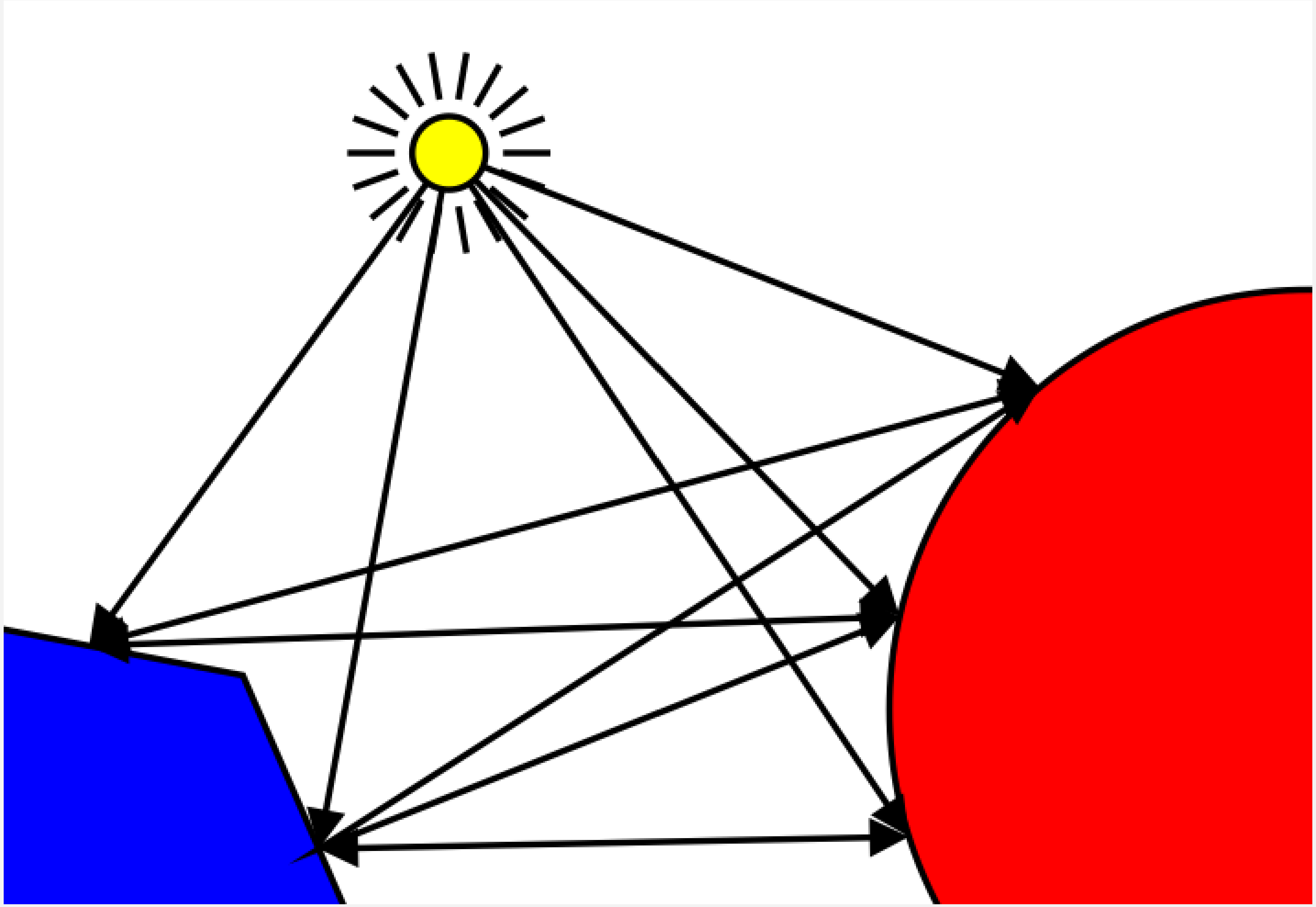
http://www.hootphotography.com

Radiosity

- Physically based
- Object hit by light becomes a new light source
- Not only object-light interaction
- But also object-object light interaction
- Energy exchange between objects

http://www.ehow.com/video_4938383.html

General situation

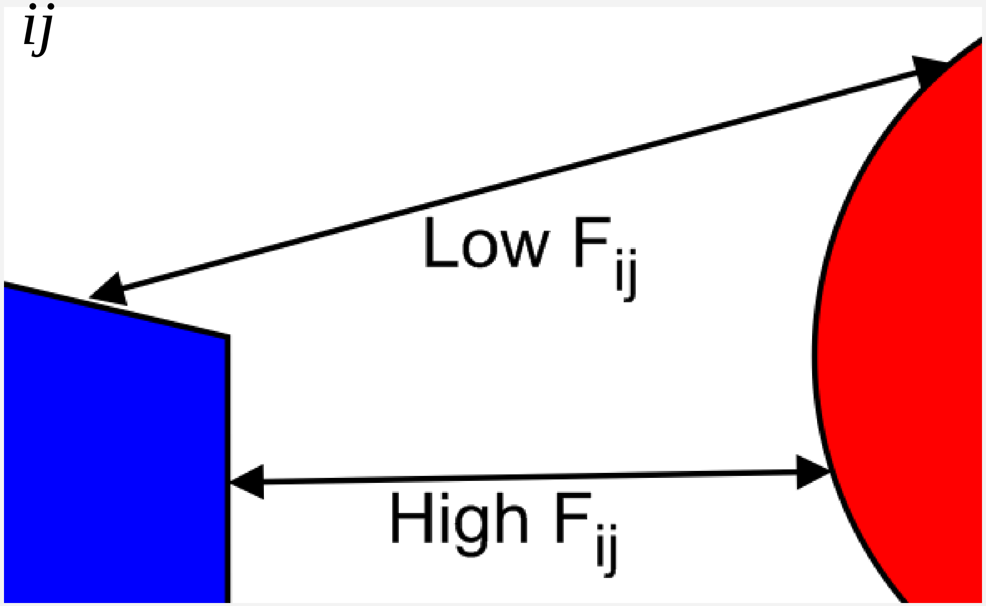


The math behind it

- Energy is either emitted (E) or bounced (B)
- All reflections are perfectly diffuse
- Surface A_i radiosity:

$$B_i = E_i + p_i \sum_{\forall A_j} B_j F_{ij}$$

- Form factors F_{ij}
  - how two surface elements A_i and A_j affect each other



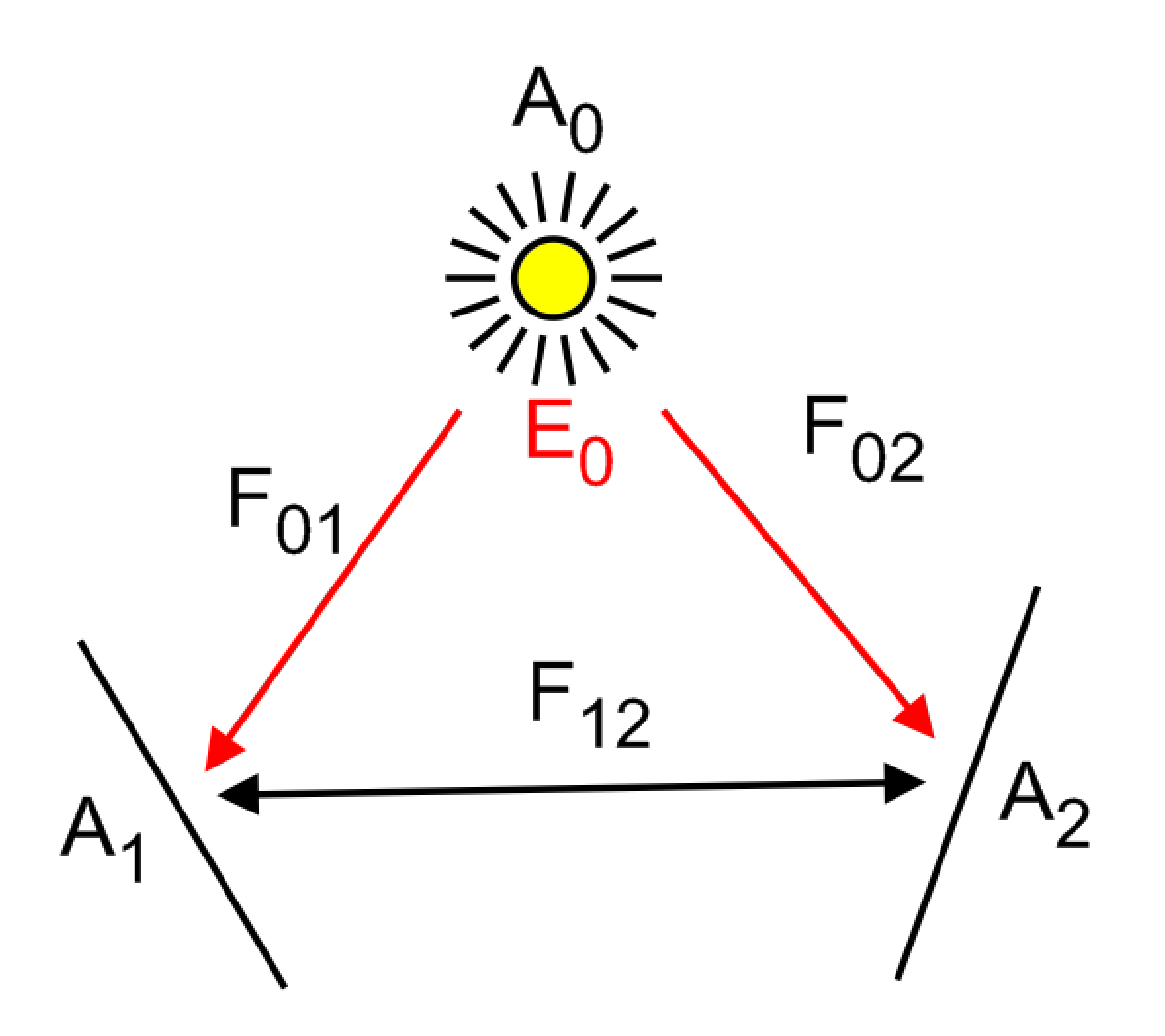
Elementary example

- Problem:

  $B_1 = ?$

  $B_2 = ?$

- Let E_1=E_2=0



Energy preservation

- Bounced = Emitted + p*Received

$$B_1 = E_1 + p_1(B_0F_{01} + B_1F_{11} + B_2F_{12})$$

$$B_1 - p_1(B_0F_{01} + B_1F_{11} + B_2F_{12}) = E_1$$

$$-p_1B_0F_{01} + B_1(1 - p_1F_{11}) - p_1B_2F_{12} = E_1$$

- Repeat for all 3 surfaces

Result = linear system

$$\begin{pmatrix} 1-p_0F_{00} & -p_0F_{01} & -p_0F_{02} \\ -p_1F_{10} & 1-p_1F_{11} & -p_1F_{12} \\ -p_2F_{20} & -p_2F_{21} & 1-p_2F_{22} \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} E_0 \\ E_1 \\ E_2 \end{pmatrix}$$

- $E_1 = E_2 = 0$
- E_0 = light source parameter

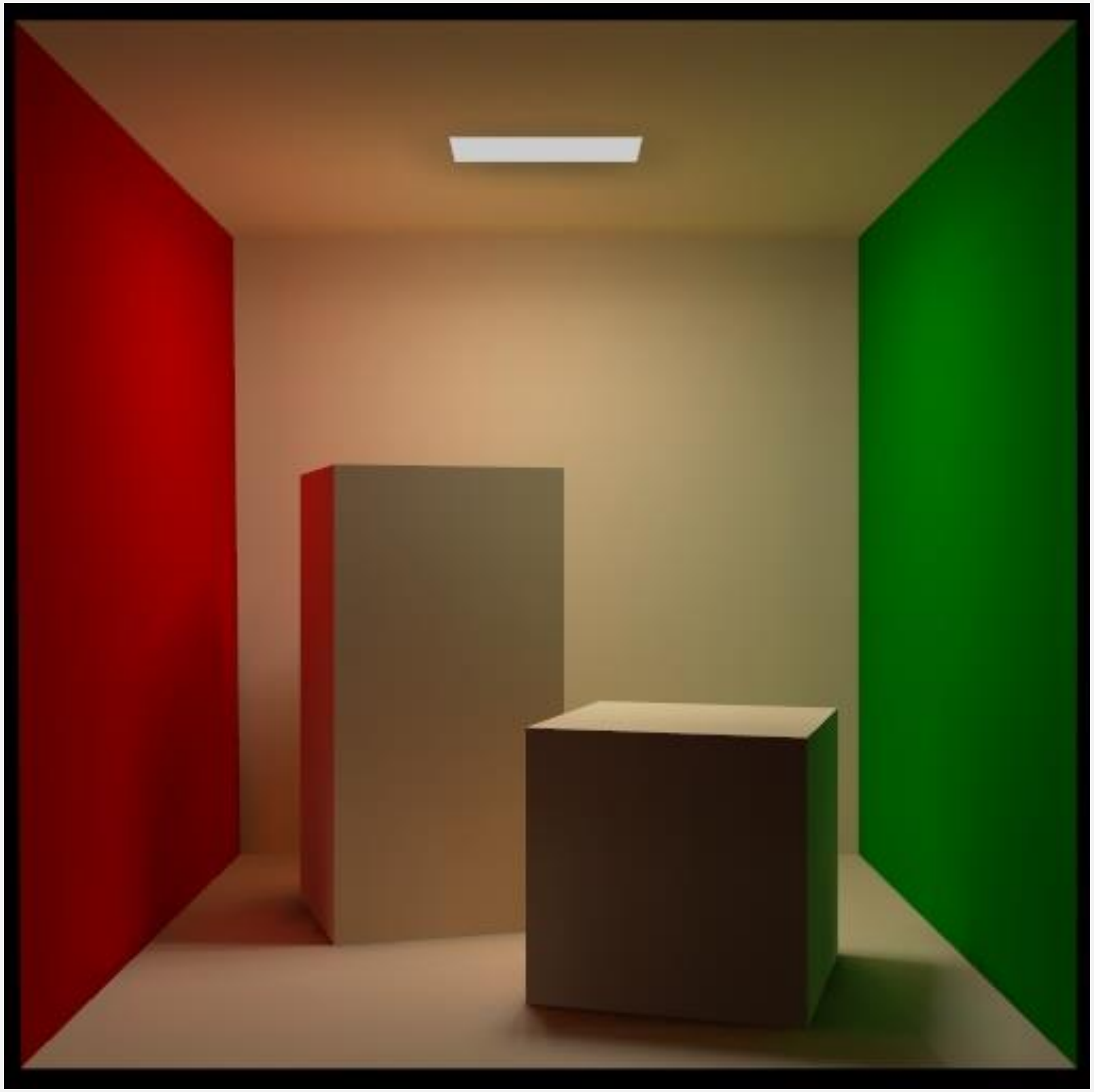
- In real – tens of thousands of surfaces

Radiosity pros and cons

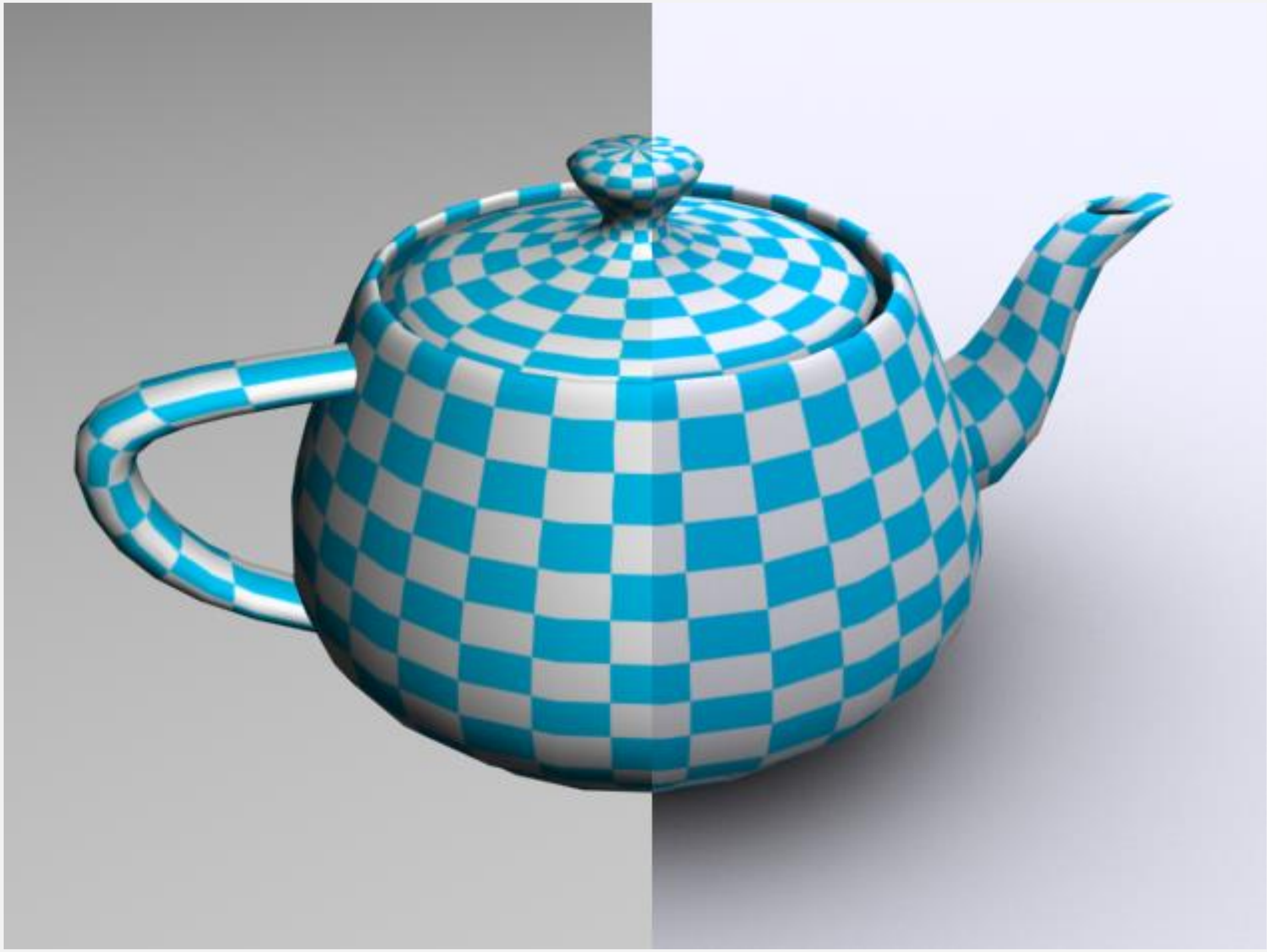
- Physically correct
- Extreme computational expenses
- Indirect light
  – soft realistic shadows
- Area lights and object lights are easy to do
- Color bleeding possible too
- Only diffuse light transfer = Problems with reflections and specular light

Example

- Indirect light
- Color bleeding
- Soft shadows
- Area light

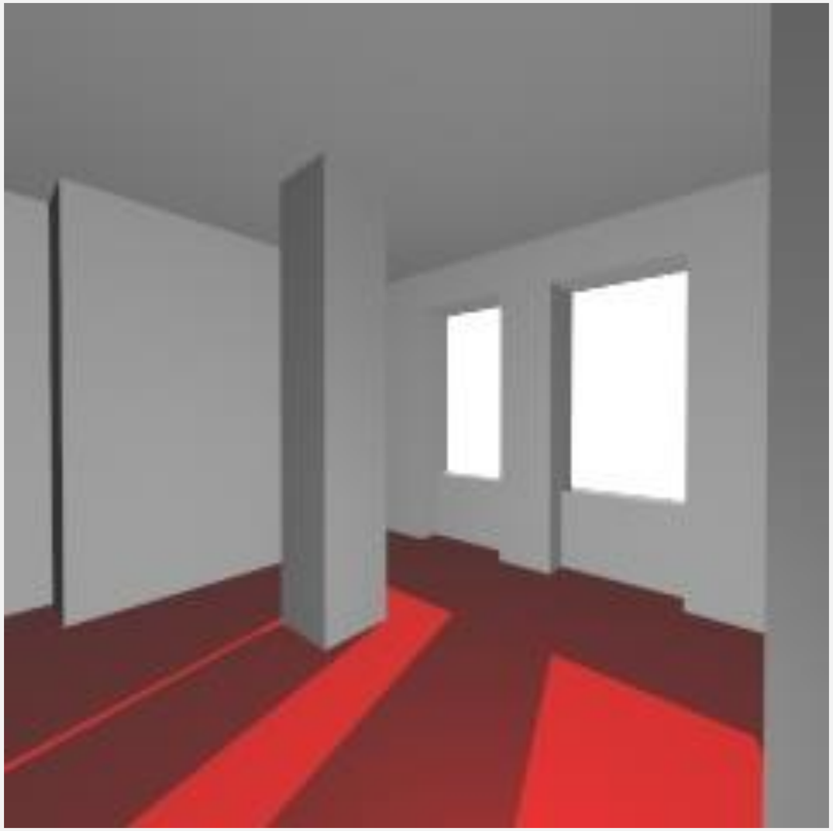


Radiosity example



direct illumination indirect illumination

Raytracing vs. radiosity

http://www.soe.ucsc.edu/classes/cmps161/Winter04/projects/aames/index.htm

Questions?