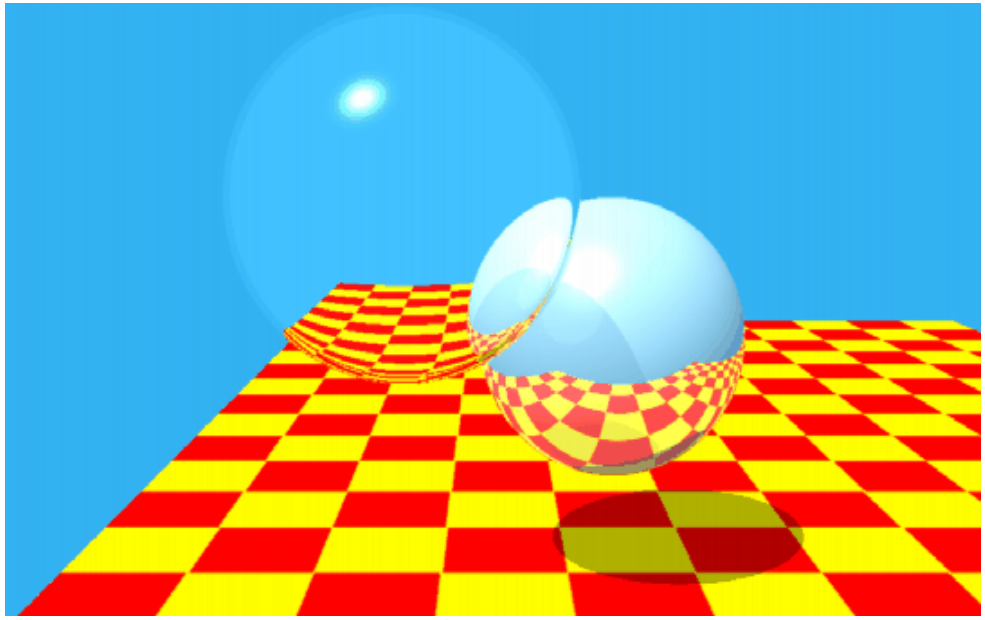
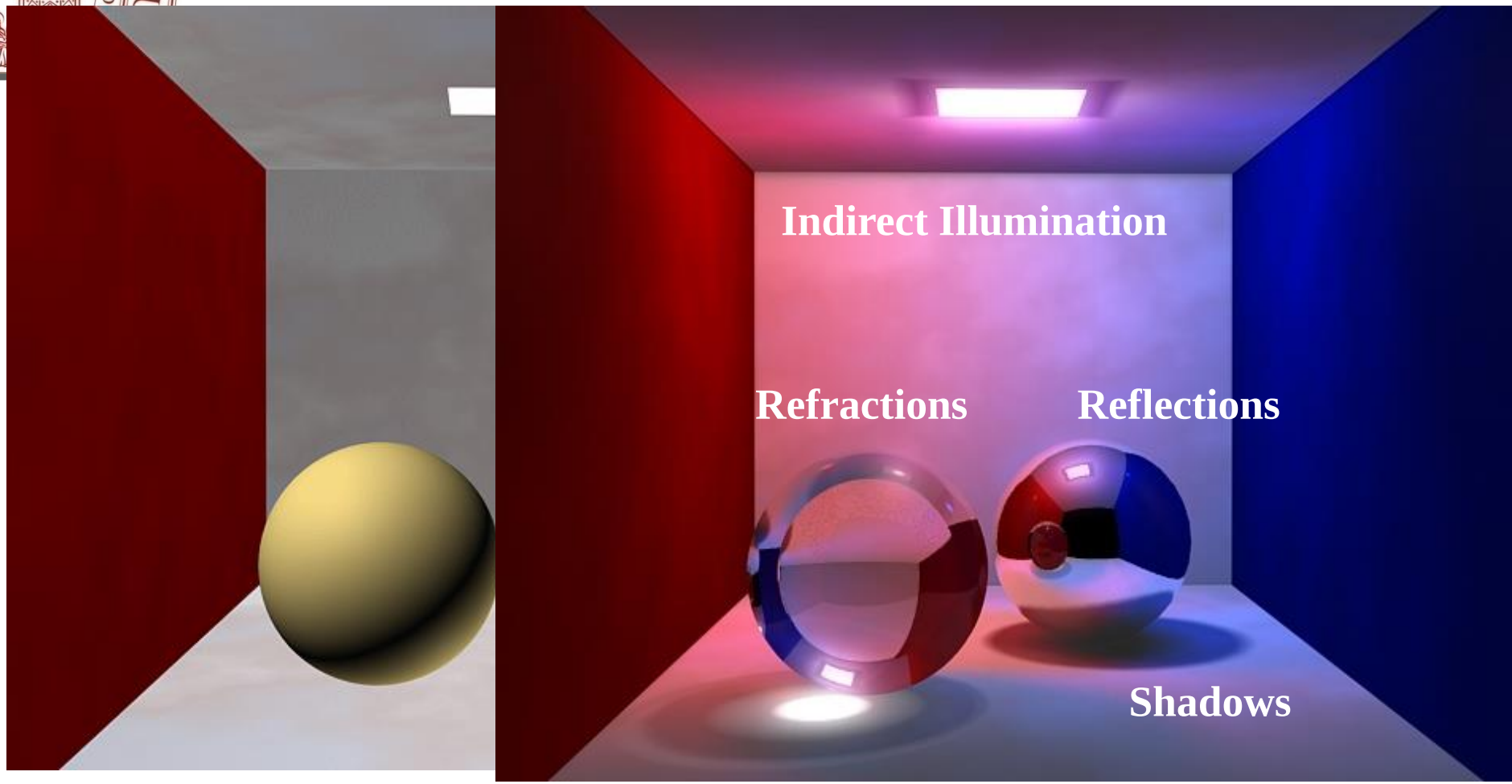
Global Illumination



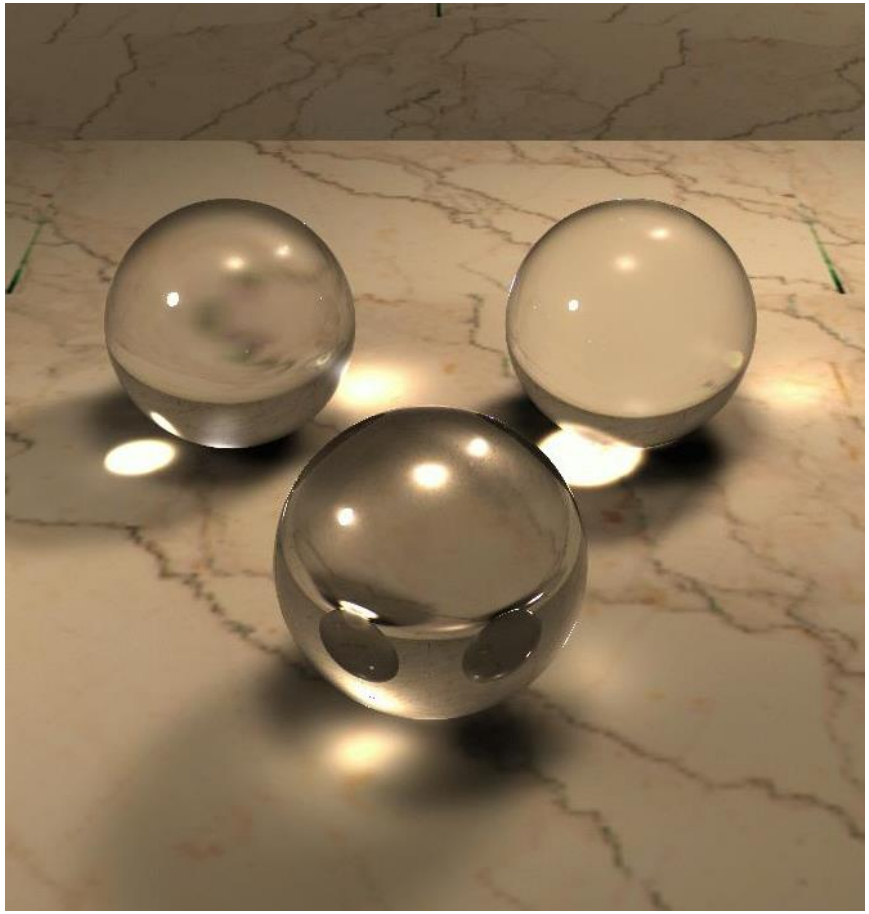
“An improved illumination model for shaded display” by T. Whitted, CACM 1980, Resolution 512x512 image, VAX 11/780, Time 74 min.

A ray traced image with recursive ray tracing, transparency and refractions

Color, depth and photorealism..



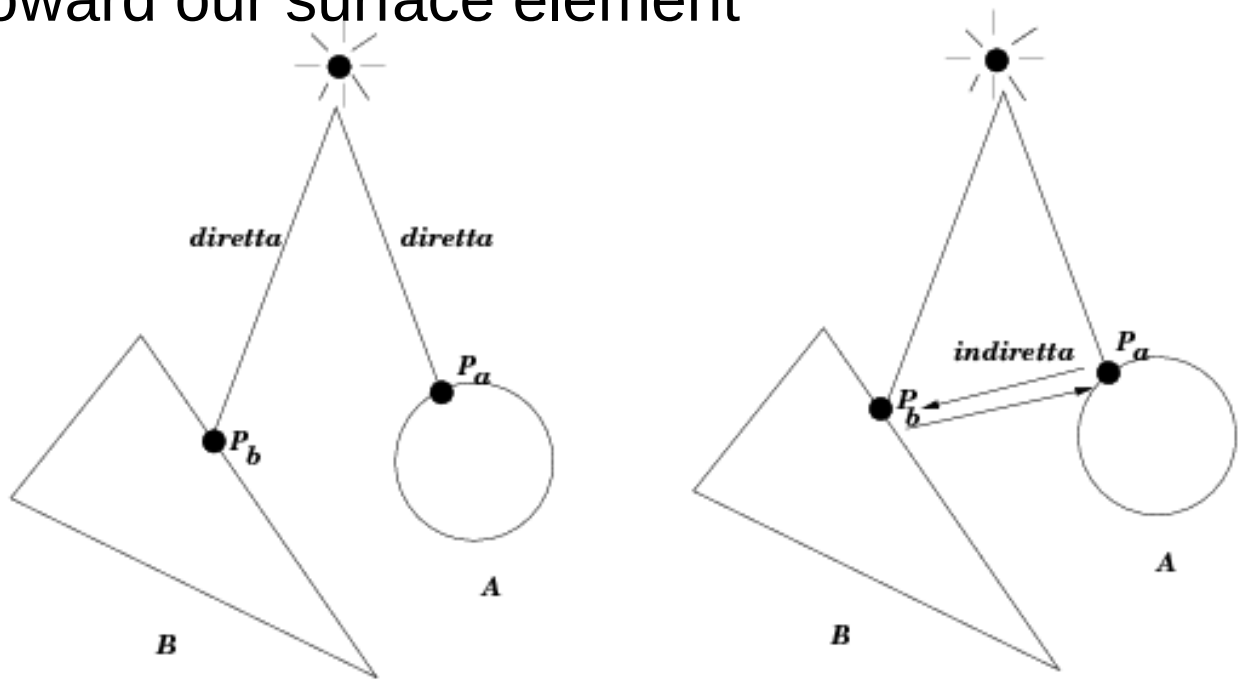
From the geometric model..... Toward its realistic visualization



The direct Illumination algorithm cannot generate images like these

Local and Global Models

- **Local model:** the color of a surface element depends only on light from light sources and ignores effects of all other objects in the scene
- **Global model:** the color of a surface element depends on the light from light sources and from light bounces off other objects toward our surface element



Global illumination models

Rendering Equation:

Physically-based illumination model which describe energy transport and radiation

Global illumination models:

- Ray Tracing

  Global specular interaction;

  Depends on the point of view

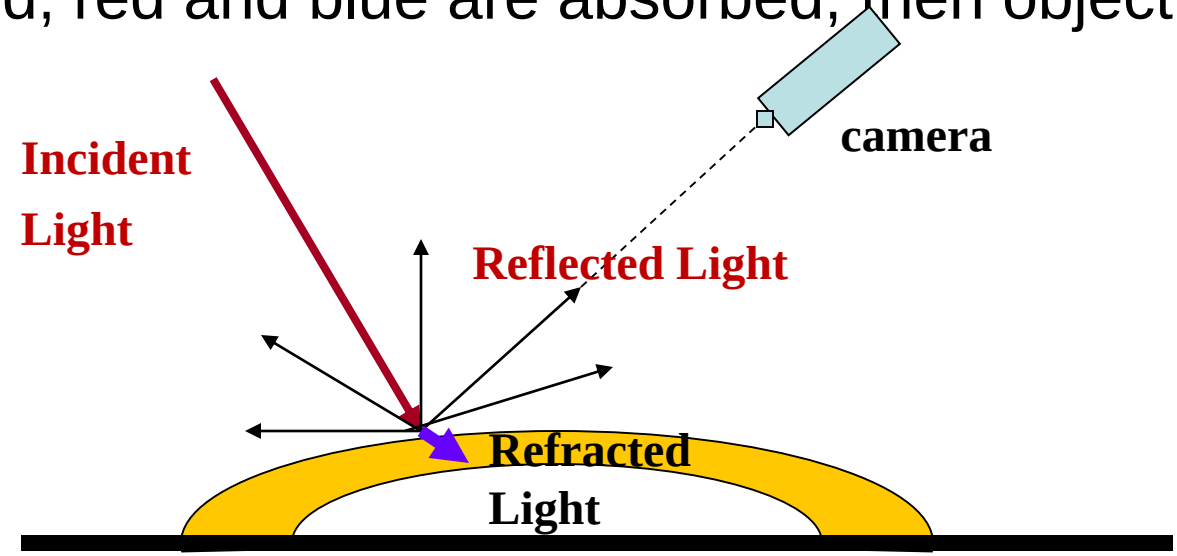
- Radiosity

  Global diffuse interaction;

  It does not depend on the point of view

Real World Lighting

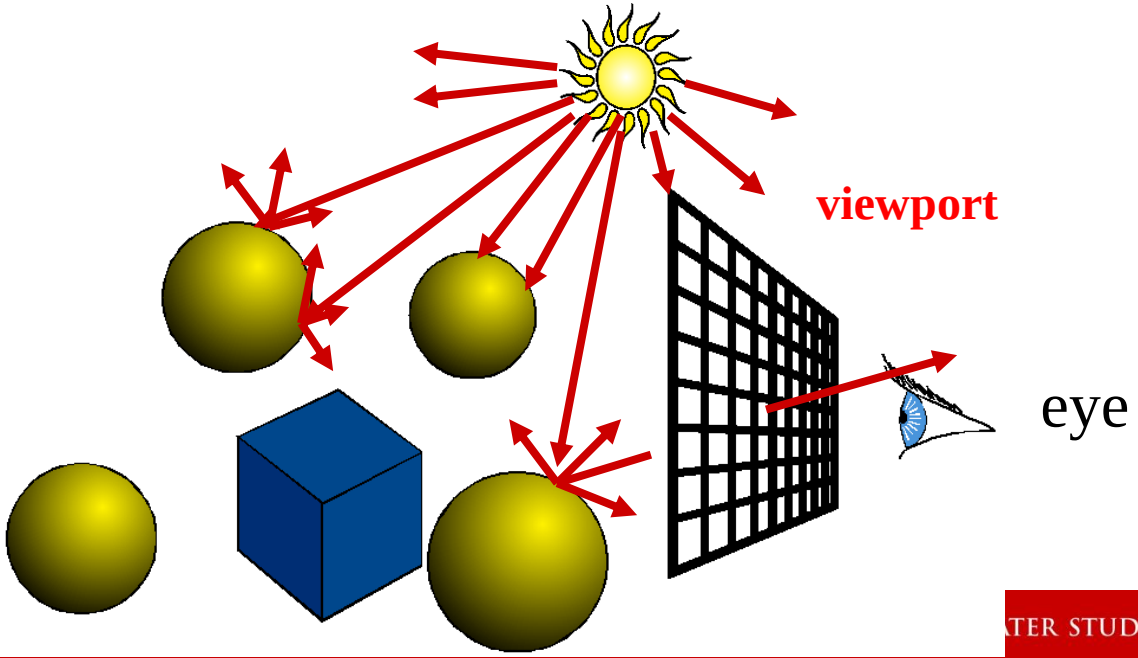
- Light sources emit photons.
- The incoming light from light sources is partially absorbed, partially reflected and partially refracted
- The reflected light defines the object color: if only green is reflected, red and blue are absorbed, then object appears green



- Reflectance – a fraction of the incident light that is reflected

Real lighting

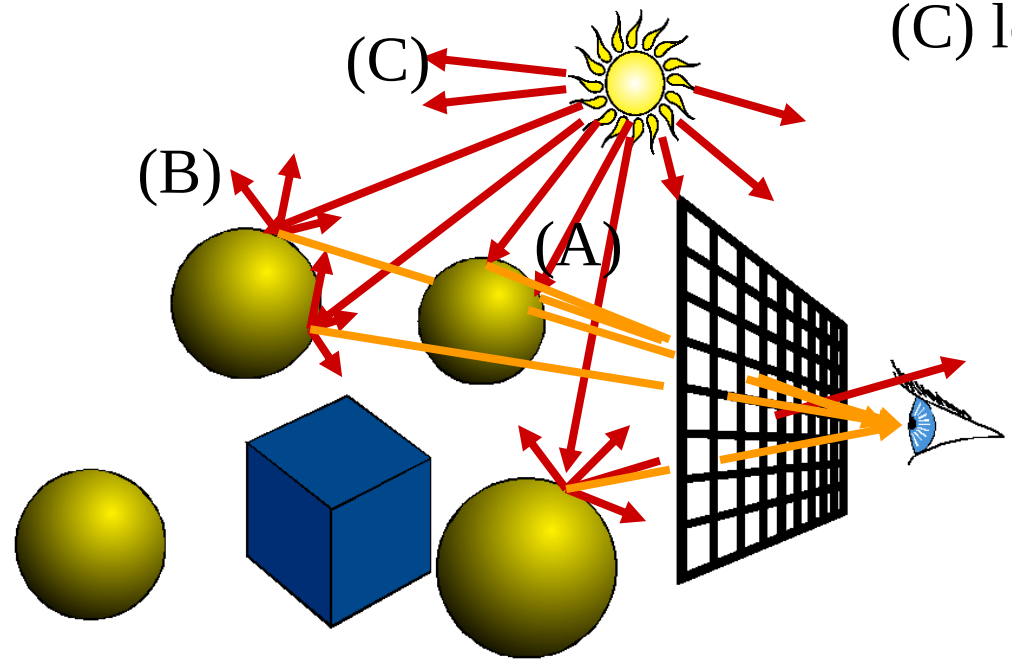
- Ray of light or photons go from the light source to the scene, and they travel until they eventually reach the viewer's eye position
- The color of an object point visible from the camera is given by
  - the light ray that directly hits it and it is reflected toward the camera,
  - and by indirect light rays bouncing from another object



Real lighting

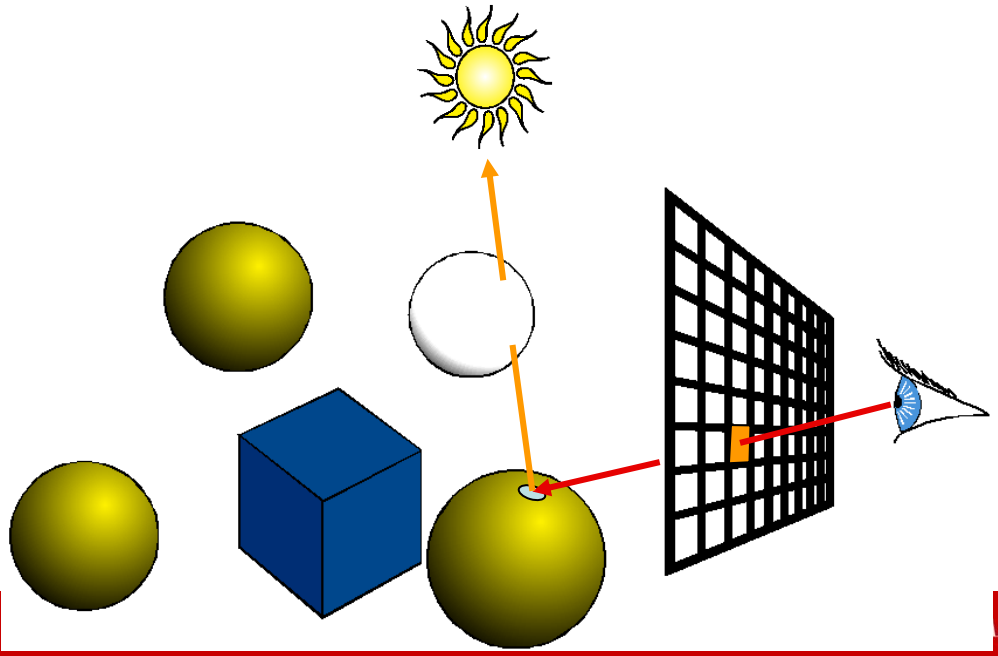
- Following the path of the light rays from the light source to the scene...
- Only a few of them reach the eye position (ray A)!
- Most light rays instead

(B) absorbed into a material
(C) leave the scene



Basic Ray tracing

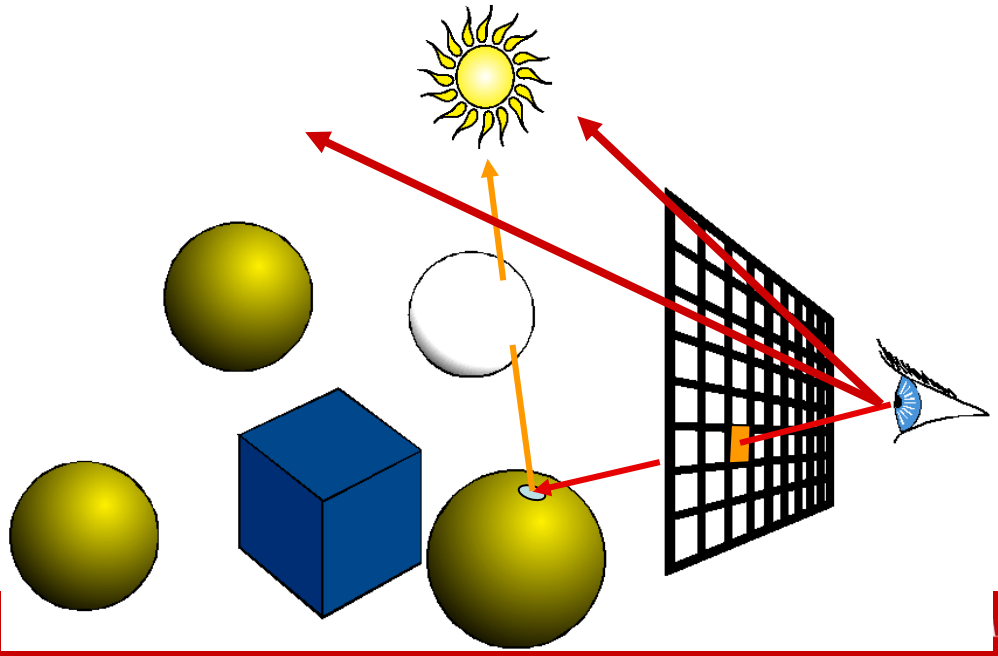
We are interested only into the rays which reach the eye, then instead of forward mapping infinite number of rays from light source to object to viewer, backmap finite number of rays from viewer through each sample to object to light source (or other object)



Basic Ray tracing

We must assign one color for each pixel,

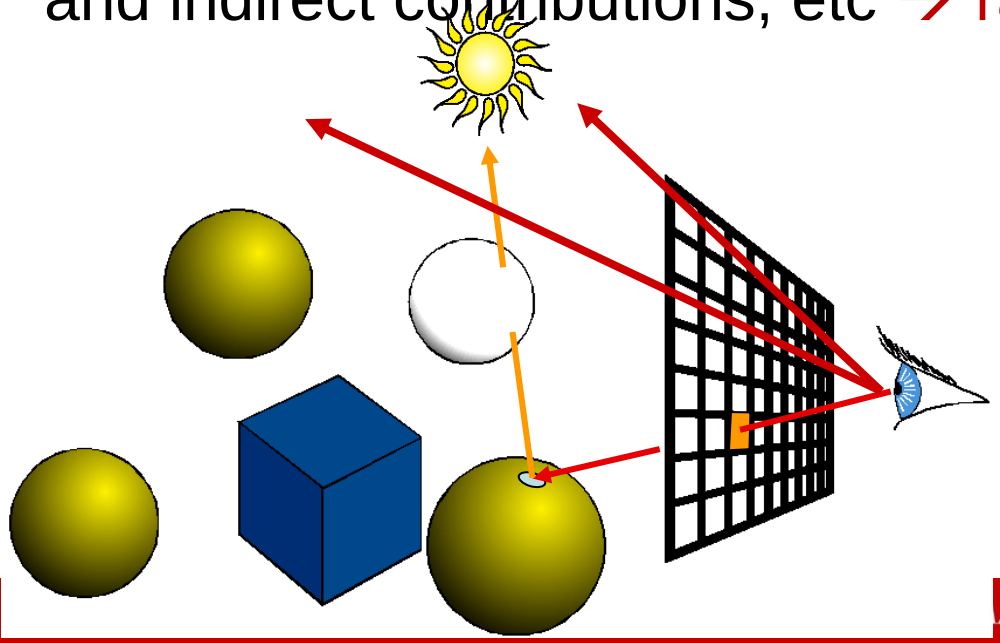
- Generate primary (‘eye’) ray
  - ray goes out from eye through a pixel center
- Find closest object along ray path
  - find first intersection between ray and an object in scene
  - It can be a **light source** or an **object**



If the ray doesn’t hit **anything**, then we can color the pixel to some specified ‘background’ color

Basic Ray tracing

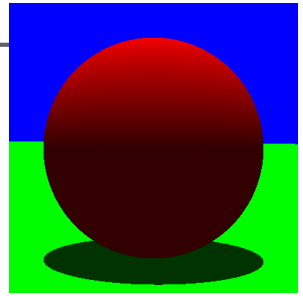
- Once we have the key intersection information,
- apply an illumination model to determine the direct contribution from light sources → ray casting
- For greater accuracy, recursively generate secondary rays to capture indirect contributions from inter-object reflections (specular components only) which themselves have direct and indirect contributions, etc → ray tracing

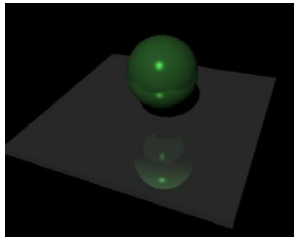


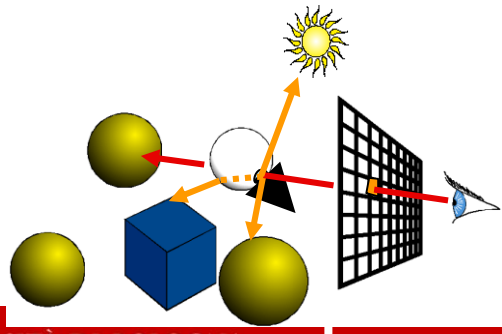
The result of the lighting equation is a color, which is used to color the pixel

Ray tracing

- Shadow/Direct Illumination (Ray Casting)
- Reflection
- Refraction
- Recursive Ray Tracing



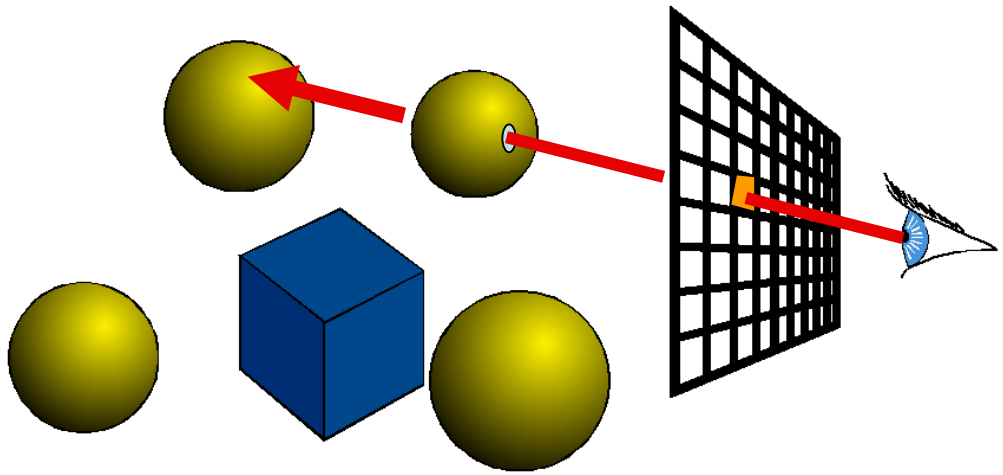




Ray Casting Algorithm

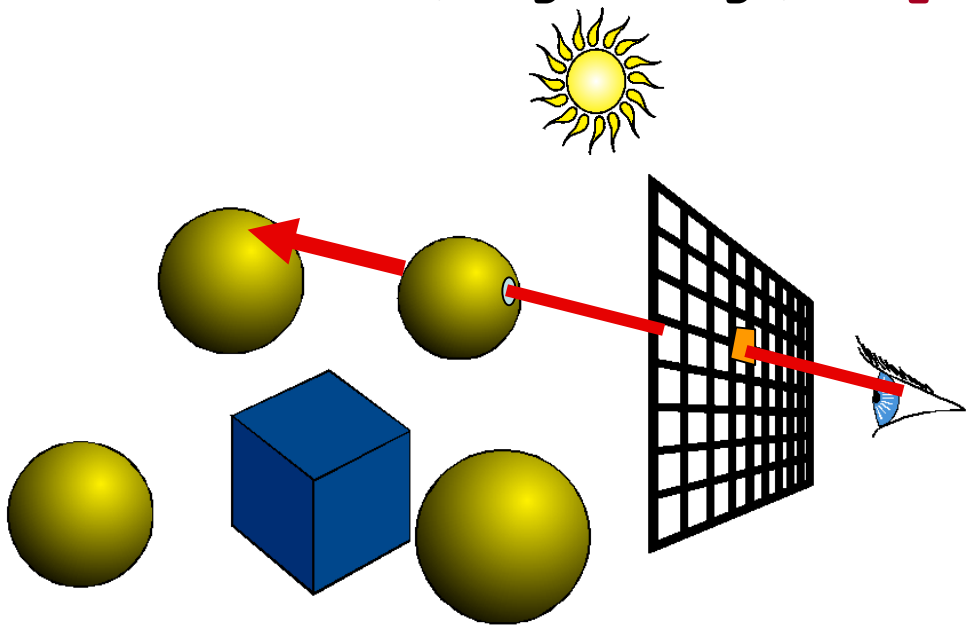
```
For every pixel (x,y)
   Construct a ray from the eye
   color[x,y]=castRay(ray)
```

- Complexity? O(n * m)
  - n: object number, m: pixel number



Ray Casting: Color castRay(ray)

```
color castRay(ray) {
    hitObject(ray, hitpoint, n ,object);
    color = object color;
    if(object is light)
        return(color);
    else
        return(lighting(hitpoint,n, color));
}
```



Shadow Rays

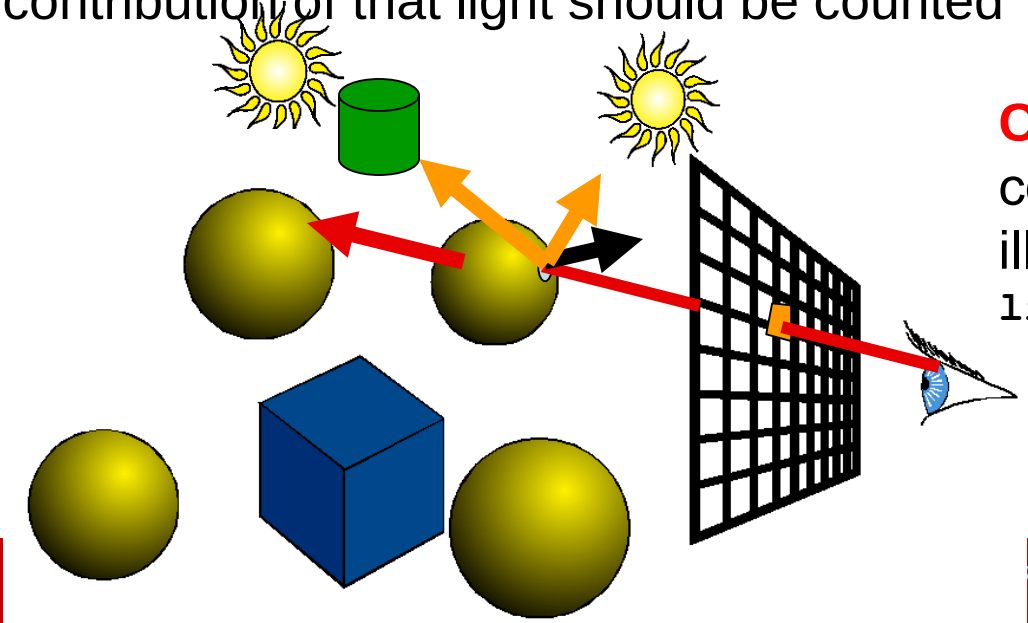
Each light in scene makes contribution to color and intensity of a surface element IFF it reaches the object!

- could be occluded by other objects in scene

Construct a ray from surface to each light, called **shadow ray,**

Check if the ray intersects other objects before it gets to light

- if ray does NOT intersect that same object or another object on its way to the light, then the full contribution of the light can be counted
- if ray **does** intersect an object on its way to the light, then **no** contribution of that light should be counted

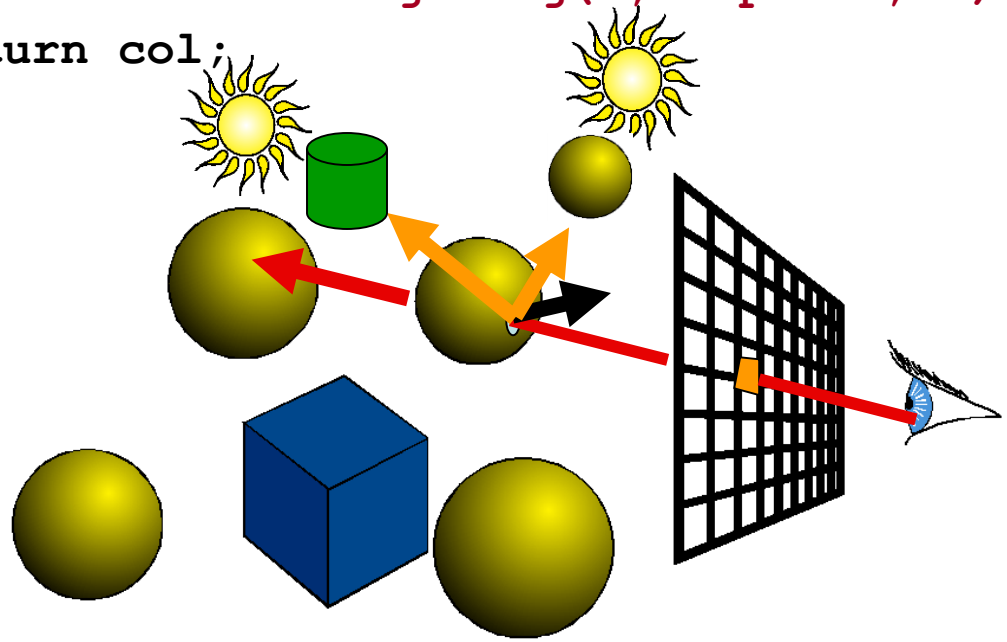


Color: compute the pixel color by applying the Phong's illumination model

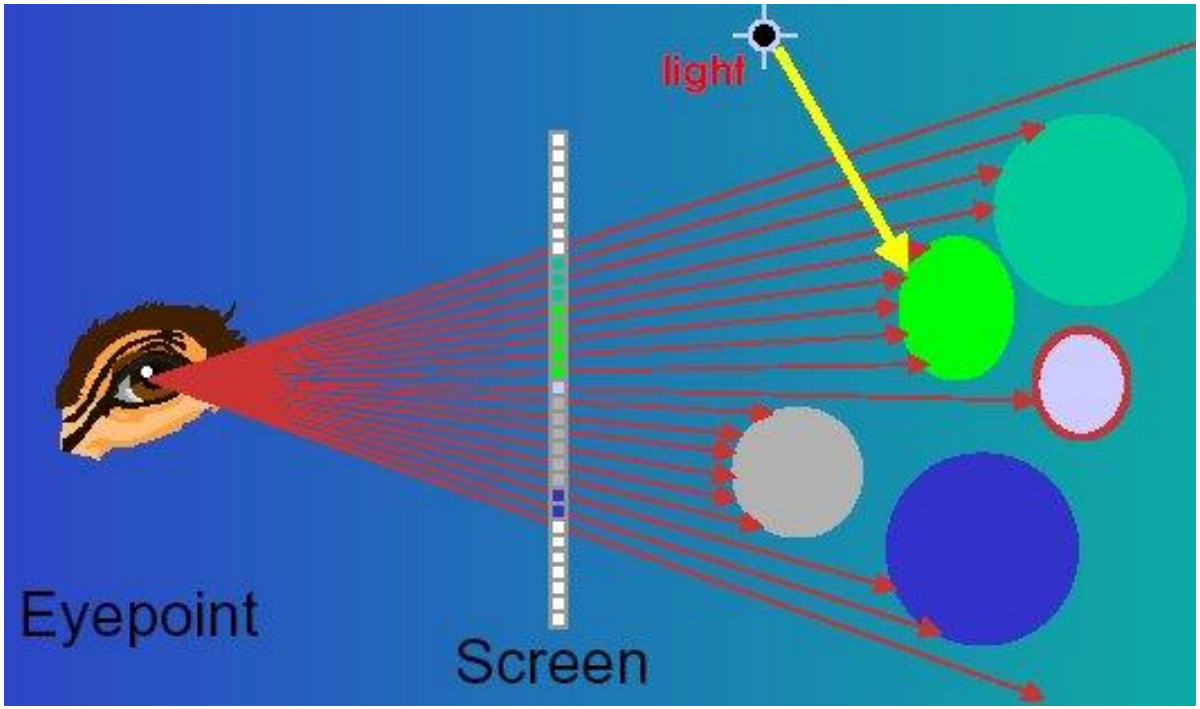
`lighting(hitpoint,n,color)`

Ray Casting: Color castRay(ray)

```
col castRay(ray ) {
    hitObject(ray, hitpoint,n, object);
     col = object color;
    For every light L
        Construct rayS(hitPoint, L->getDir());
        For every object ob
            hitObject(rayS, hit2, ob);
        If (rayS does NOT intersect any ob)
            col=col+lighting(L,hitpoint, n)
    Return col;
    }
```



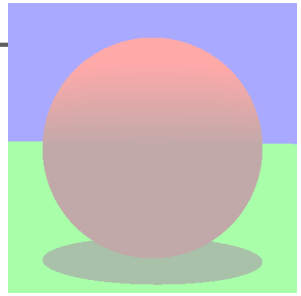
Ray casting

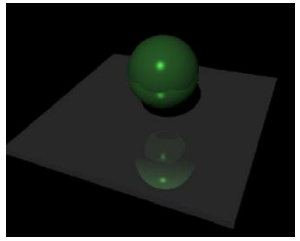


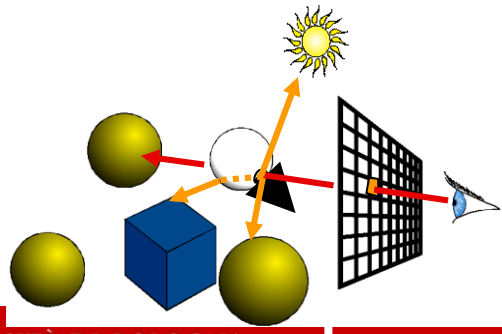
If no transparent objects are in scene, ray casting gets depth buffer hidden surface removal, shadows and local lighting illumination.

Ray tracing

- Shadow/Direct Illumination
- Reflection
- Refraction
- Recursive Ray Tracing





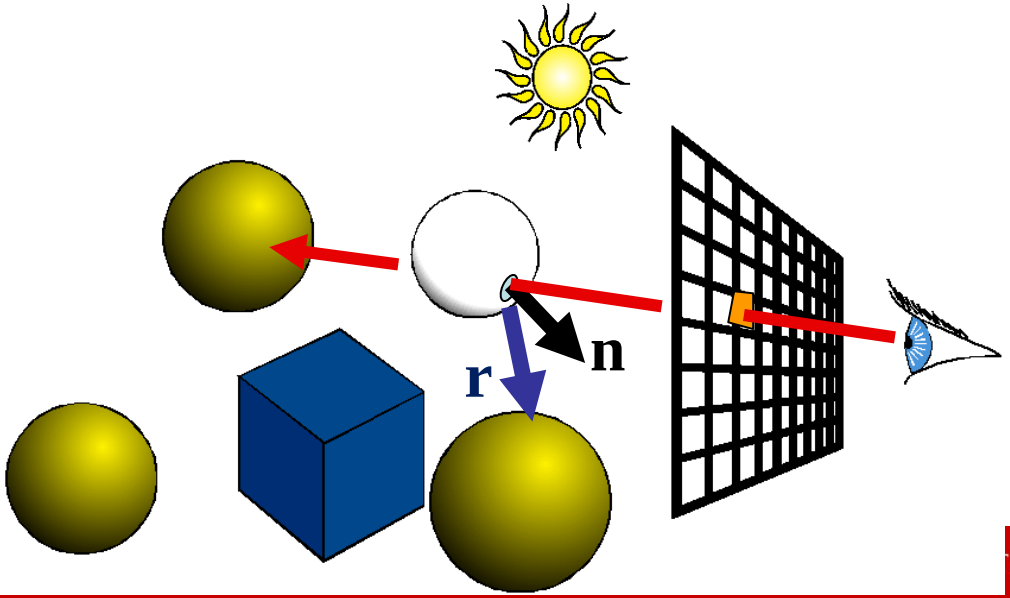


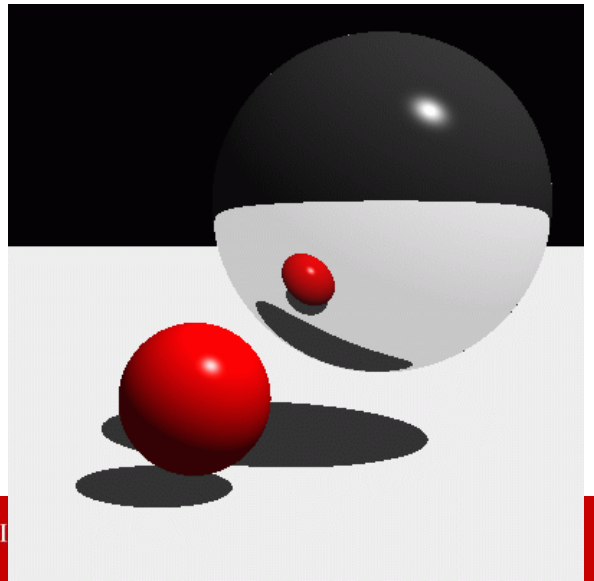
Specular Surfaces: Reflection Rays

So far accounts for light rays that originate from a point light, hit a surface, and then reflect from the surface to the eye

Light can also travel more complicated paths, perhaps bouncing multiple times from the surfaces before reaching the eye.

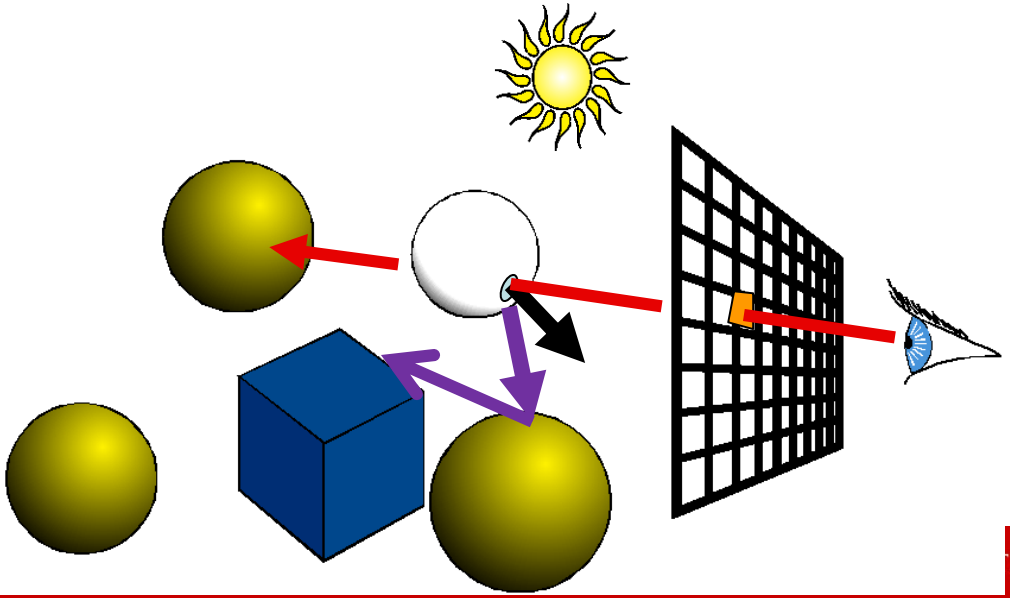
Add **reflection ray**: start from point of intersection and send a ray toward specular reflection

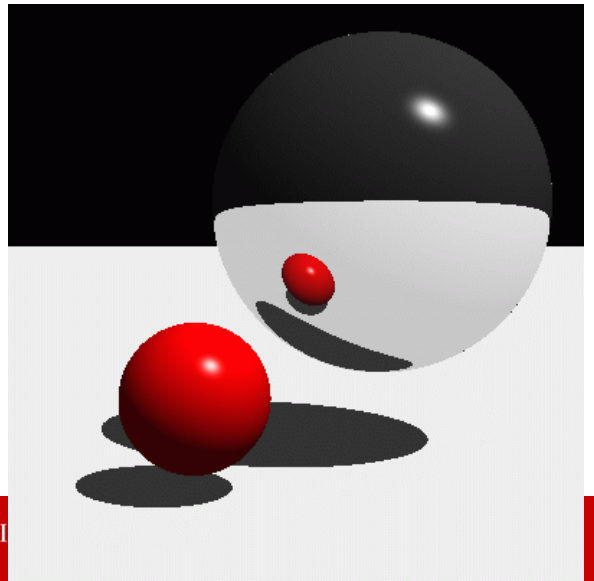




Specular Surfaces

- bounce off other objects to capture specular inter-object reflections, until they either go out or gets a light source
- find the first point where the reflected ray hits an object in the scene and calculate that point's illumination from all light sources (generate shadow rays and so on..)
- This process can continue recursively with reflection rays themselves spawning their own reflection rays.





Computing Reflection Direction

Compute $\mathbf{r_v}$, given the vectors **n**, **v**,

The incoming ray v, the surface normal n, and the reflected ray all lie in a common plane.

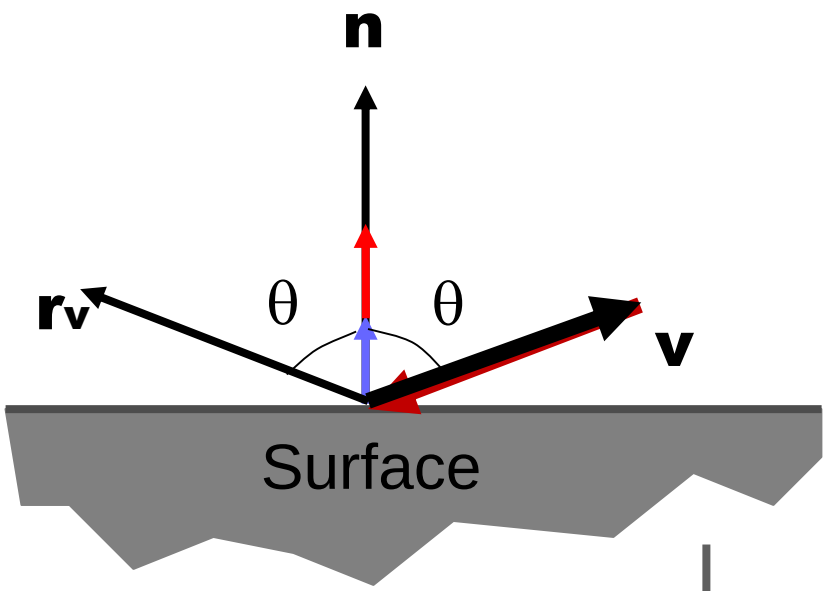
Incident angle = Reflected angle

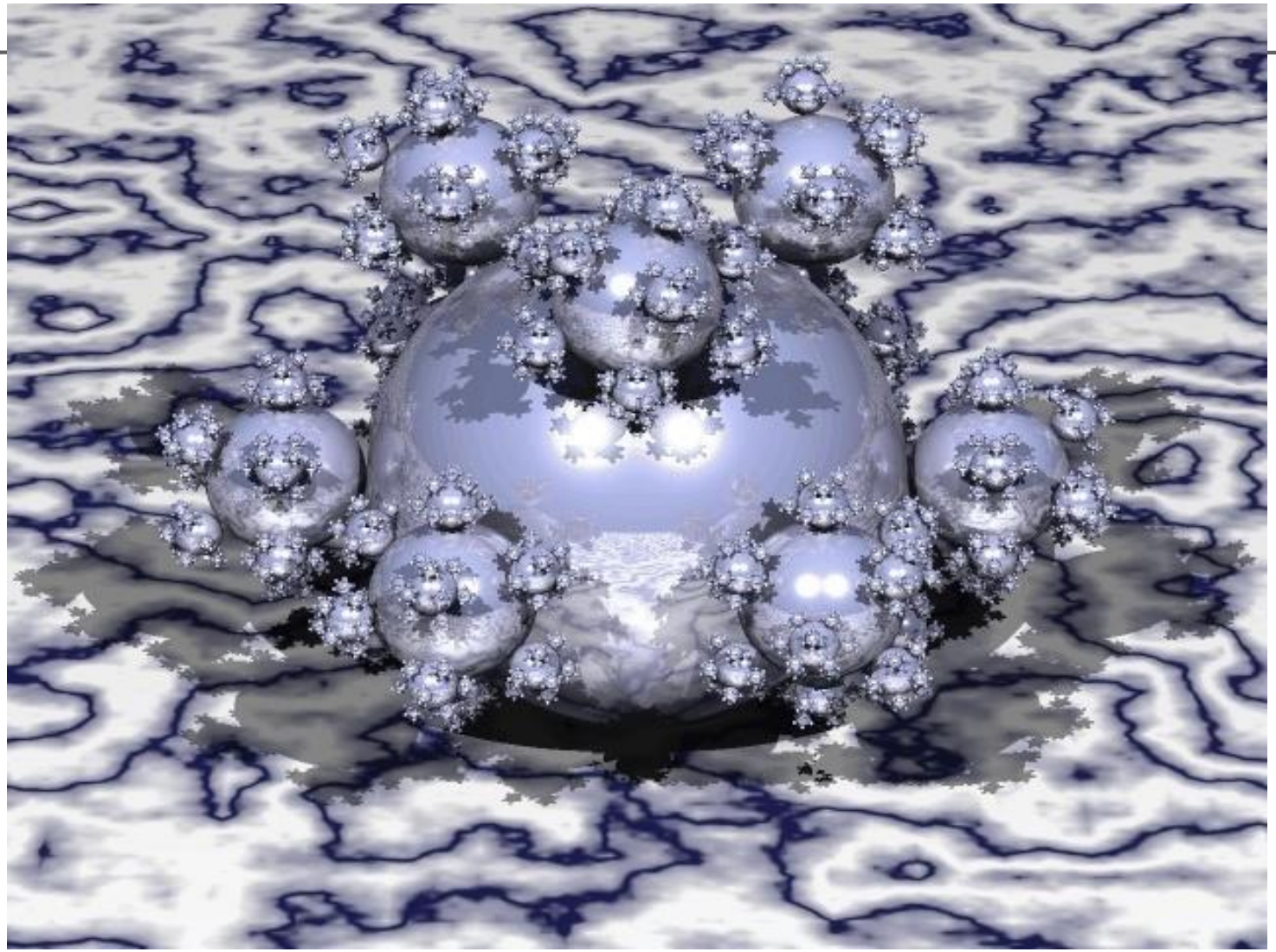
$$\mathbf{r}_v + \mathbf{v} = 2 \quad \cos\theta \ \mathbf{n}$$

$$\mathbf{r}_v = 2(\mathbf{n} \cdot \mathbf{v})\mathbf{n} - \mathbf{v}$$

v opposite

$$\mathbf{r}_v = \mathbf{v} - 2(\mathbf{n} \cdot \mathbf{v})\mathbf{n}$$

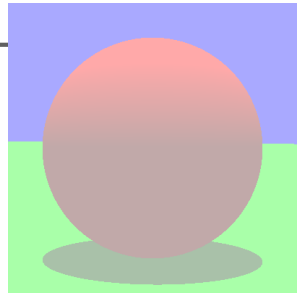


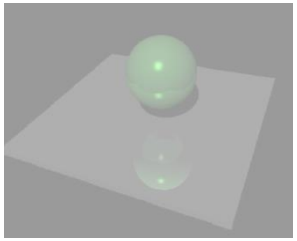


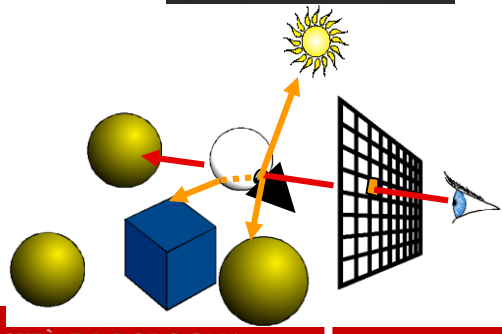
“Sphereflake” fractal **Henrik Wann Jensen**

Ray tracing

- Shadows/Direct Illumination
- Reflection
- Refraction
- Recursive Ray Tracing

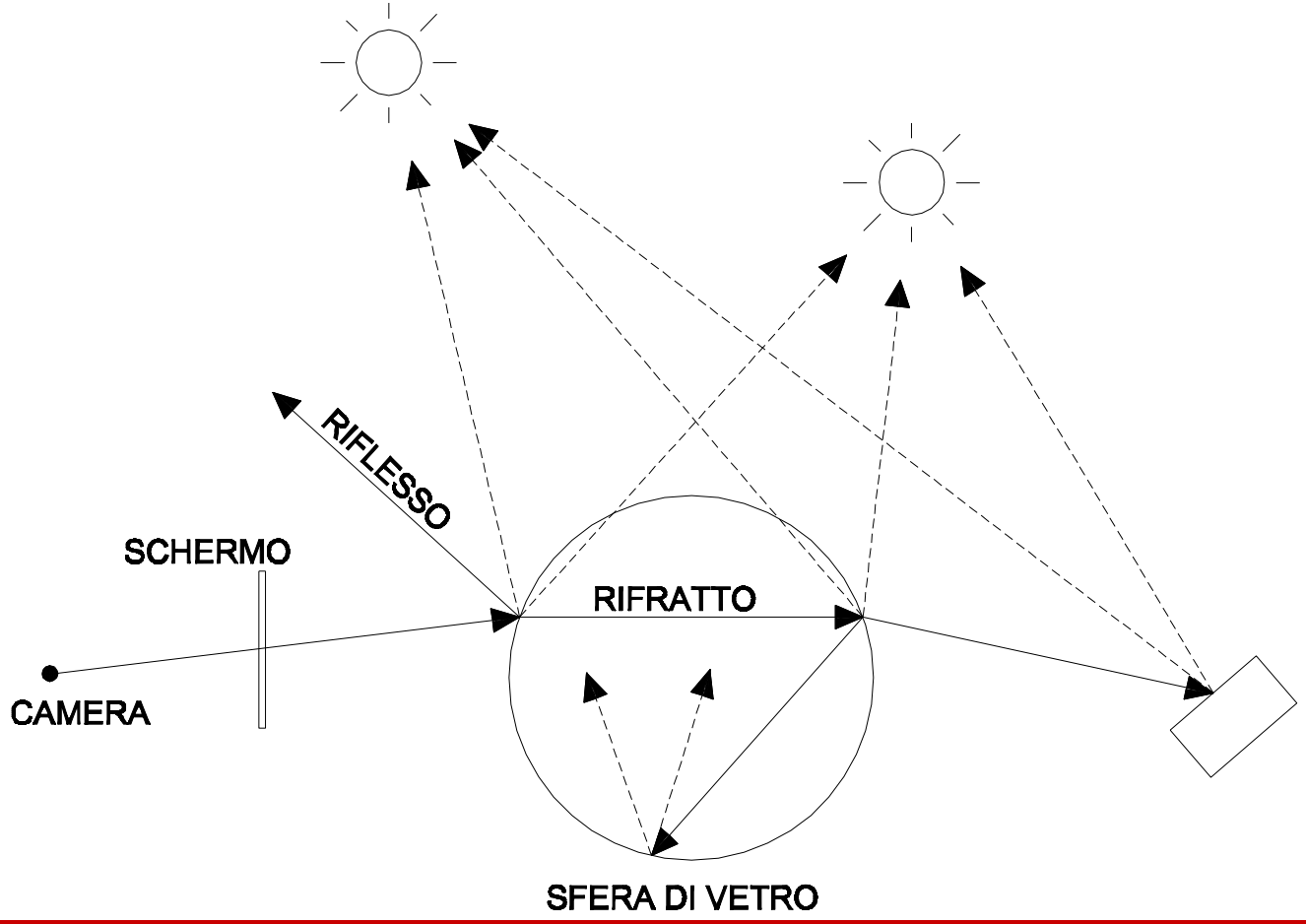






Transparecy

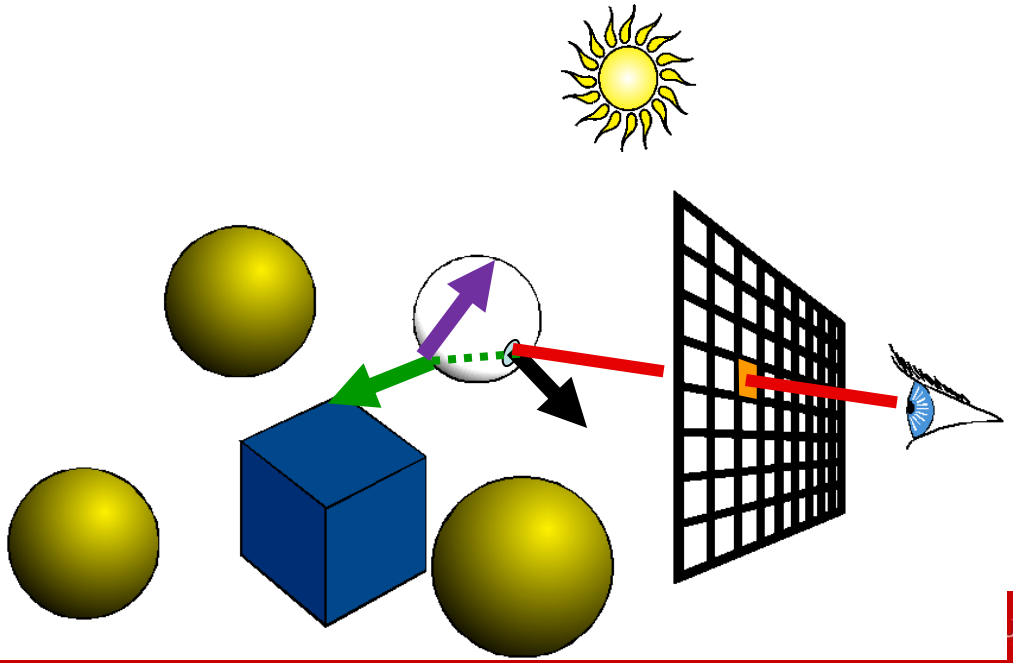
Ray tracing can also be used to accurately render the light bending in transparent surfaces due to refraction, when light passes from one medium to another



Transparency: Transmission Ray

A **transmission ray** is generated when a ray hits the surface of a transparent object: the transmission ray continues on through the surface. Refraction causes the direction of the transmitted ray to change.

The amount of refraction is calculated using the index of refractions.



At the first intersection point (inside) again a **reflection ray** and a **transmission ray** are generated This process continues recursively

Refraction is wavelength-dependent

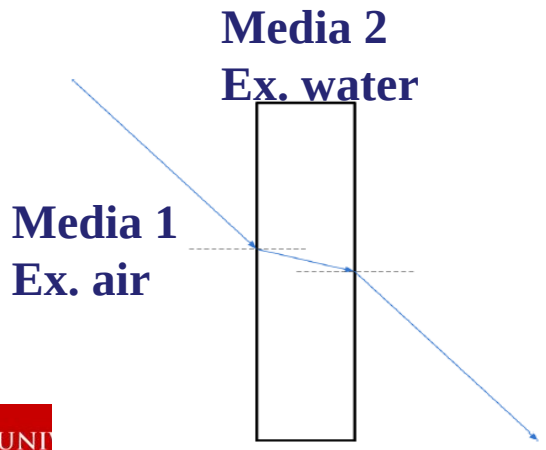
- A ray that hits the surface of a transparent object is refracted. This change in direction is caused physically by the difference in the speed of light in the two media (air and water, for instance)
- *Objects hidden by transparent objects appear deformed*
- Use of the index of refraction of the media (η)

Snell-Descartes Law

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{\eta_1}{\eta_2} = \eta_r$$

θ_1 angle of incidence

θ_2 angle of refraction/transmission



Transmitted Ray

D – incident ray direction

N – unit surface normal

R – refracted /transmitted ray direction

T – unit surface tangent

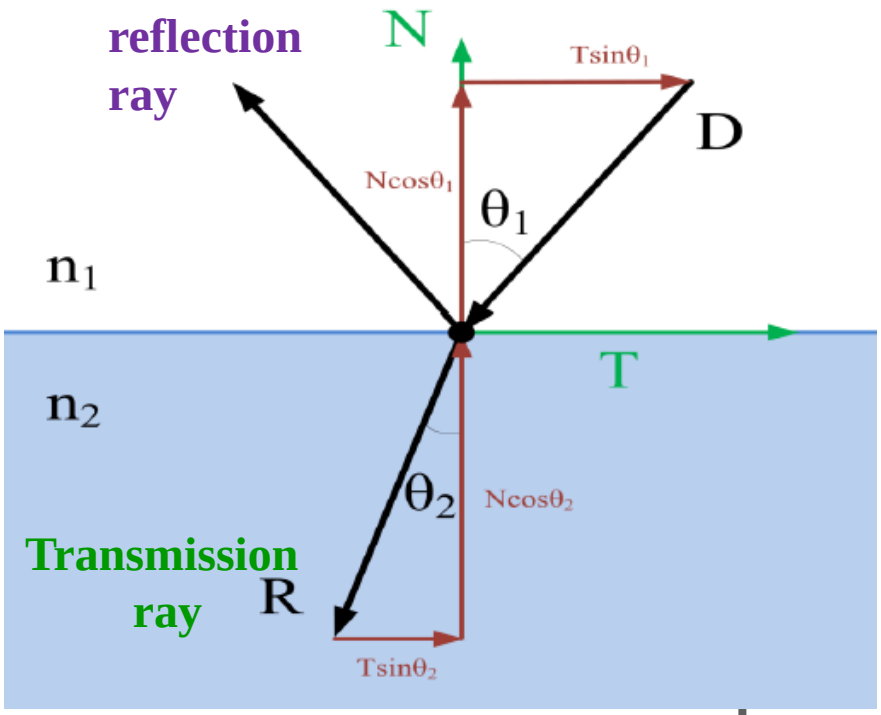
(in the plane N and I)

Snell-Descartes Law

$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$$

$$0 = T\sin\theta_1 + N\cos\theta_1 + D$$

$$0 = T\sin\theta_2 + N\cos\theta_2 + R$$



Transmitted Ray

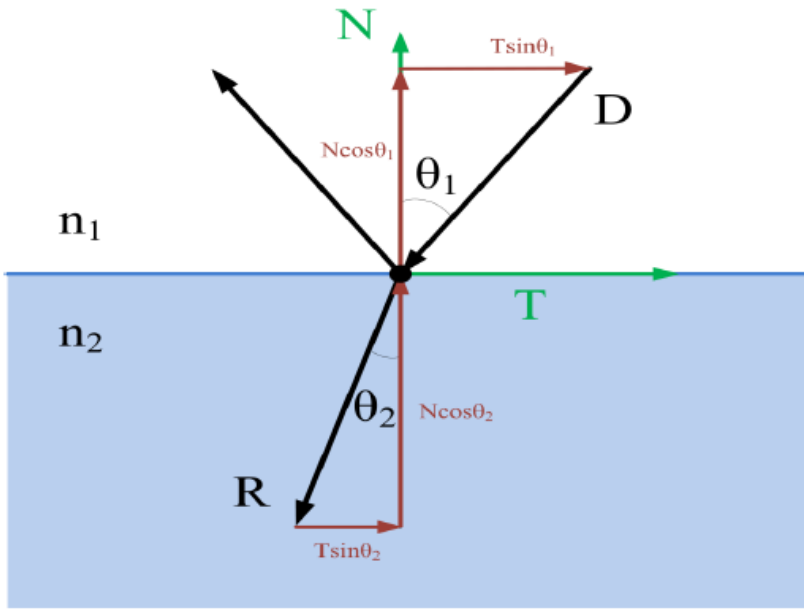
$$\boldsymbol{R} = -\boldsymbol{T}\sin\theta_2 - \boldsymbol{N}\cos\theta_2$$

$$= (\boldsymbol{D} + \boldsymbol{N}\cos\theta_1)\frac{\sin\theta_2}{\sin\theta_1} - \boldsymbol{N}\sqrt{1-\sin^2\theta_2}$$

$$= (\boldsymbol{D} + \boldsymbol{N}\cos\theta_1)\frac{n_1}{n_2} - \boldsymbol{N}\sqrt{1-\left(\frac{n_1}{n_2}\sin\theta_1\right)^2}$$

$$= \boldsymbol{D}\frac{n_1}{n_2} + \boldsymbol{N}\left(\frac{n_1}{n_2}\cos\theta_1 - \sqrt{1-\frac{n_1^2}{n_2^2}(1-\cos^2\theta_1)}\right)$$

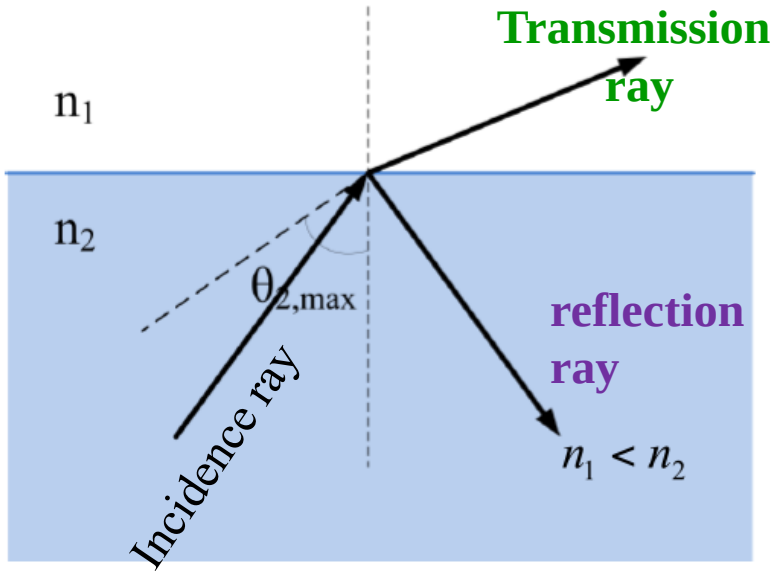
$$= \boldsymbol{D}\frac{n_1}{n_2} - \boldsymbol{N}\left(\frac{n_1}{n_2}\boldsymbol{D}\bullet\boldsymbol{N} + \sqrt{1-\frac{n_1^2}{n_2^2}(1-(\boldsymbol{D}\bullet\boldsymbol{N})^2)}\right)$$



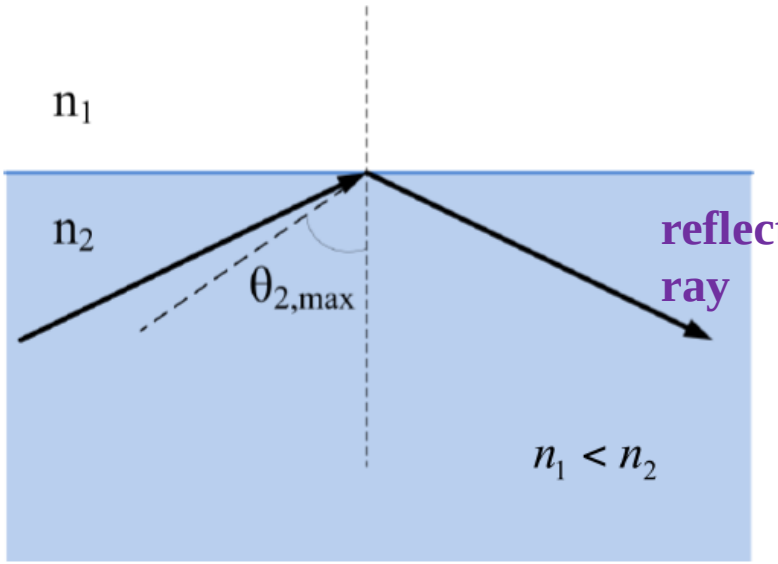
When the square root is negative, there is no transmitted ray and all the light is reflected (Total internal reflection)

Total internal reflection

- Light goes from a material with a higher refraction index n_2 to a material with a lower refraction index n_1, there is no refraction if the incident angle exceeds a critical angle
- Then all the light reflects



when $\theta_2 < \theta_{2,max}$, both refraction/transmission and reflection

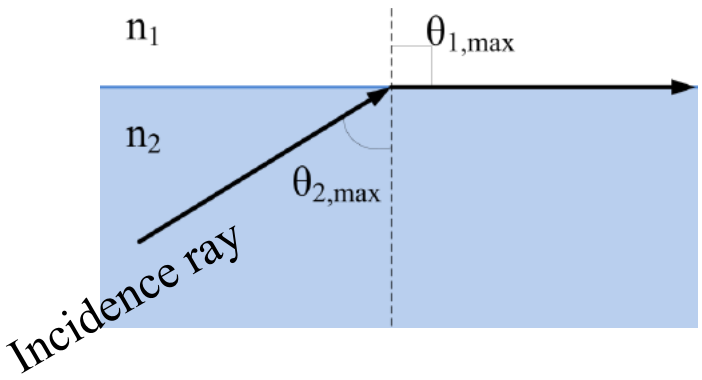


when $\theta_2 > \theta_{2,max}$, only reflection

Total internal reflection

- Light goes from a material with a higher refraction index n_2 to a material with a lower refraction index n_1, there is no refraction if the incident angle exceeds a critical angle
- Then all the light reflects

$$\theta_{1,\max} = \frac{\pi}{2}$$



Viewing angle

- The proportion of reflection versus transmission gradually increases as we go from a perpendicular to a parallel viewing direction

Perpendicular view:
high transmission, low reflection

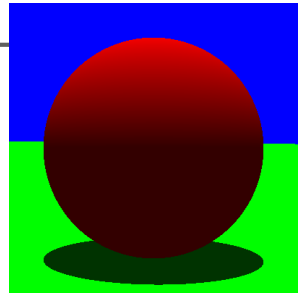
Parallel (grazing) view:
high reflection, low transmission

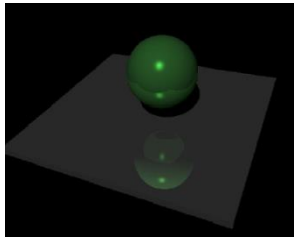
Ray tracer on GPU: RTX

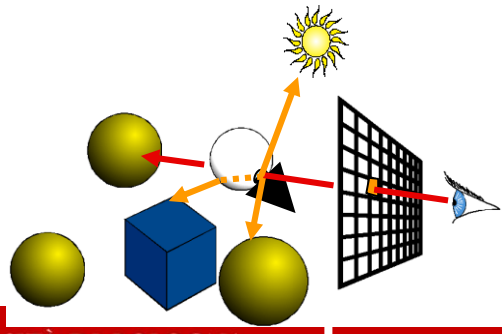
Image credit: NVIDIA (this ray traced image can be rendered at interactive rates on modern GPUs

Ray tracing

- Shadows/ Direct Illumination
- Reflection
- Refraction
- Recursive Ray Tracing

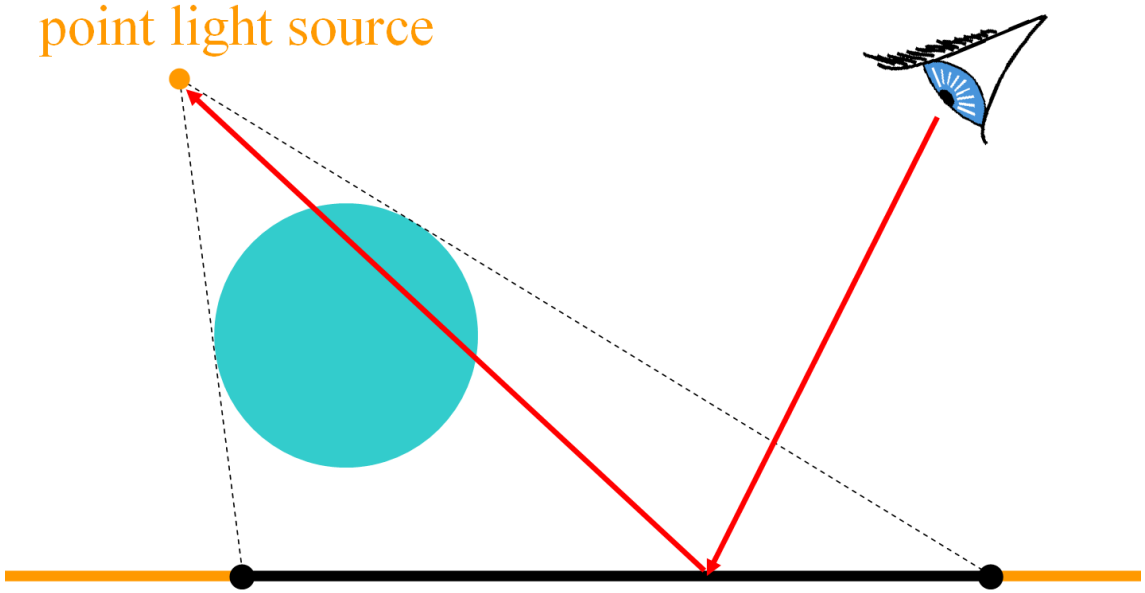


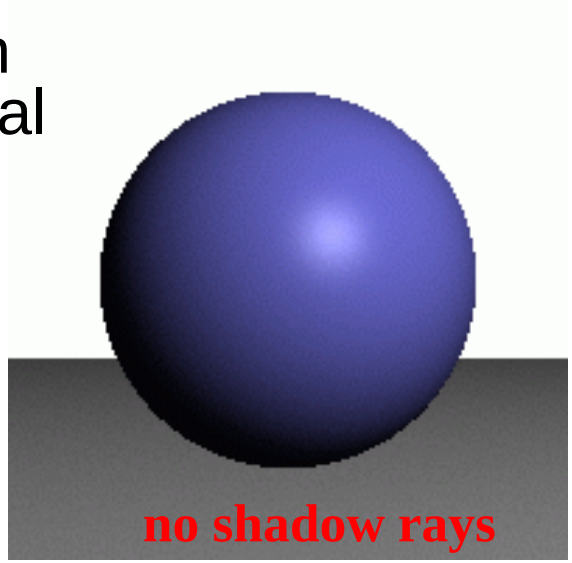


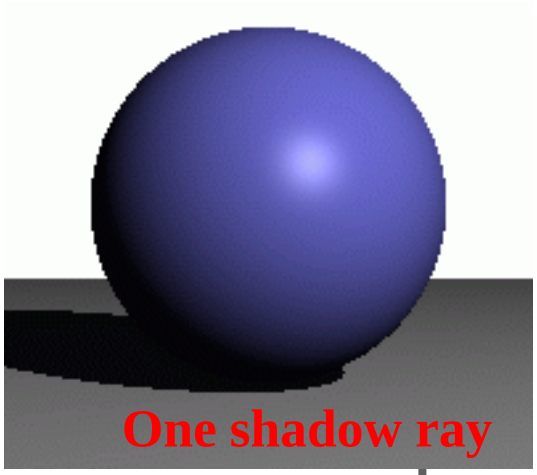


Shadows in Ray Tracing

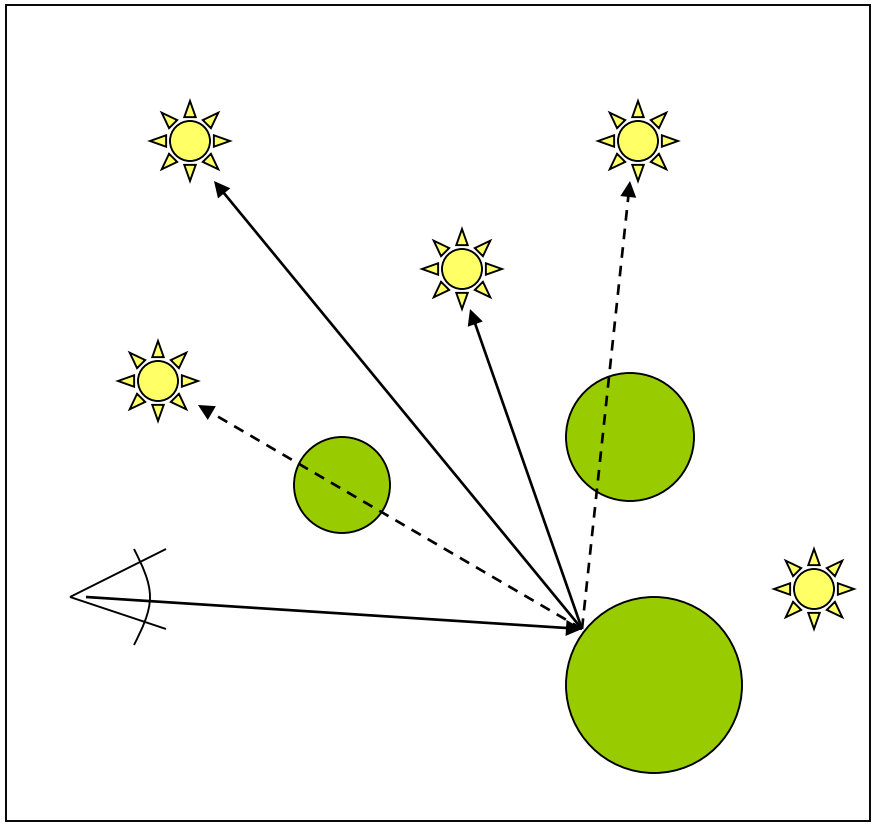
- If we wish to compute the illumination with shadows for a point, we shoot an additional ray from the point to every light source
- A light is only allowed to contribute to the final color if the ray doesn't hit anything in between the point and the light source





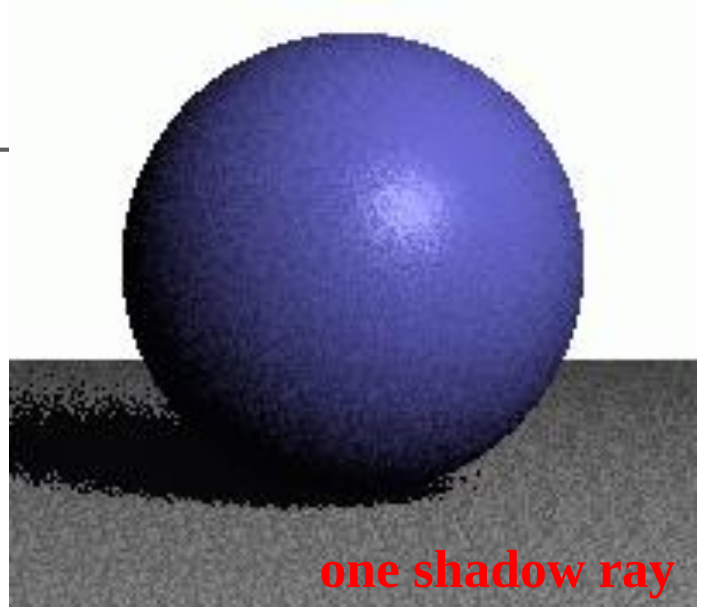


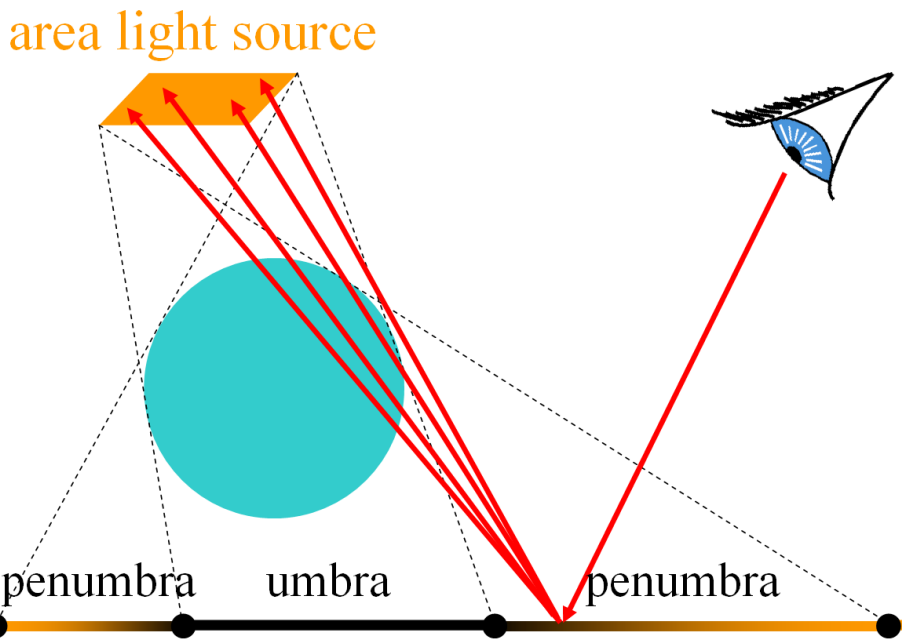
Shadow Rays

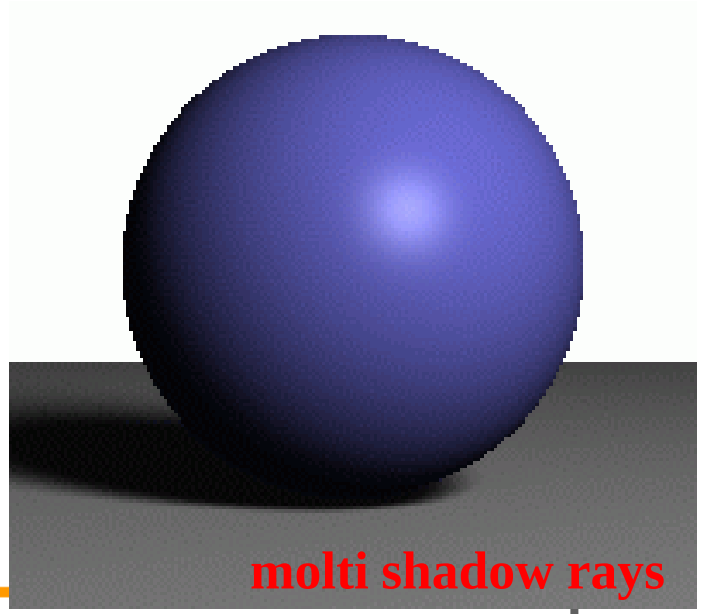


Soft Shadows

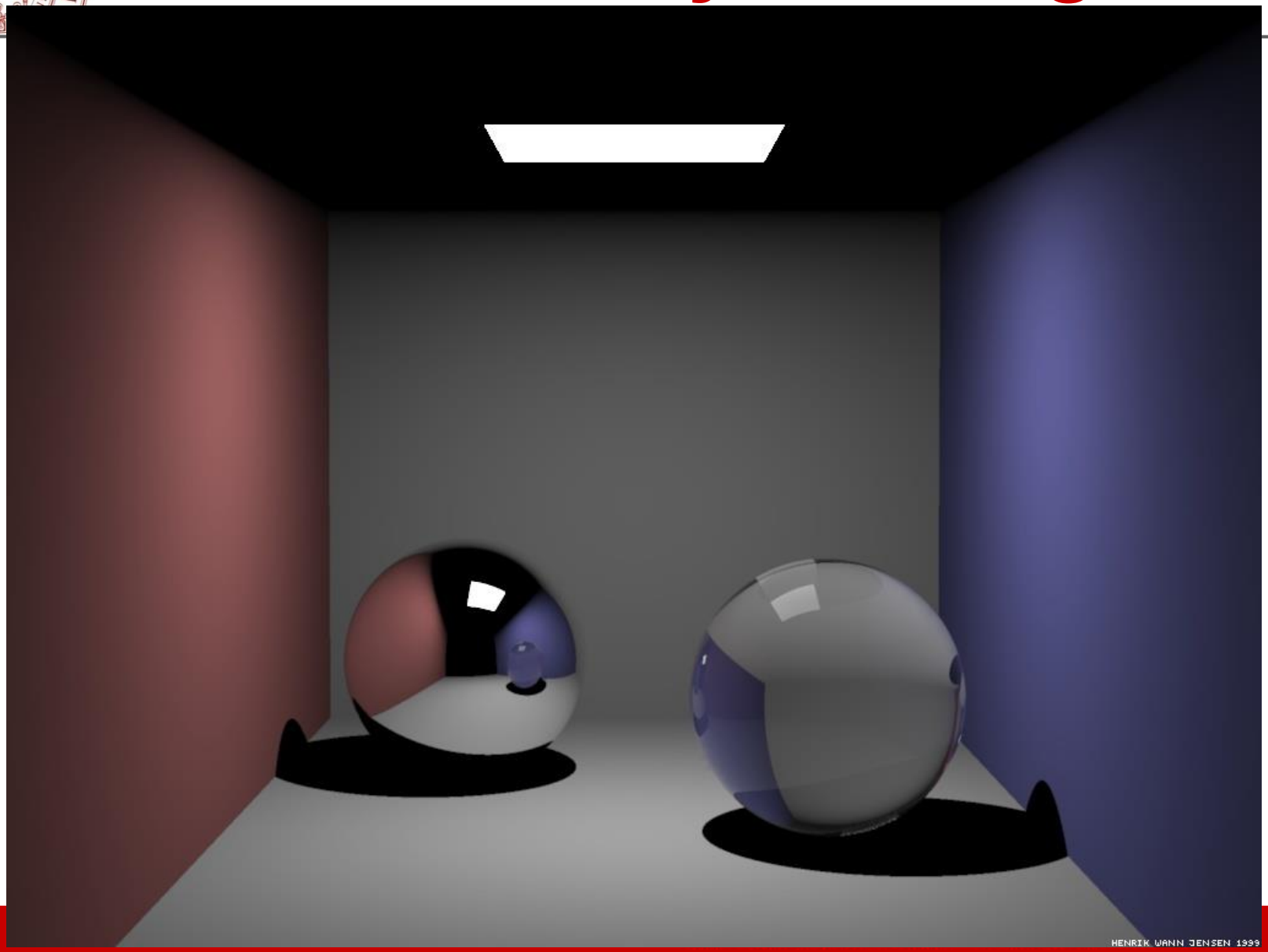
- Multiple shadow rays to reproduce an area light source (soft shadows)



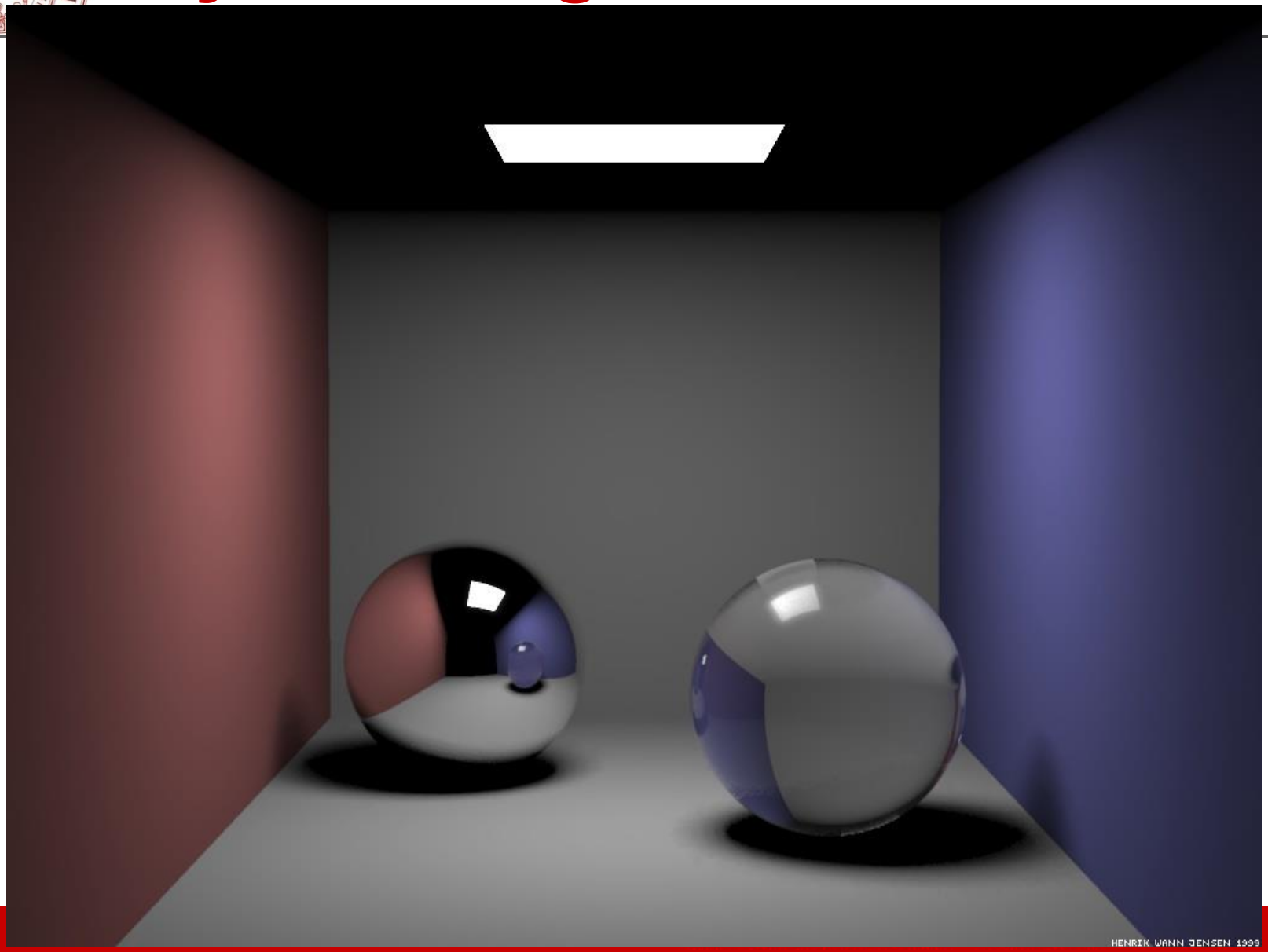




Basic Ray Tracing

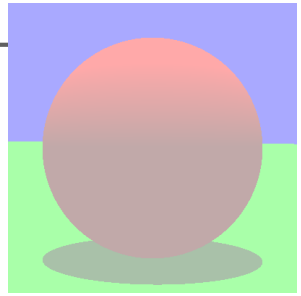


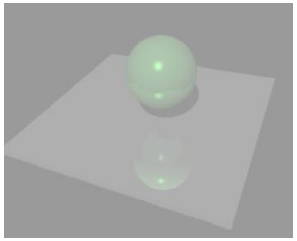
Ray Tracing + Soft Shadows

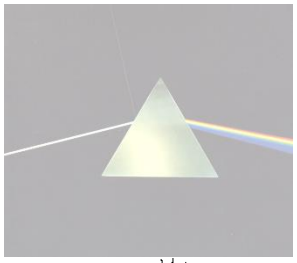


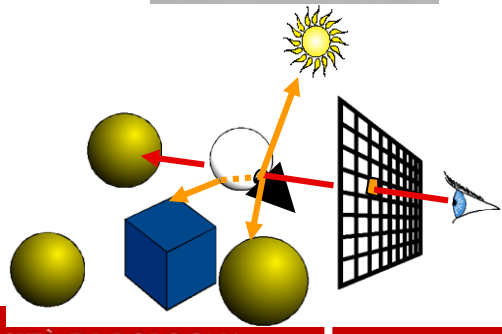
Ray tracing

- Shadows/Direct Illumination
- Reflection
- Refraction
- Recursive Ray Tracing



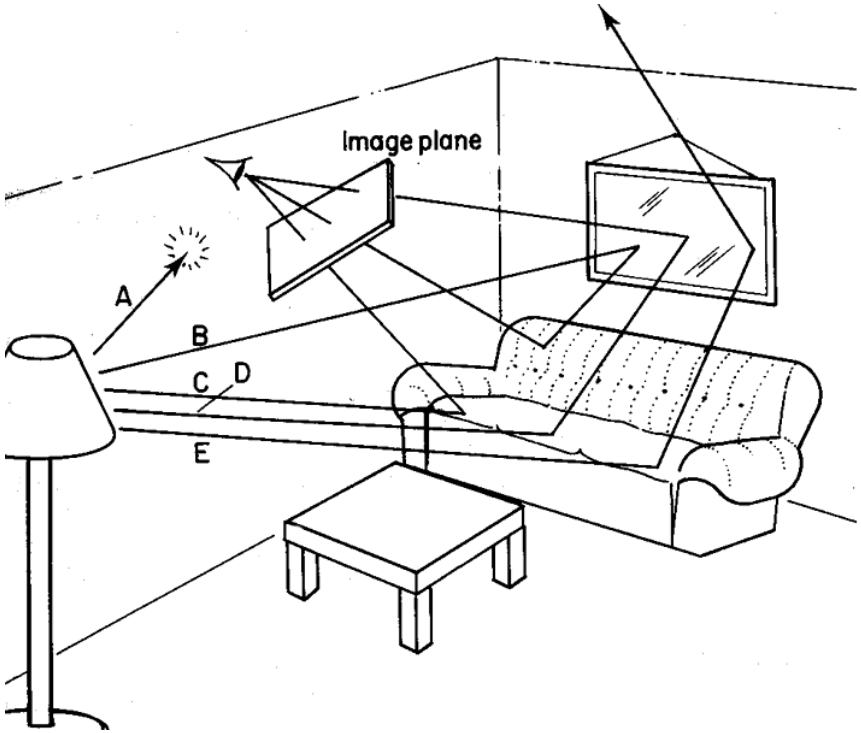


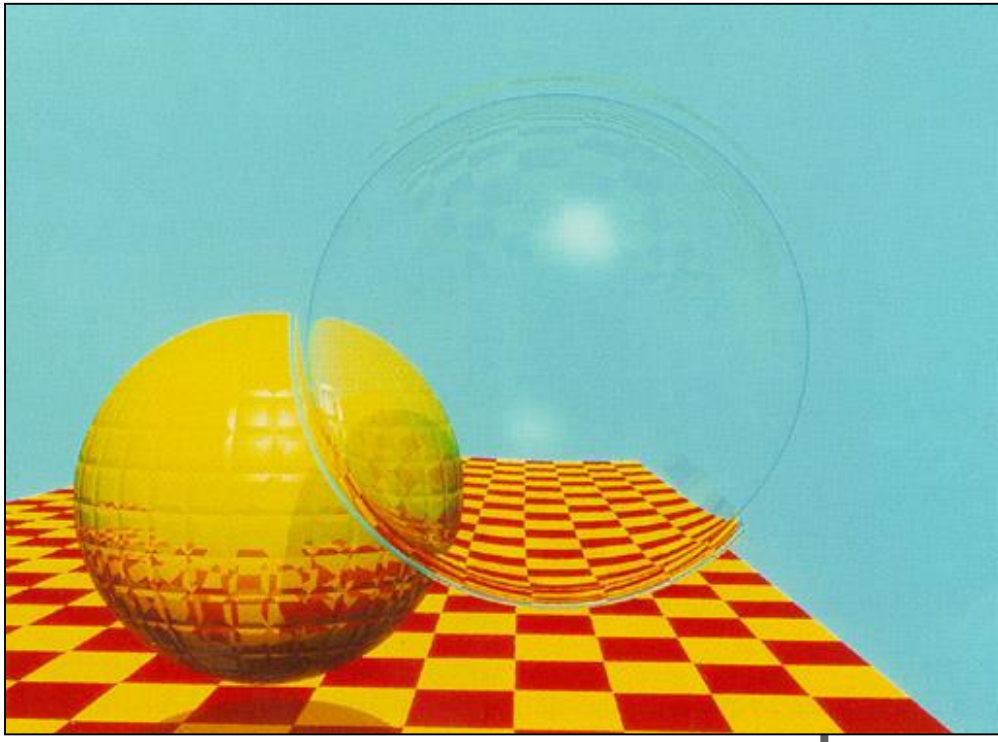




Ray Tracing History

- Ray Casting: Appel, 1968
- CSG and quadrics: Goldstein & Nagel 1971
- Recursive ray tracing: Whitted, 1980

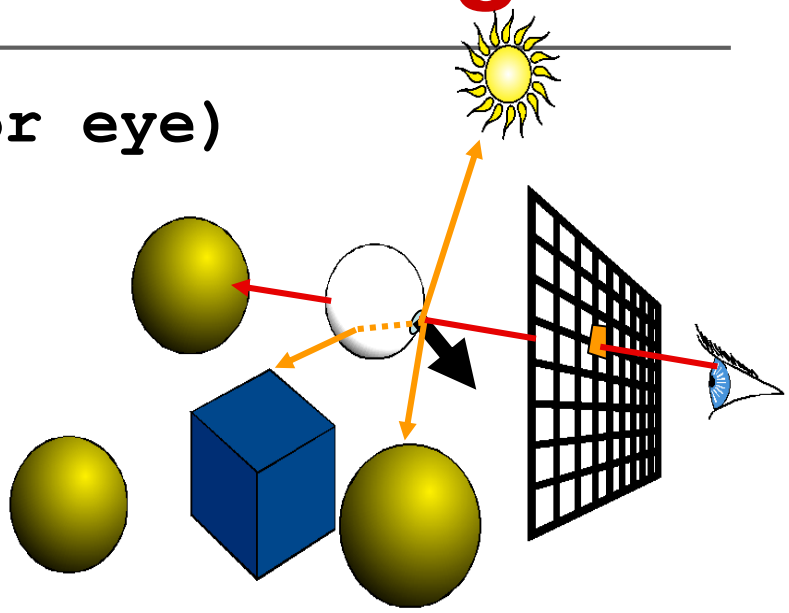




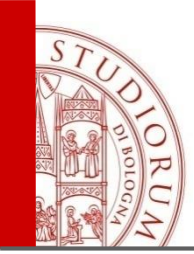
Ray Tracing: recursive algorithm

```
Image raytraceImage(vector eye)

begin
Image image;
for (each scanline Y)
      for (each pixel X)
   image[X][Y]=
   traceray(eye,dirFromEyeToXY,RecursionStep)
      end
   end
return image;
```



traceray() recursive procedure that trace a ray, defined by a point and direction, and compute the color of the first intersected surface (Initialize Recursion Step = max number of bounces – recursive steps)



traceray()

•Find the first intersection of the ray with the scene; if the ray does not hit any object then “background color” is returned; if the ray intersects the light, then light color is returned;

•Generates **shadow rays** to the light sources

•Calculates the illumination of that point according to the local lighting model

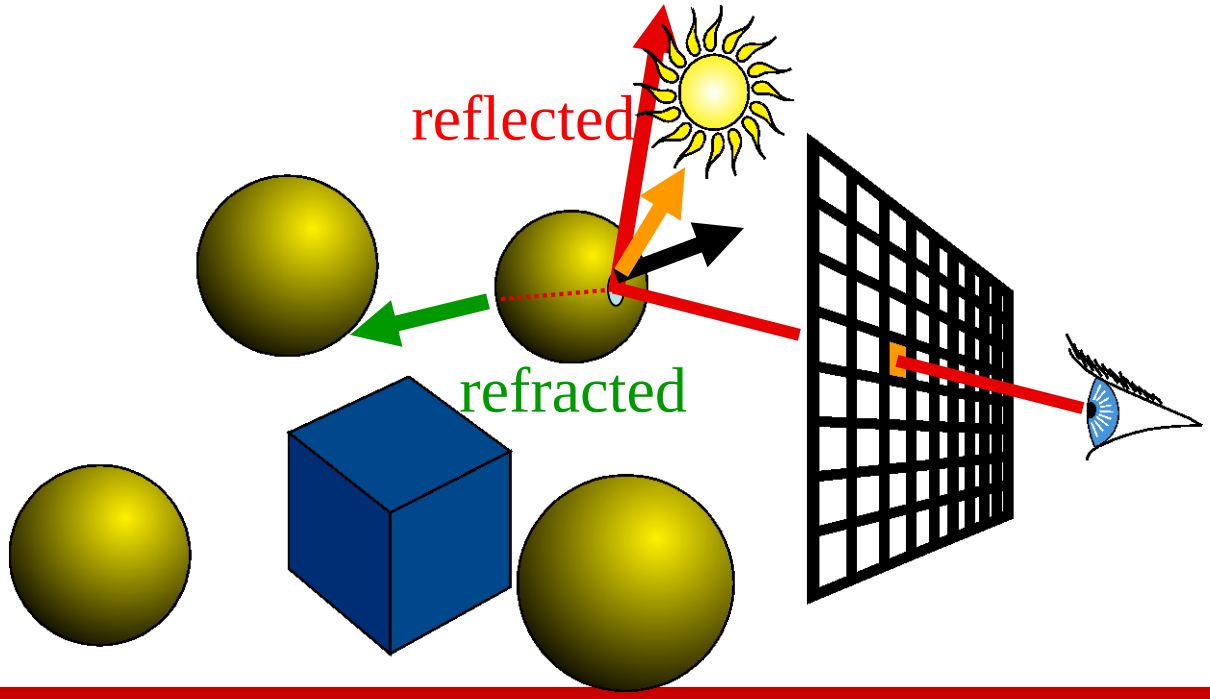
•Spawns a **reflection ray r** and a **transmission ray t** as appropriate

•Calls itself recursively with r and t

•Combines the resulting illumination contributions and returns

Ray Tracing

Any reflected, refracted, shadows rays are called *secondary rays*



```
color c=traceray(point p, vector d, int step)
begin
color local, riflesso, trasmesso; point q; normal n;

if (step > max) return(background-color);
[obj,q] = intersect(p,d,status);
if (status = light-source) return(light-source-color);
if (status = no-intersection) return(background-color);

/* calcolo della normale al punto di intersezione,della
direzione del raggio riflesso e del raggio trasmesso */
n=normal(q);
r=reflect(q,n);
t=transmit(q,n);

local = phong(q,n,r);
If  (obj.mirror)      riflesso  = traceray(q,r,step+1);
If (obj.transparent) trasmesso = traceray(q,t,step+1);
return(local + trasmesso + riflesso);
end
```

Lighting Equation Model

$$I = I_{local} + K_r I_{reflect} + K_t I_{transmit}$$

Kr in [0,1] factor specifying what fraction of the light from the reflection direction is reflected

Kt in [0,1] material property, it specifies the fraction of light transmitted through the surface

I_reflect intensity of the incoming reflected light, computed recursively in the direction of the reflection ray;

I_transmit intensity of the incoming reflected light, computed recursively in the transmission direction;

I_local lighting as computed by the local illumination model (Phong)

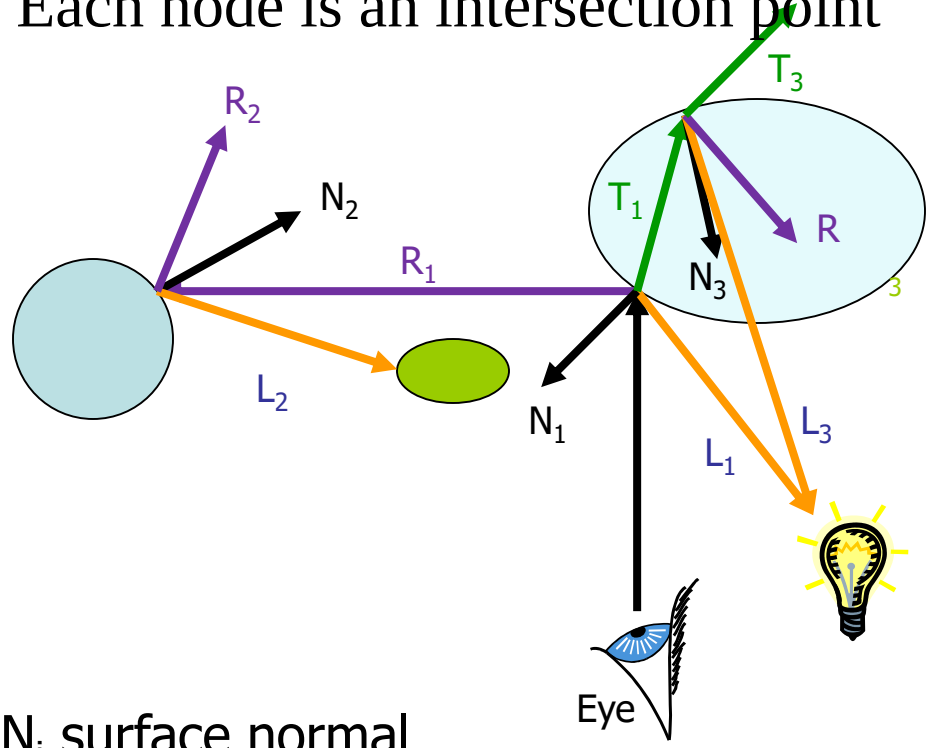
Ray Tracing: recursive algorithm

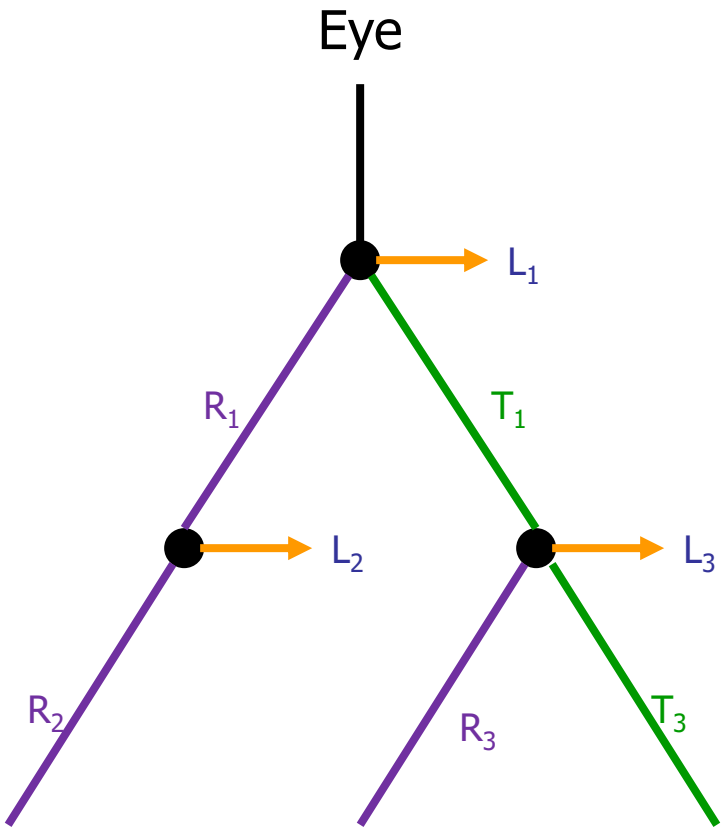
intersect() find intersections of a ray with objects in the virtual scene. Test the ray against each surface of each object for possible intersections.

It should be computationally efficient, it is the major portion of the raytracing execution time

The ray Tree

Each node is an intersection point





N_i surface normal

R_i reflected ray

L_i shadow ray

T_i transmitted (refracted) ray

Stopping Criteria

- Recursion depth
  - Stop after a number of bounces
- Ray contribution
  - Stop if transparency/transmitted attenuation becomes too small

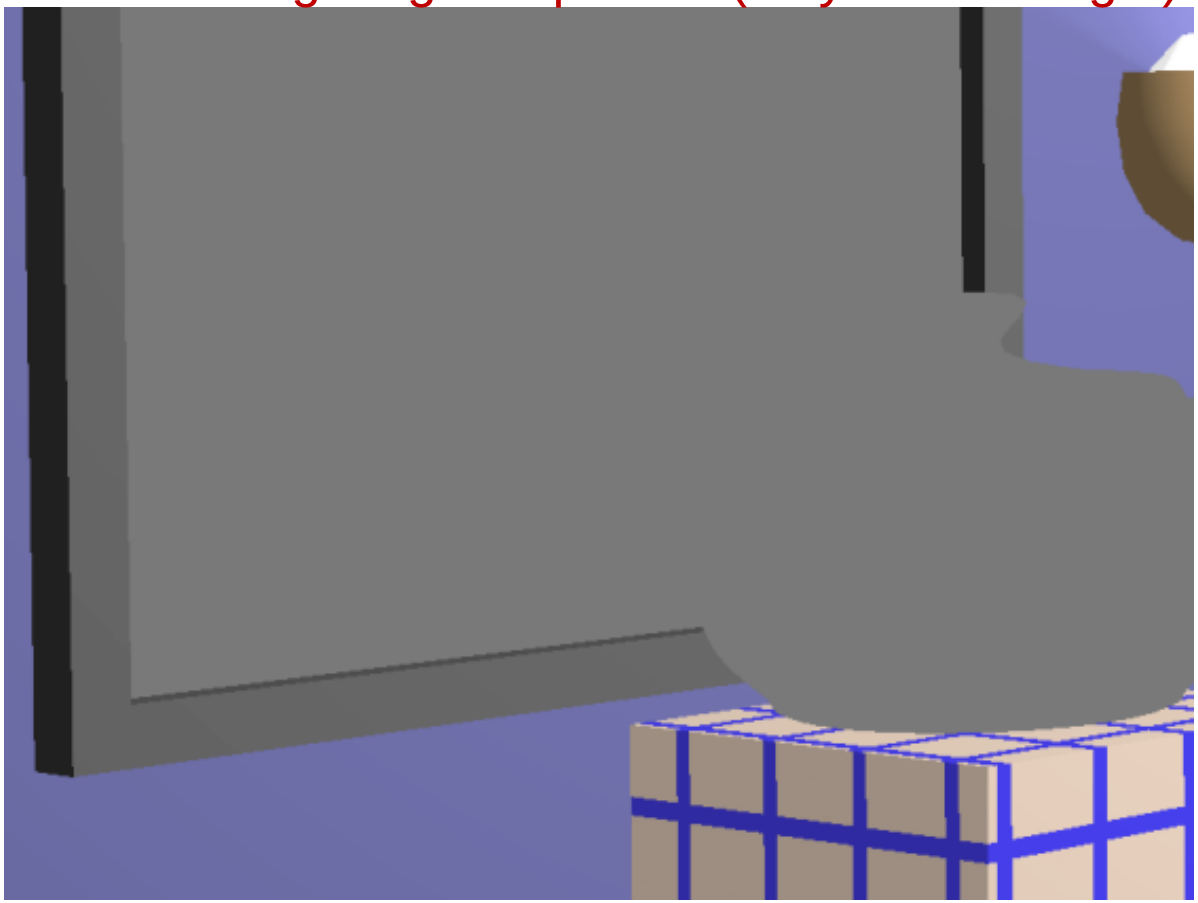
Compute the energy left and thus reflected by a ray at each intersection

Add a parameter (energy) to the recursive call:

```
color c = traceray(point p, vector d, int step,
                                      float energy)
```

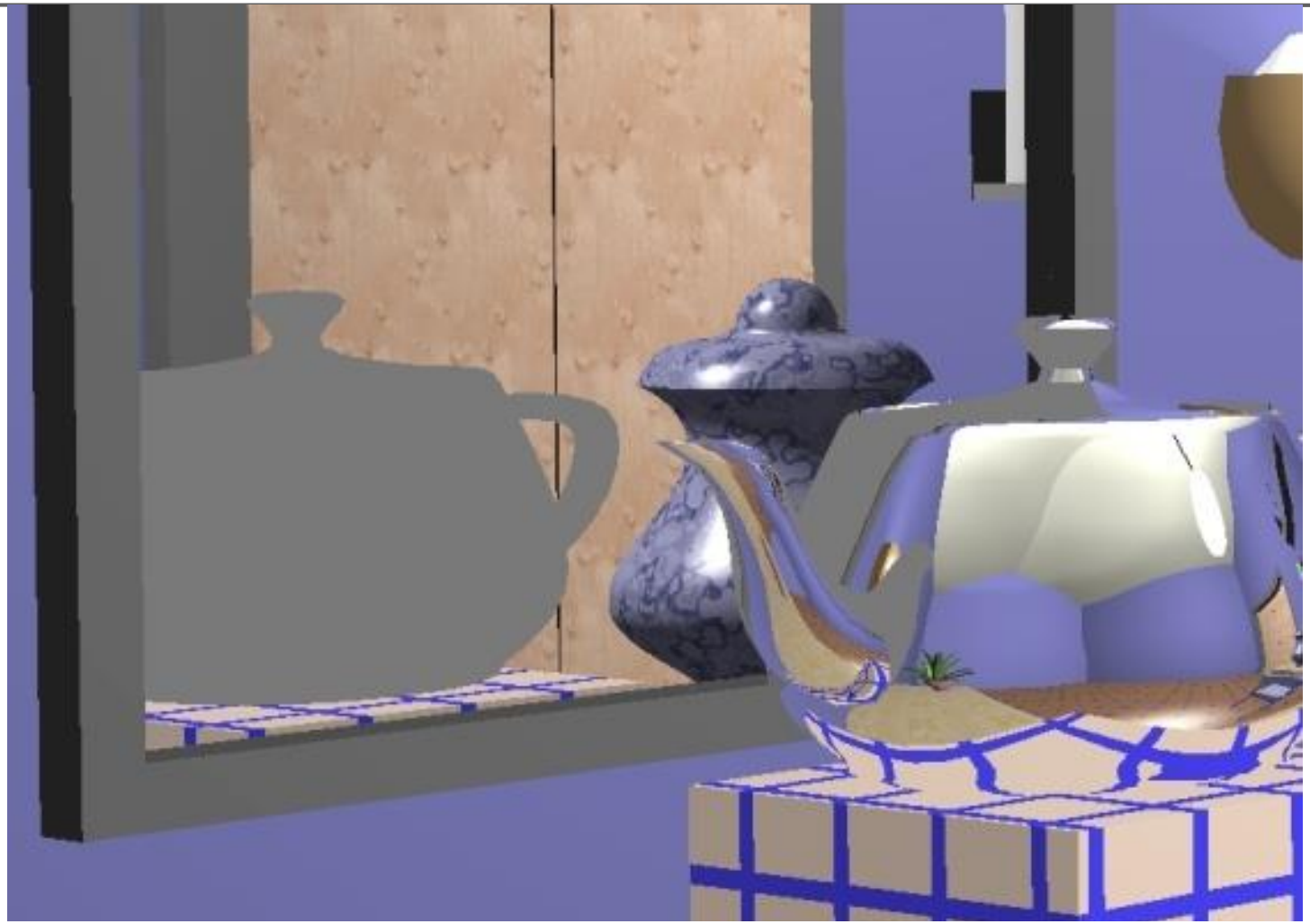
Effects of recursive depth

Recursive depth from 0 to 7. Teapot and mirror are perfect specular objects without local lighting component (only ambient light)



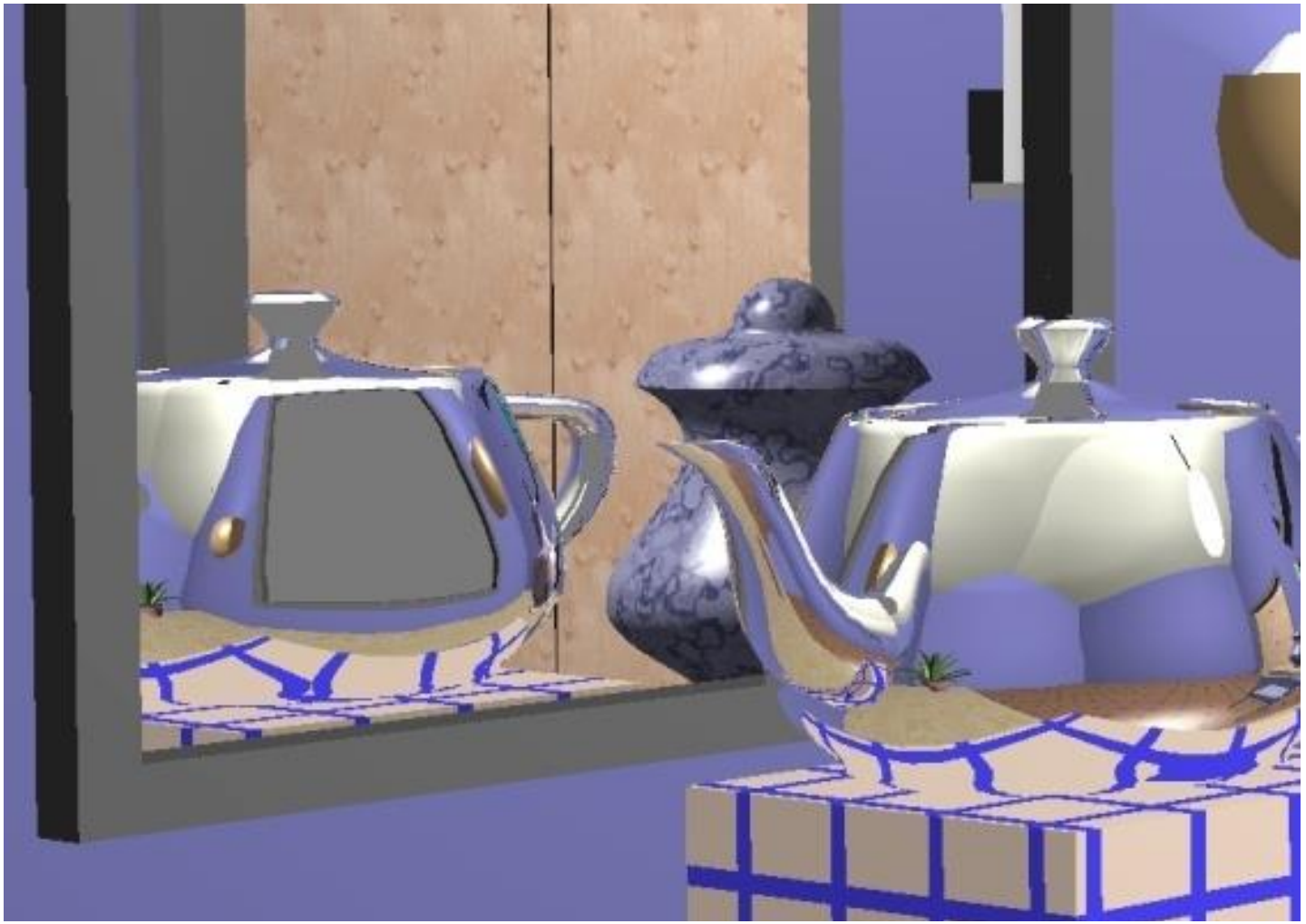
depth=0: the rays stop on mirror and teapot without local contribution

Effects of recursive depth



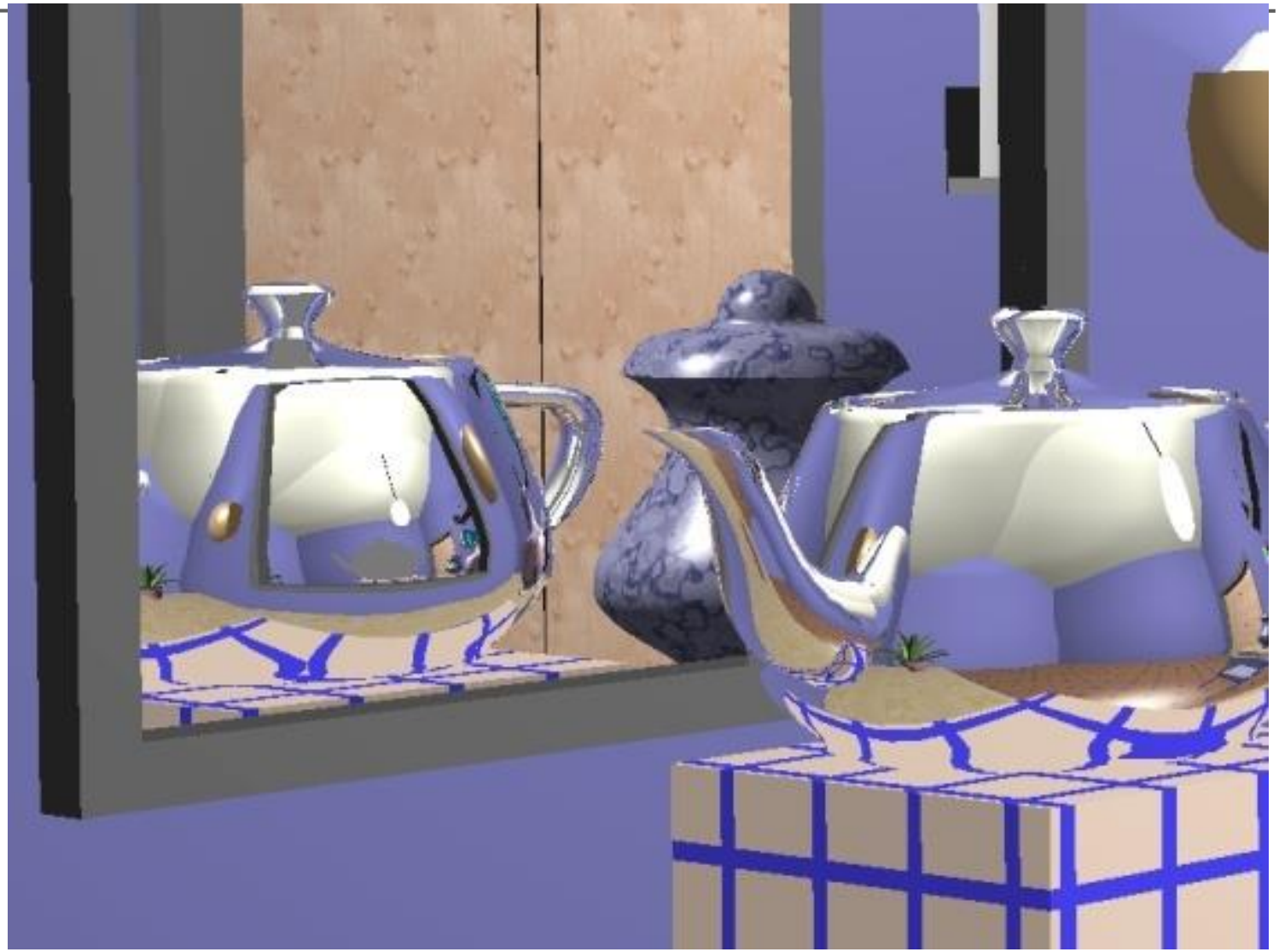
depth=1: rays eye/mirror/ teapot stop on teapot and produce a gray shadow on the mirror

Effects of recursive depth



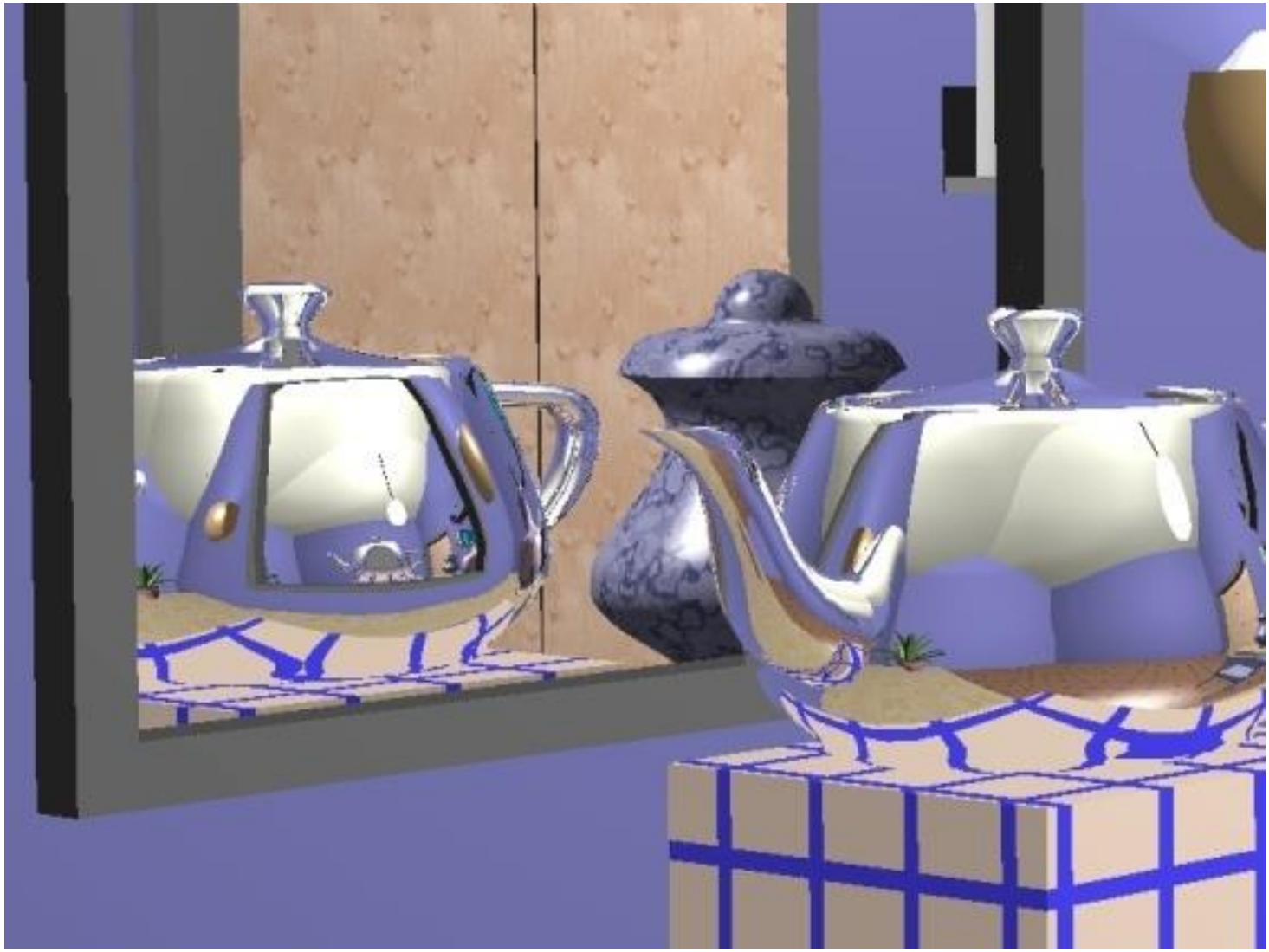
depth=2: gray shadow on the reflectd teapot on the mirror due to the unique ambient light contribution

Effects of recursive depth



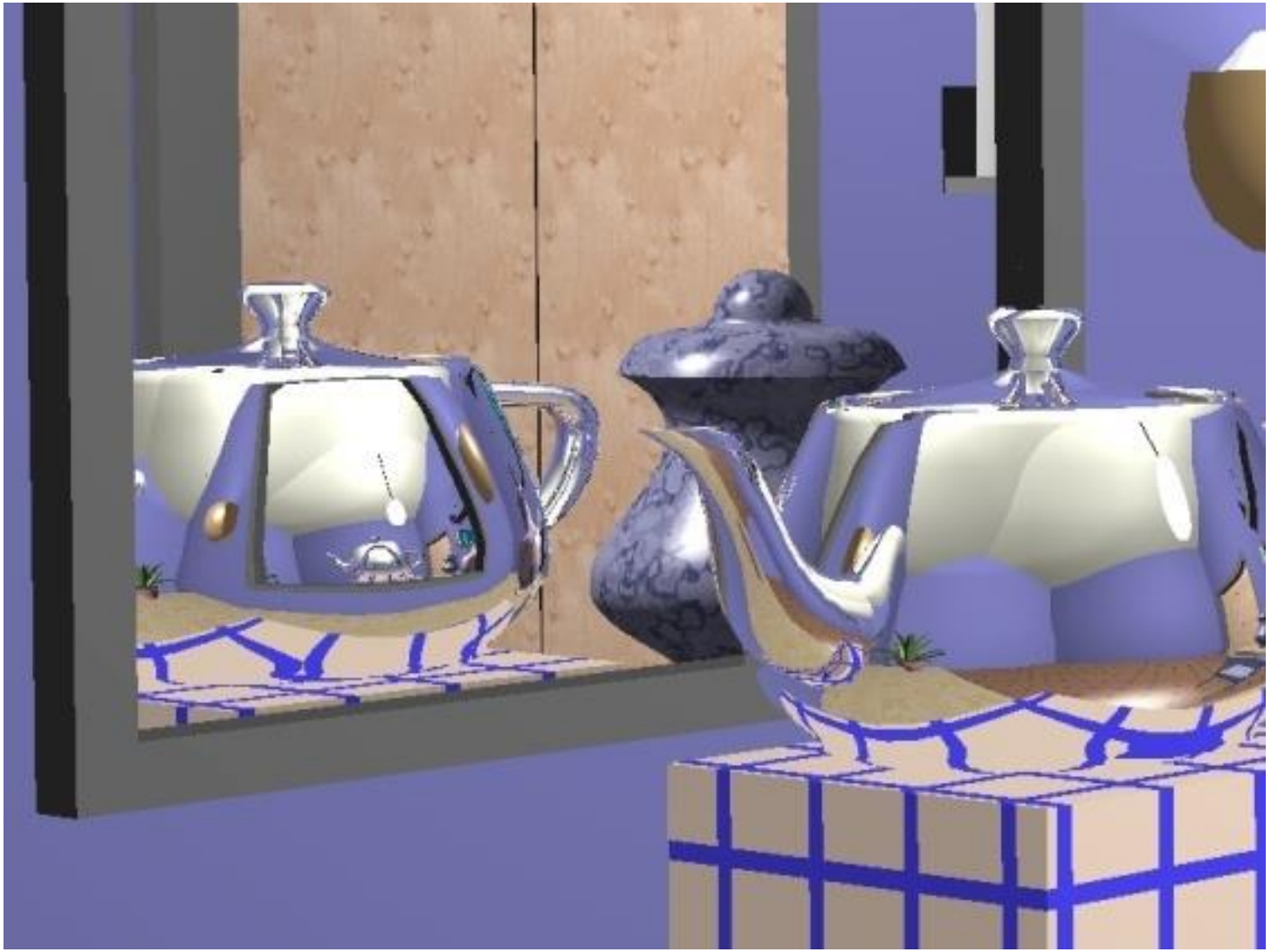
depth=3

Effects of recursive depth



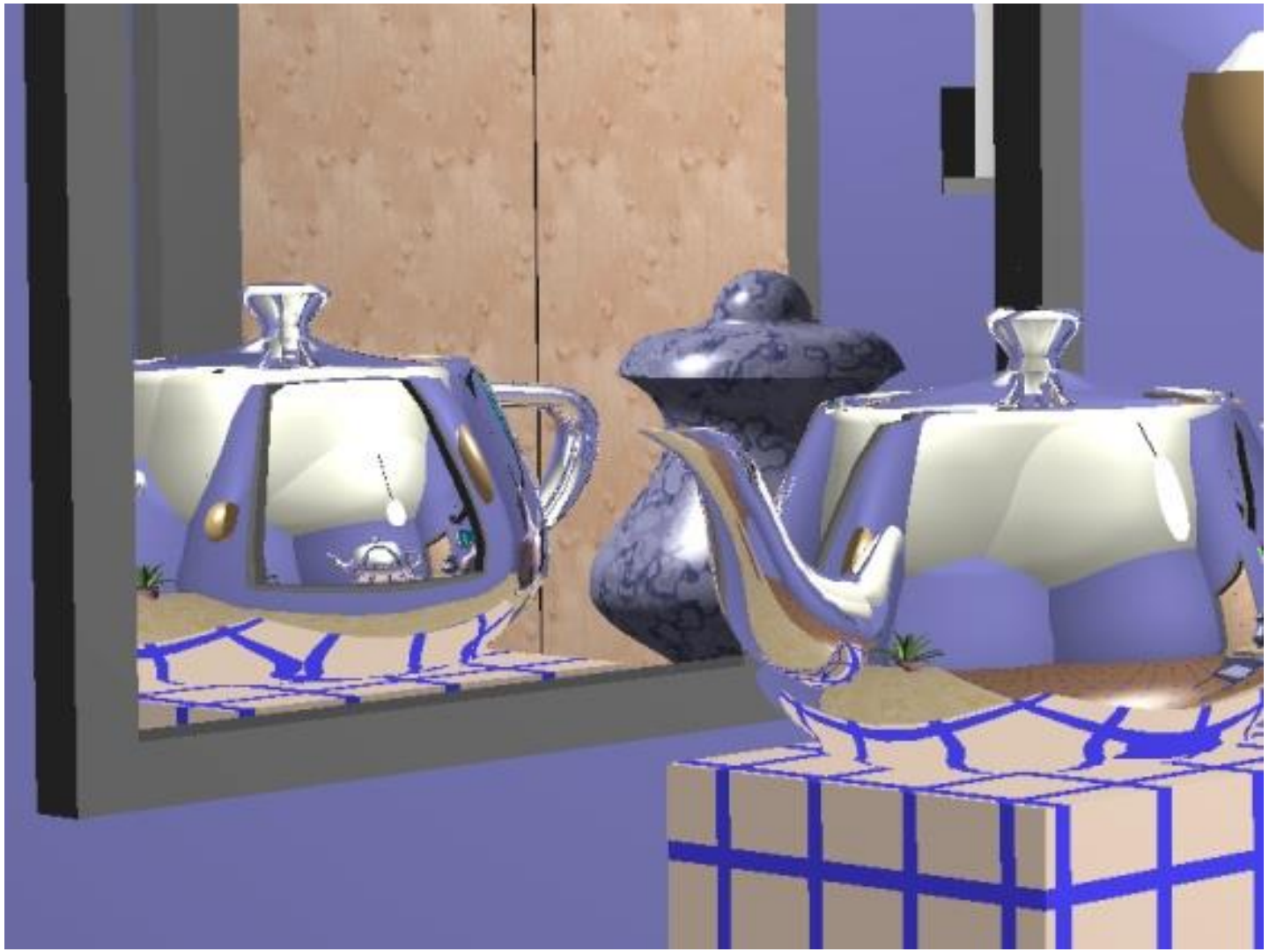
depth=4

Effects of recursive depth



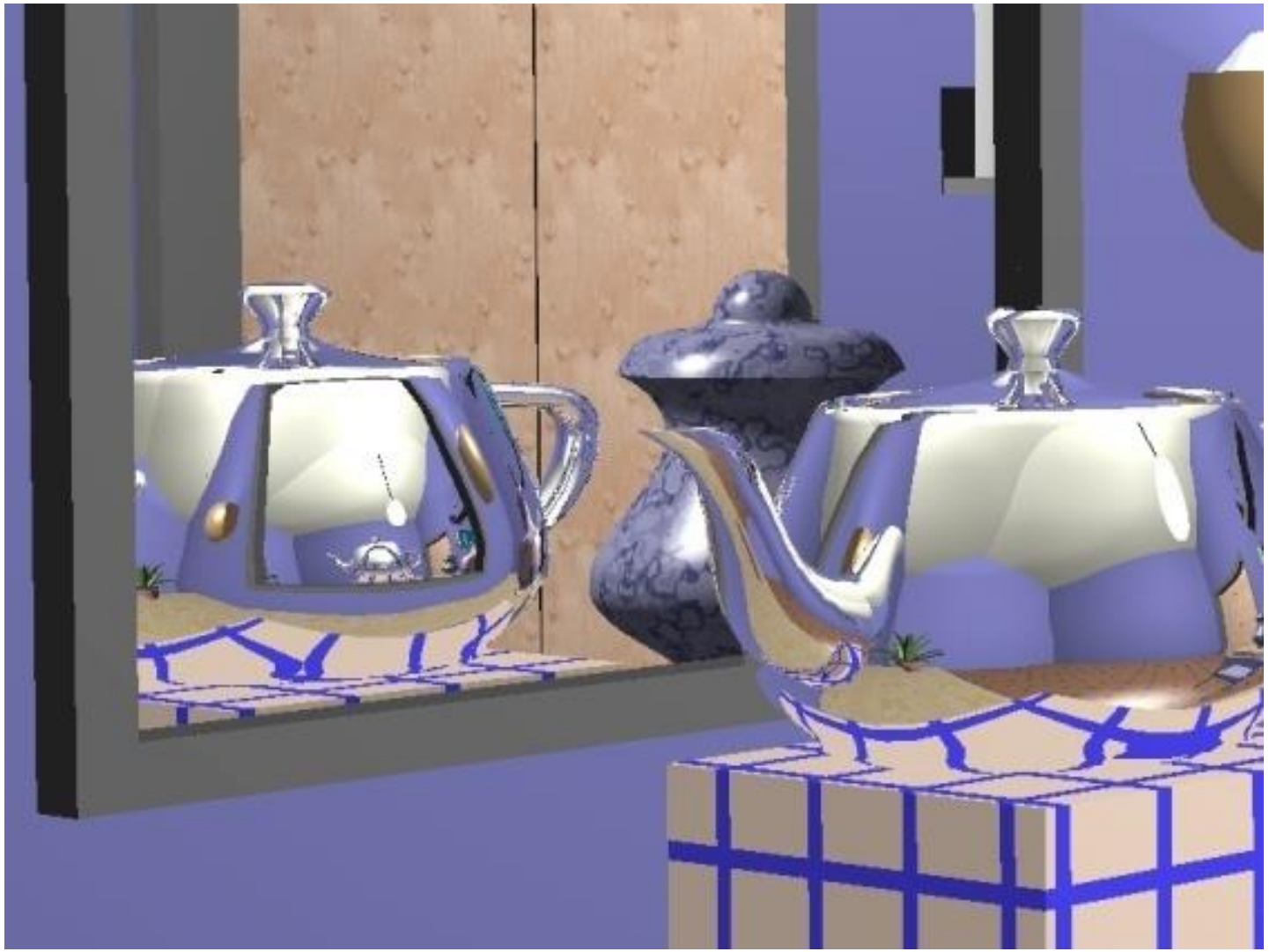
depth=5

Effects of recursive depth



depth=6

Effects of recursive depth



depth=7

Intersection ray - surface

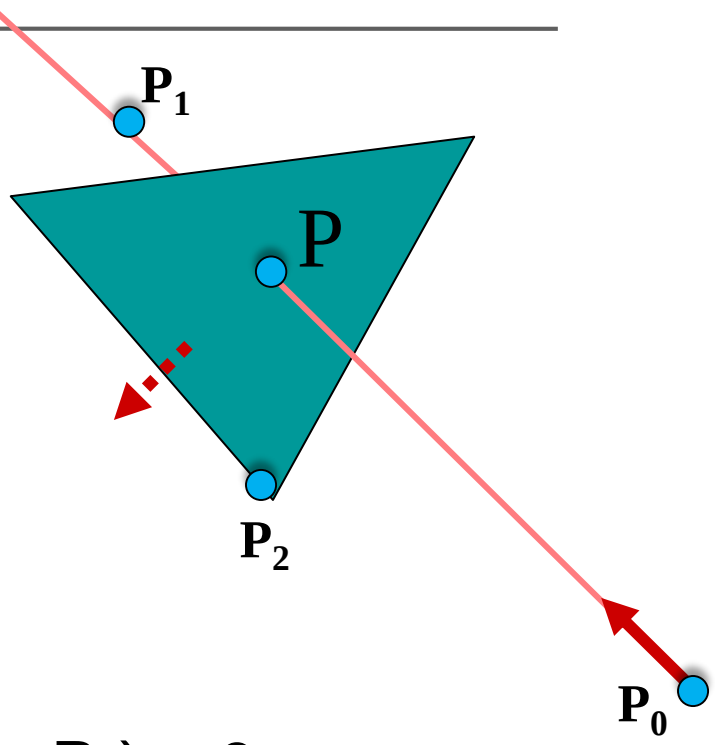
- This is the most computationally expensive procedure in the recursive ray tracing
- Objects must be stored in smart data structures
- Most of the ray tracers work with poly mesh only

- Ray- triangle
- Ray- sphere
- Ray-parametric patch (NURBS)
- Ray-implicit surface

Step 1: Ray – Plane Intersection

- Explicit (Parametric) Ray equation

$$P(t) = P_0 + t\,(P_1 - P_0) \qquad t \text{ in } [0,\infty)$$
$$P(t) = (1-t)\,P_0 + t\,P_1$$



- How do we intersect?

  Insert explicit equation of ray into the plane equation

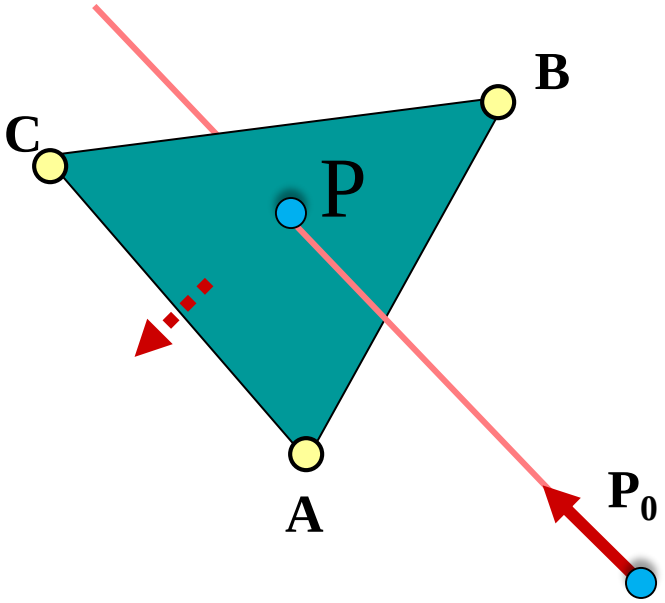
- $P(t) = (1 - t)P_0 + tP_1 \qquad n \cdot (P(t) - P_2) = 0$

Find $\mathbf{t = n\cdot(P_2 - P_0)/n\cdot(P_1 - P_0)}$

Then find the intersection point P = P(t)

Step 2: Ray - triangle intersection

- Check if P is inside or outside the triangle



Compute barycentric coordinates of P with respect to the triangle A B C

$$P(\alpha, \beta, \gamma) = \alpha A + \beta B + \gamma C$$

$$\text{con } \alpha + \beta + \gamma = 1$$

P belongs to the triangle if

$$0 < \alpha < 1$$

$$0 < \beta < 1$$

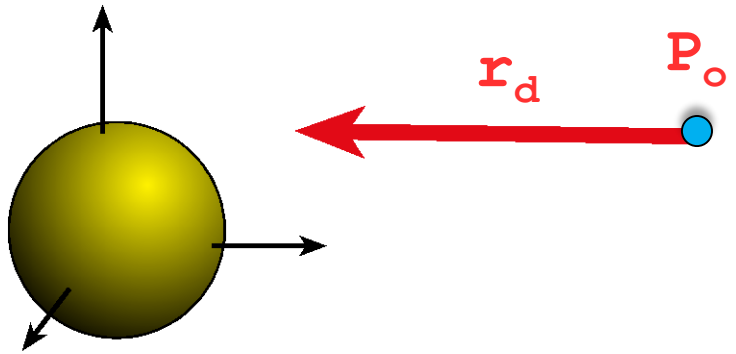
$$0 < \gamma < 1$$

Ray-Sphere Intersection

- Sphere Equation (centered at the origin c=0) (implicit): $(P(t)-c)^2 = r^2$
- Ray Equation (parametric):

$$P(t) = P_0 + t\,(P_1 - P_0) = P_0 + t\,r_d \qquad t \text{ in } [0,\infty), ||\,r_d\,|| = 1$$

- An intersection point satisfies both



Ray-Sphere Intersection

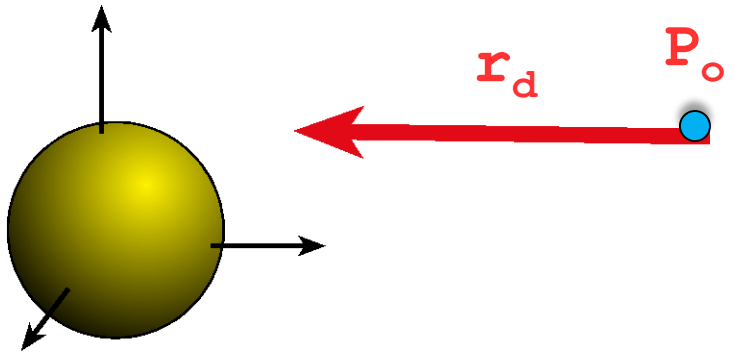
$$0 = P(t) \cdot P(t) - r^2$$

$$0 = (P_0 + tr_d) \cdot (P_0 + tr_d) - r^2$$

$$0 = P_0 \cdot P_0 + 2tr_d P_0 + t^2 r_d{}^2 - r^2$$

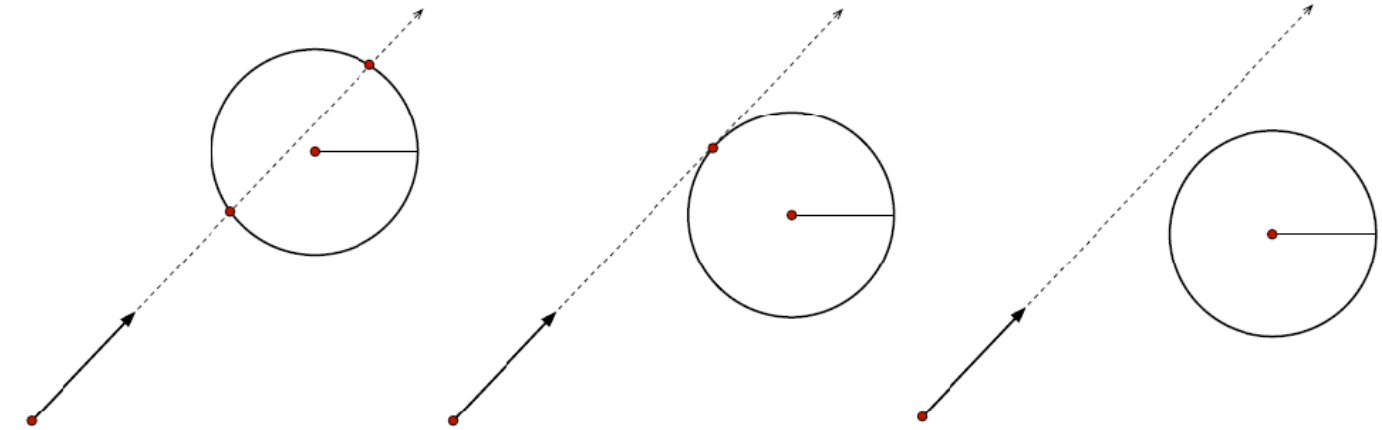
$$0 = P_0 \cdot P_0 + 2tr_d P_0 + t^2 - r^2$$

Solve quadratic equations for t



Ray-Sphere Intersection

- Compute discriminant



$\Delta > 0$ $\Delta = 0$ $\Delta < 0$

For the case with two solutions, the smallest non-negative real *t* represents intersection nearest to eye-point

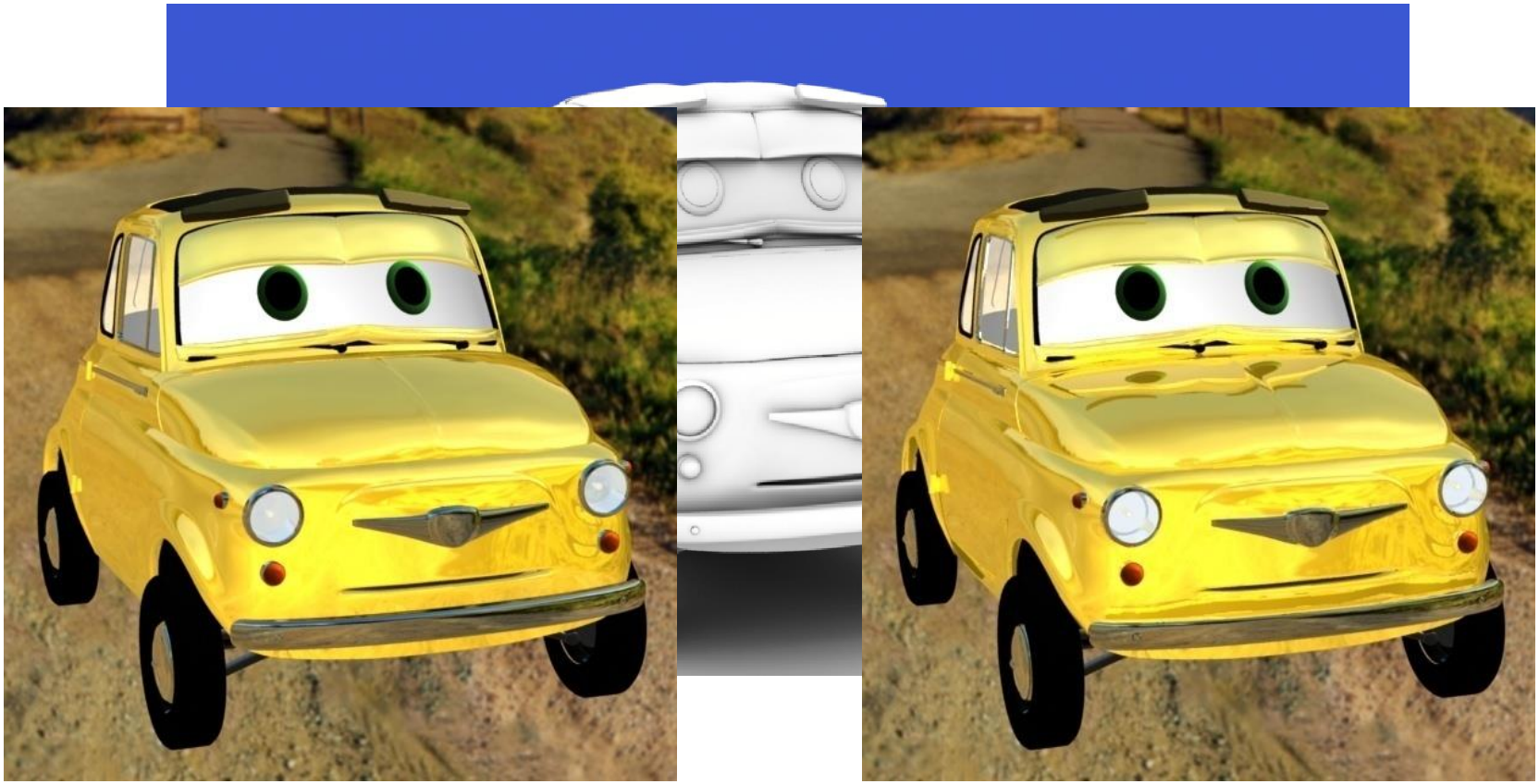
Ray tracing for ‘Cars’

Per Christensen

Pixar Animation Studios

PIXAR

Why ray tracing?



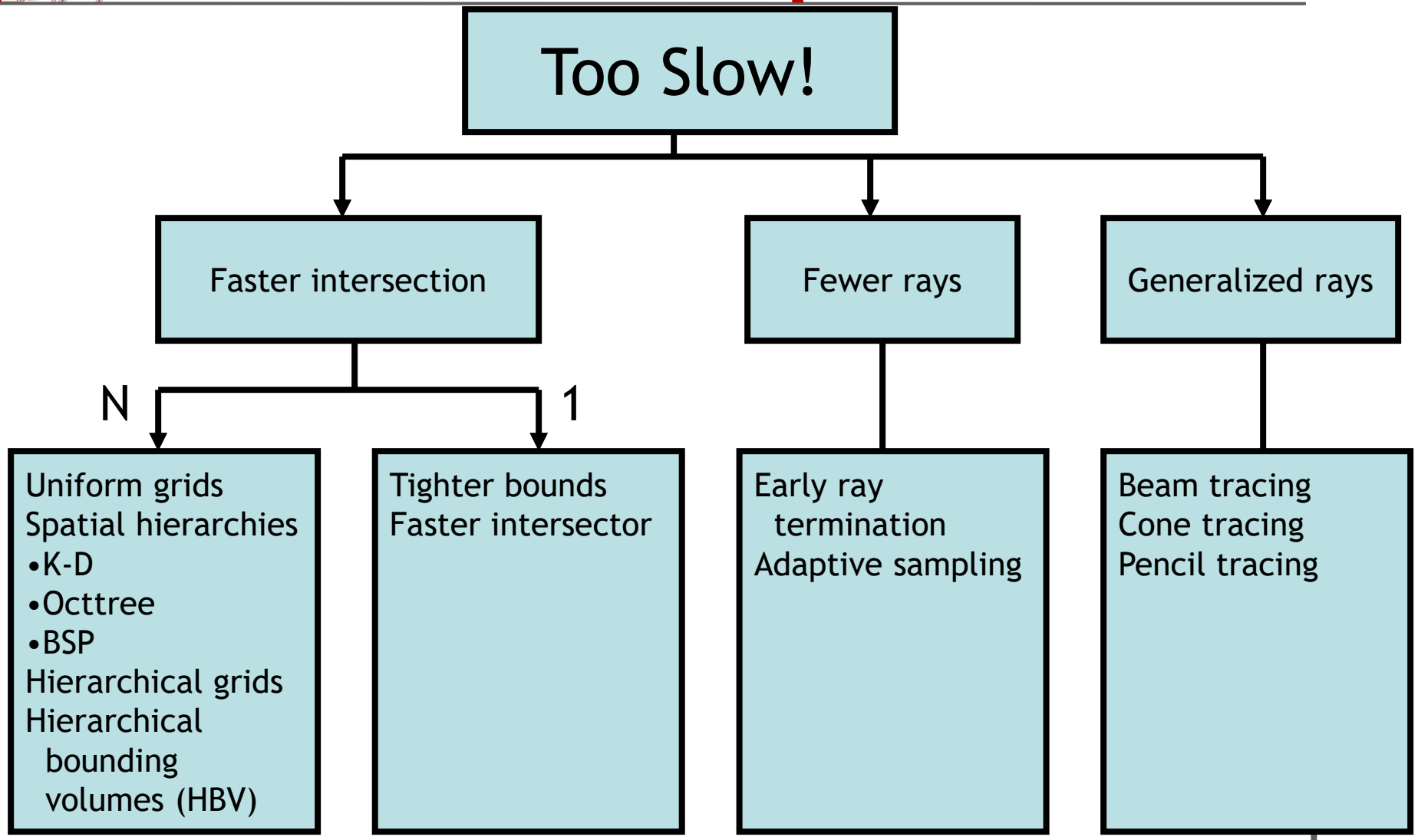
Environment map

Reflections using Ray-tracing

PIXAR

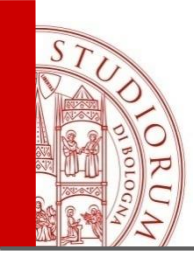
Ray tracing effects

Ray Tracing Acceleration Techniques



Radiosity vs. Ray Tracing

Scena senza la luce ambiente
RT è view-dependent,
Radiosity è view-independent



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Serena Morigi

Dipartimento di Matematica

serena.morigi@unibo.it

http://www.dm.unibo.it/~morigi