MIT 6.837 - Ray Tracing

Ray Tracing

Courtesy of James Arvo and David Kirk. Used with permission.

MIT EECS 6.837

Frédo Durand and Barb Cutler

Some slides courtesy of Leonard McMillan

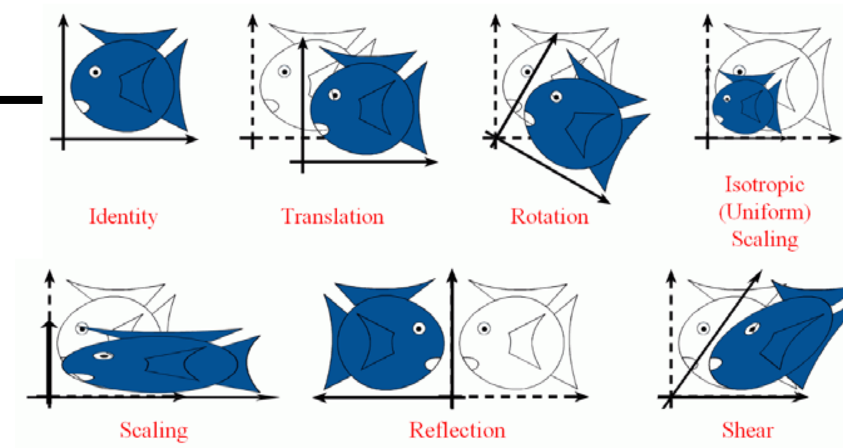
Administrative

- Assignment 2
  - Due tomorrow at 11:59pm
- Assignment 3
  - Online this evening
  - Due Wednesday October 1

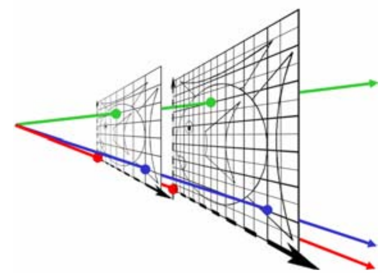
Review of last week?

Review of last week

- Linear, affine and projective transforms



- Homogeneous coordinates



- Matrix notation
- Transformation composition is not commutative
- Orthonormal basis change

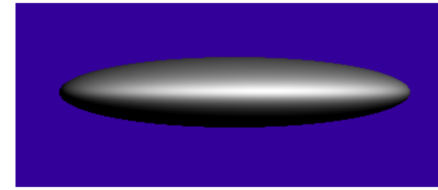
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

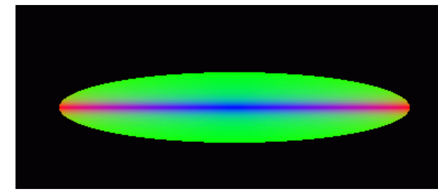
Review of last week

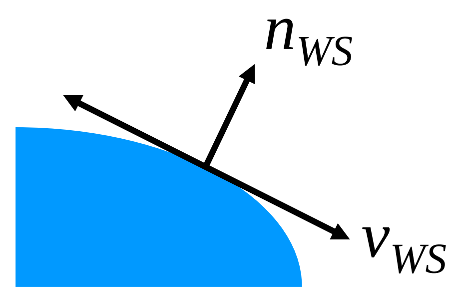
- Transformation for ray tracing
  - Transforming the ray
    - For the direction, linear part of the transform only
  - Transforming t or not
  - Normal transformation

$$n_{WS}^{\mathbf{T}} = n_{OS}\ (\mathbf{M}^{-1})$$

- Constructive Solid Geometry (CSG)



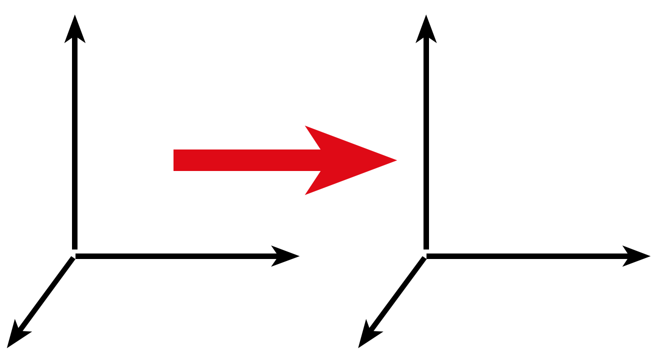




Fun with transformations: Relativity

Digression

- Special relativity: Lorentz transformation
  - 4 vector (t, x, y, z)
    - 4th coordinate can be ct or t
  - Lorentz transformation depends on object speed v



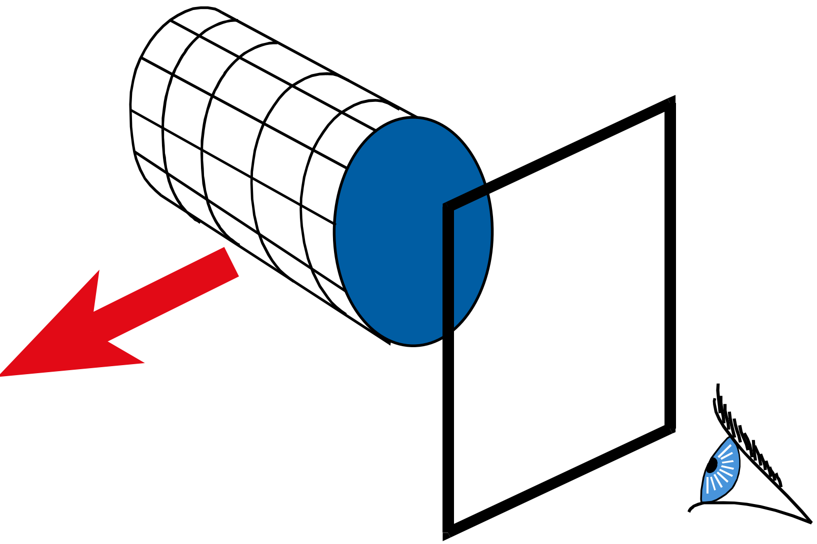
$$\begin{pmatrix} t' \\ x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \gamma & -\gamma v & 0 & 0 \\ -\gamma v & \gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} t \\ x \\ y \\ z \end{pmatrix}$$

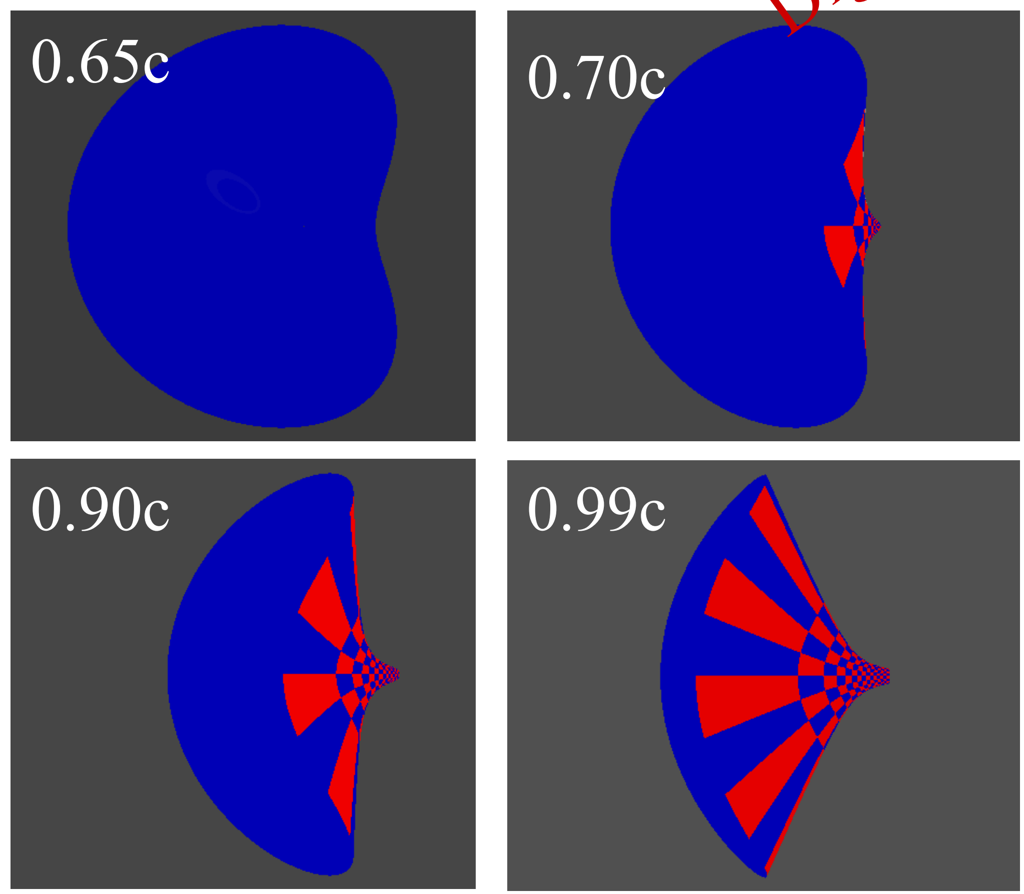
http://casa.colorado.edu/~ajsh/sr/sr.shtml

Relativity

- Transform ray by Lorentz transformation

Digression





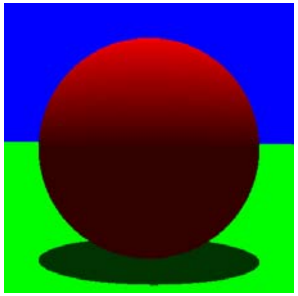
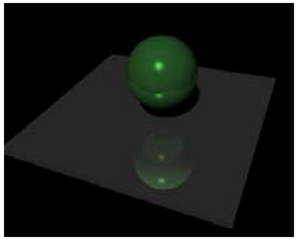
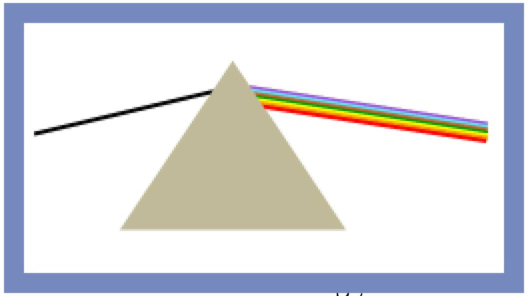
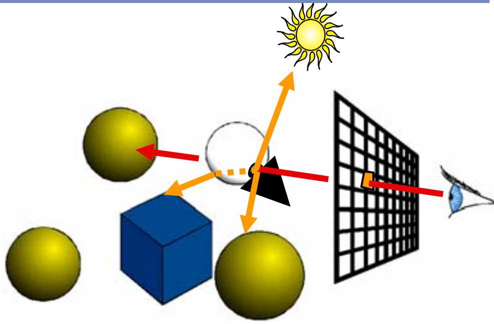
See also http://www.cs.mu.oz.au/~andrbh/raytrace/raytrace.html

Today: Ray Tracing

Image removed due to copyright considerations.

Overview of today

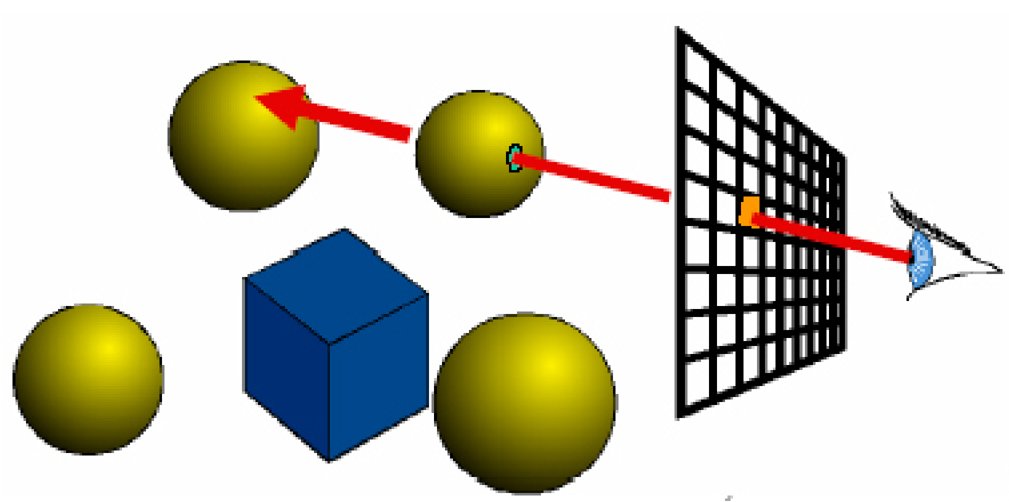
- Shadows
- Reflection
- Refraction
- Recursive Ray Tracing

Ray Casting (a.k.a. Ray Shooting)

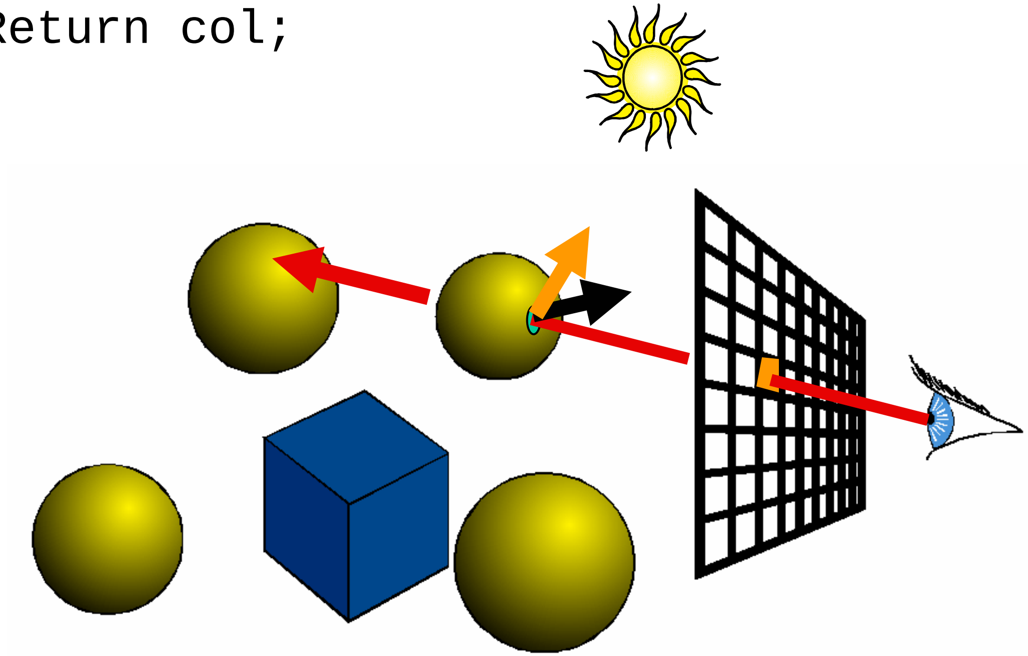
```
For every pixel (x,y)
  Construct a ray from the eye
  color[x,y]=castRay(ray)
```

- Complexity?
  - O(n * m)
  - n: number of objects, m: number of pixels



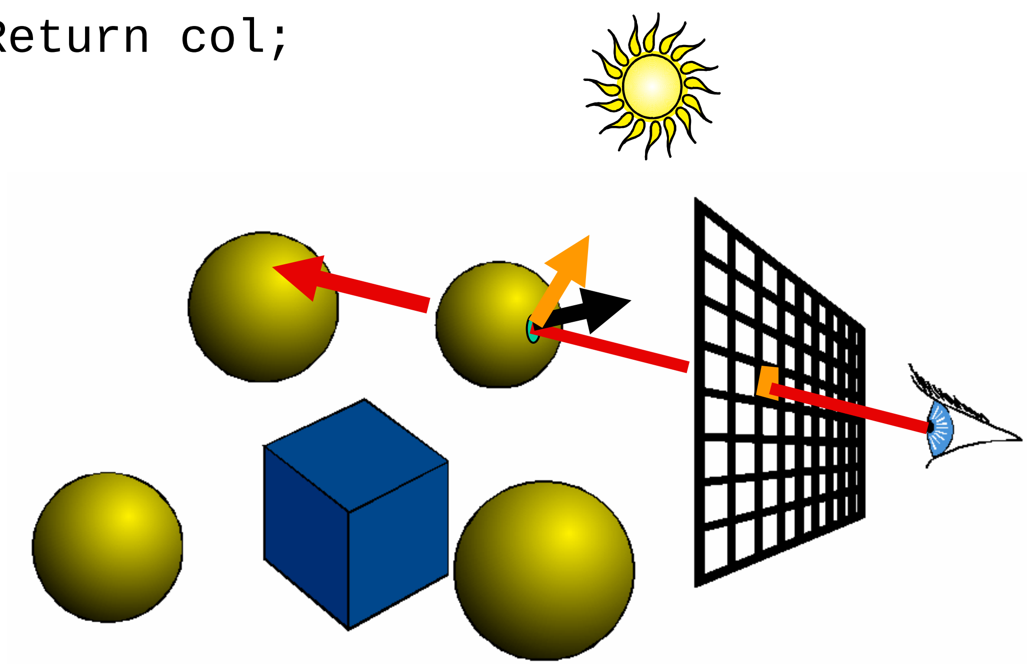
Ray Casting with diffuse shading

```
Color castRay(ray)
   Hit hit();
   For every object ob
       ob->intersect(ray, hit, tmin);
    Color col=ambient*hit->getColor();
    For every light L
       col=col+hit->getColorL()*L->getColor*
         L->getDir()->Dot3( hit->getNormal() );
    Return col;
```



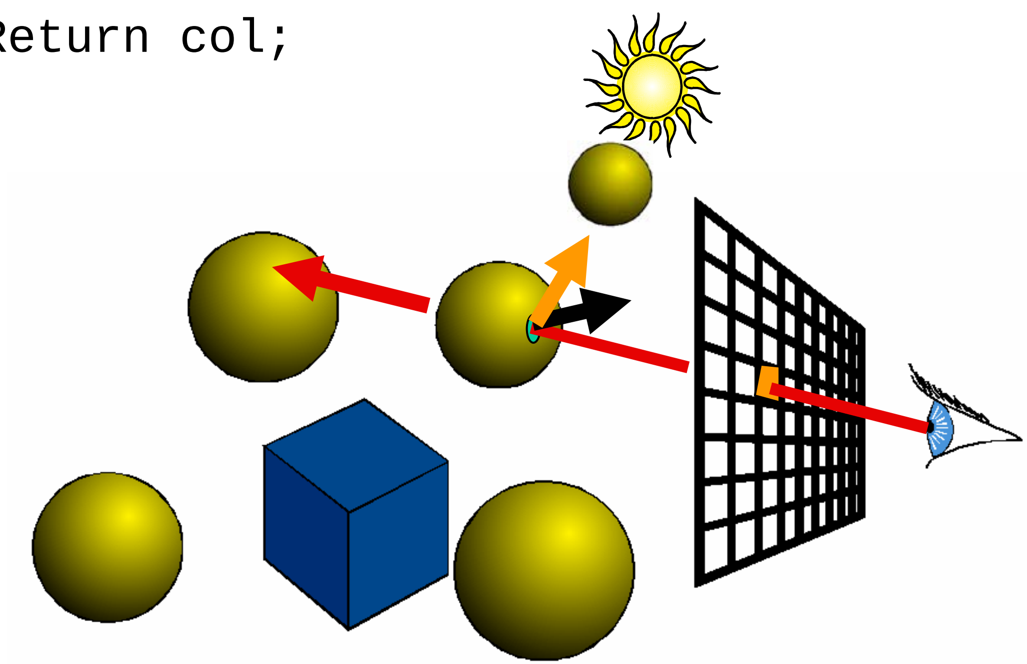
Encapsulating shading

```
Color castRay(ray)
   Hit hit();
   For every object ob
        ob->intersect(ray, hit, tmin);
    Color col=ambient*hit->getMaterial()->getDiffuse();
    For every light L
        col=col+hit->getMaterial()->shade
          (ray, hit, L->getDir(), L->getColor());
    Return col;
```



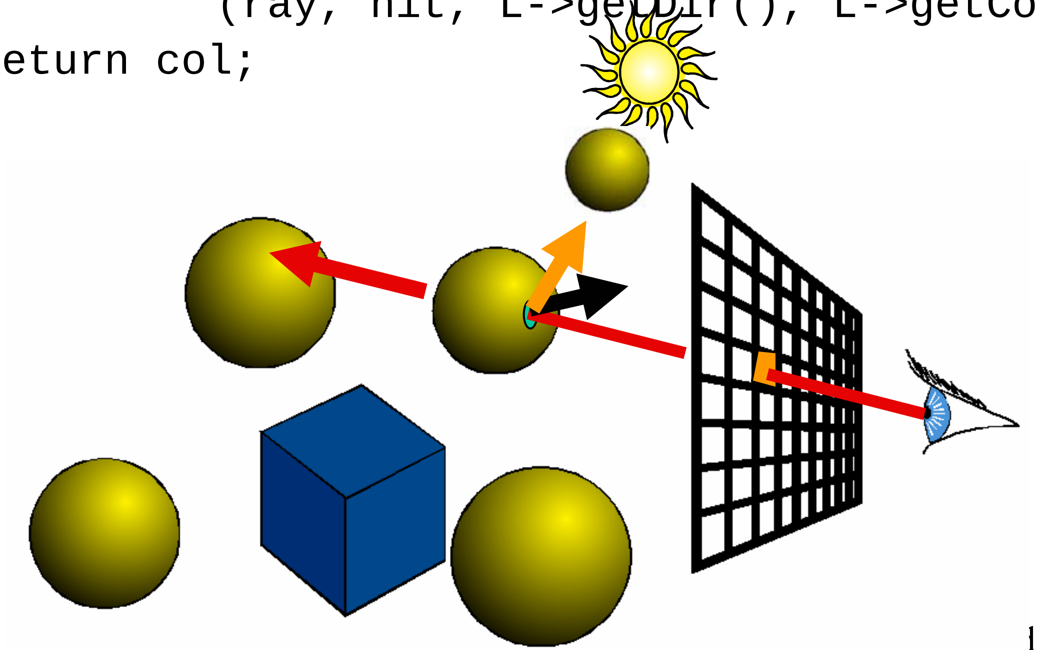
How can we add shadows?

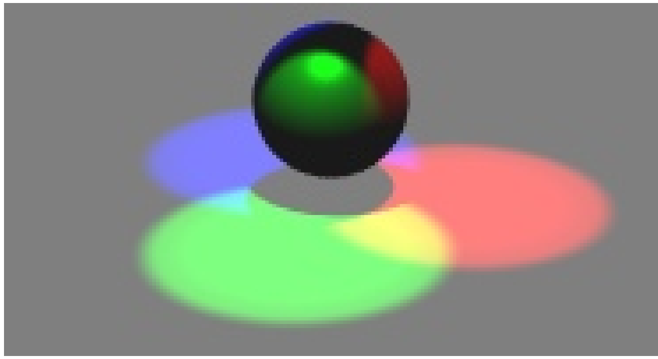
```
Color castRay(ray)
   Hit hit();
   For every object ob
        ob->intersect(ray, hit, tmin);
    Color col=ambient*hit->getMaterial()->getDiffuse();
    For every light L
        col=col+hit->getMaterial()->shade
          (ray, hit, L->getDir(), L->getColor());
    Return col;
```



Shadows

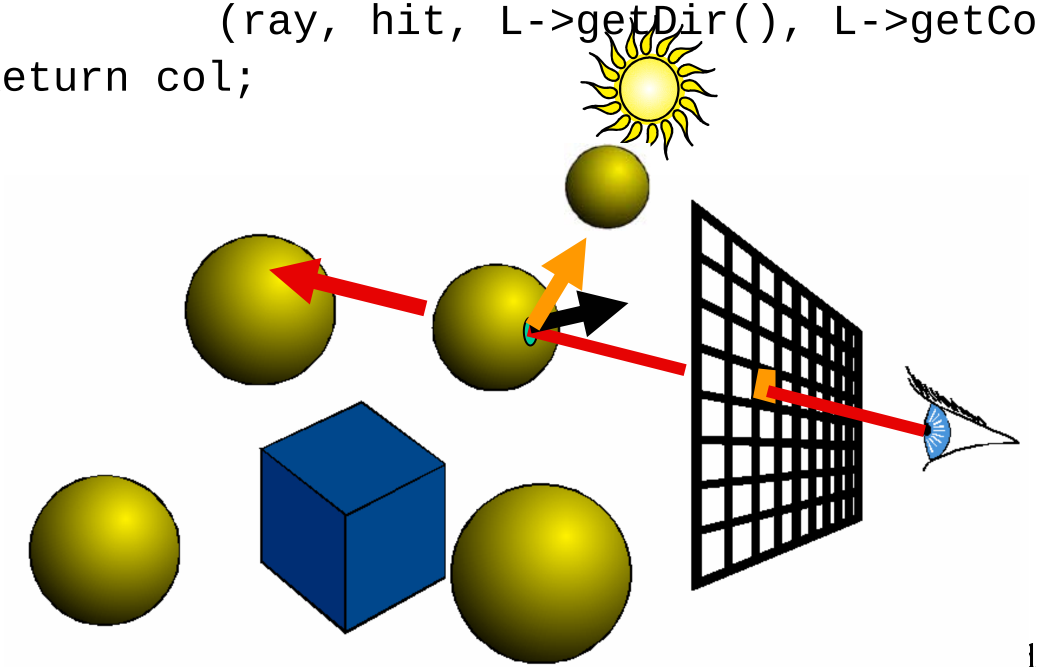
```
Color castRay(ray)
    Hit hit();
    For every object ob
        ob->intersect(ray, hit, tmin);
    Color col=ambient*hit->getMaterial()->getDiffuse();
    For every light L
        Ray ray2(hitPoint, L->getDir()); Hit hit2(L->getDist(),,)
        For every object ob
            ob->intersect(ray2, hit2, 0);
        If (hit->getT> L->getDist())
            col=col+hit->getMaterial()->shade
              (ray, hit, L->getDir(), L->getColor());
    Return col;
```

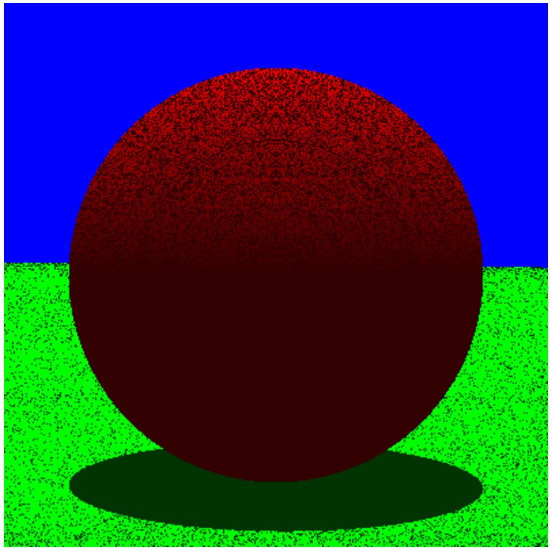




Shadows – problem?

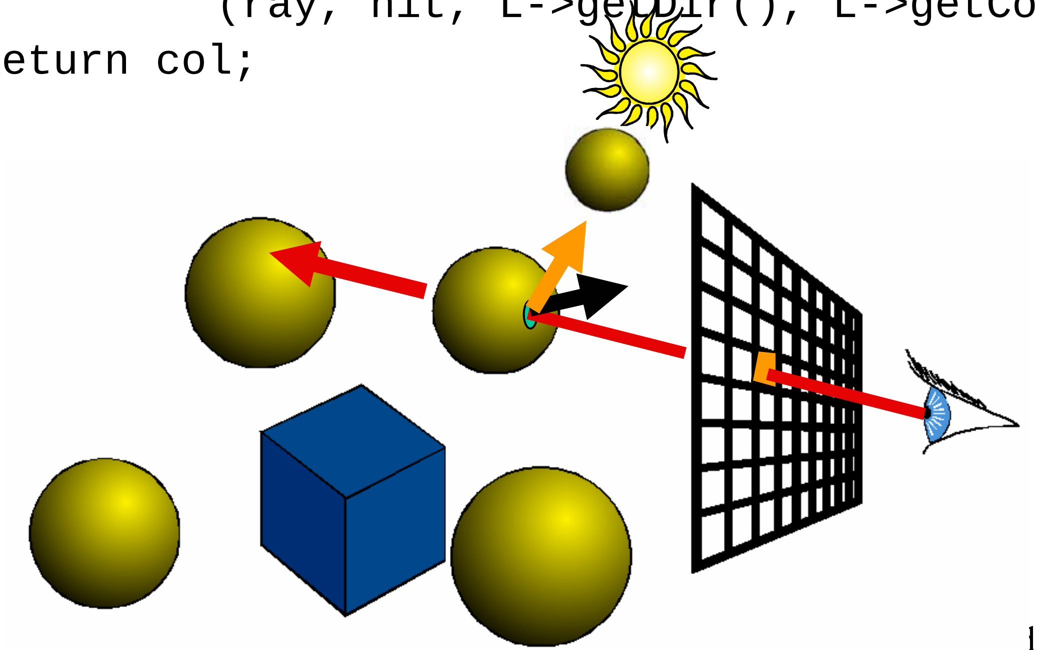
```
Color castRay(ray)
     Hit hit();
     For every object ob
              ob->intersect(ray, hit, tmin);
       Color col=ambient*hit->getMaterial()->getDiffuse();
     For every light L
          Ray ray2(hitPoint, L->getDir()); Hit hit2(L->getDist(),,)
          For every object ob
               ob->intersect(ray2, hit2, 0);
          If (hit->getT> L->getDist())
               col=col+hit->getMaterial()->shade
                 (ray, hit, L->getDir(), L->getColor());
     Return col;
```

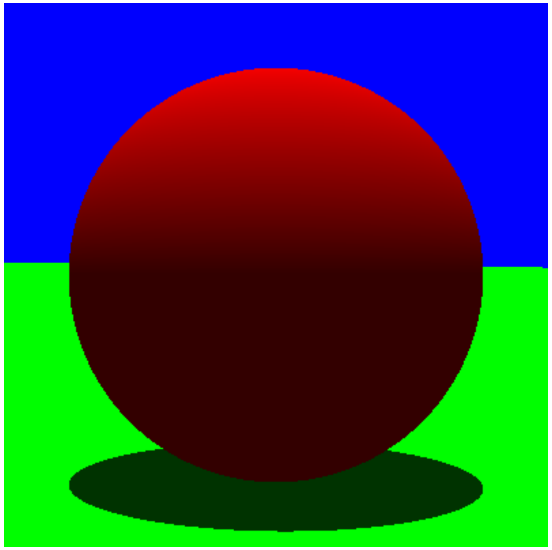




Avoiding self shadowing

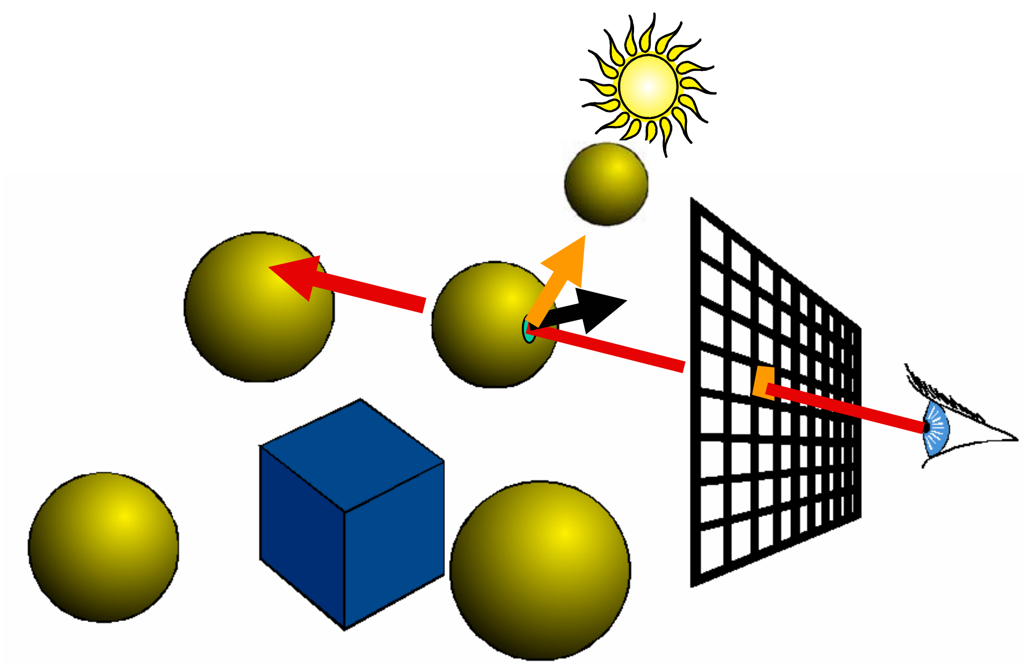
```
Color castRay(ray)
    Hit hit();
    For every object ob
          ob->intersect(ray, hit, tmin);
     Color col=ambient*hit->getMaterial()->getDiffuse();
     For every light L
          Ray ray2(hitPoint, L->getDir()); Hit hit2(L->getDist(),,)
          For every object ob
               ob->intersect(ray2, hit2, epsilon);
          If (hit->getT> L->getDist())
               col=col+hit->getMaterial()->shade
                  (ray, hit, L->getDir(), L->getColor());
     Return col;
```





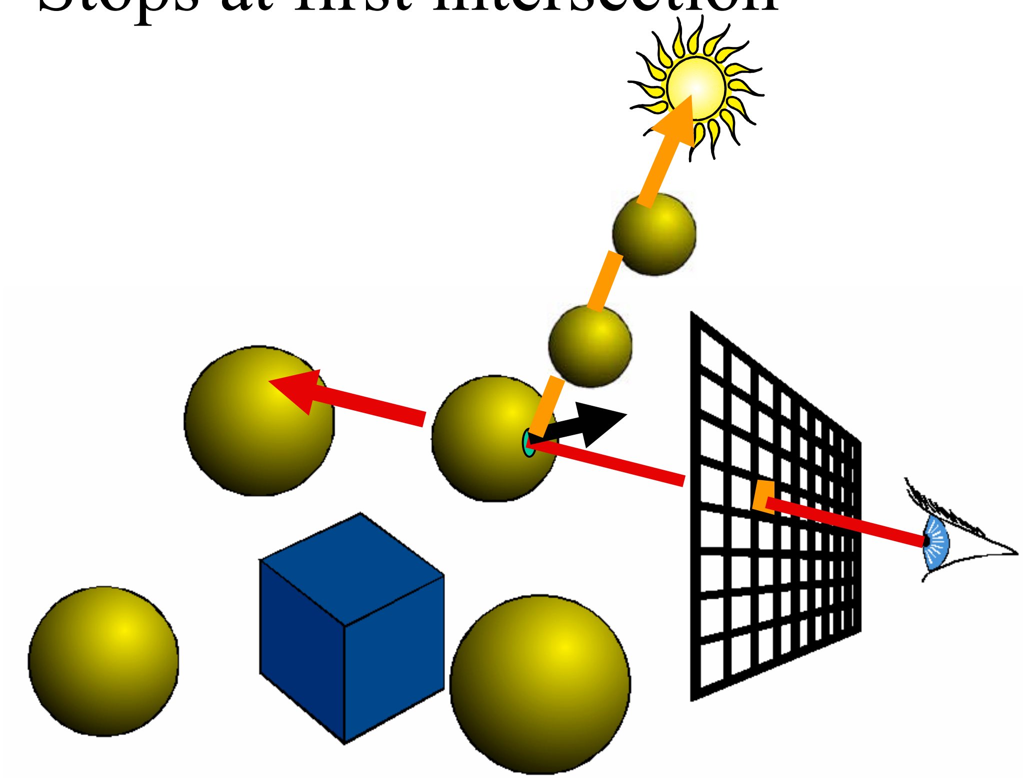
Shadow optimization

- Shadow rays are special
- How can we accelerate our code?



Shadow optimization

- We only want to know whether there is an intersection, not which one is closest
- Special routine `Object3D::intersectShadowRay()`
  - Stops at first intersection



Shadow ray casting history

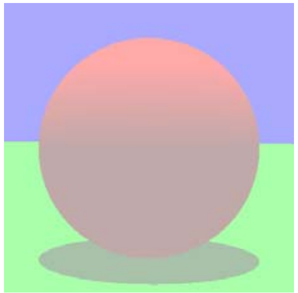
- Due to Appel [1968]
- First shadow method in graphics
- Not really used until the 80s

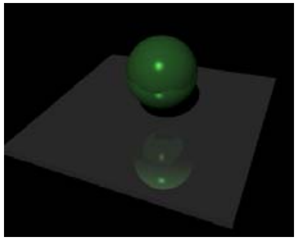
Questions?

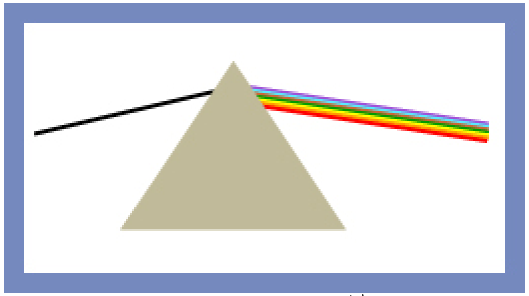
Image removed due to copyright considerations.

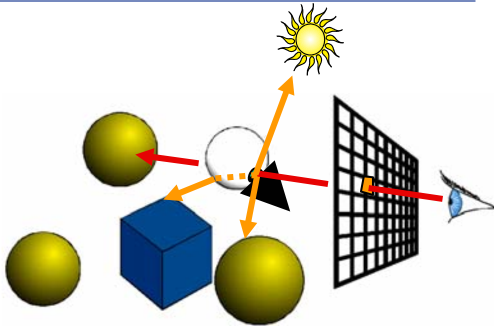
Overview of today

- Shadows
- Reflection
- Refraction
- Recursive Ray Tracing



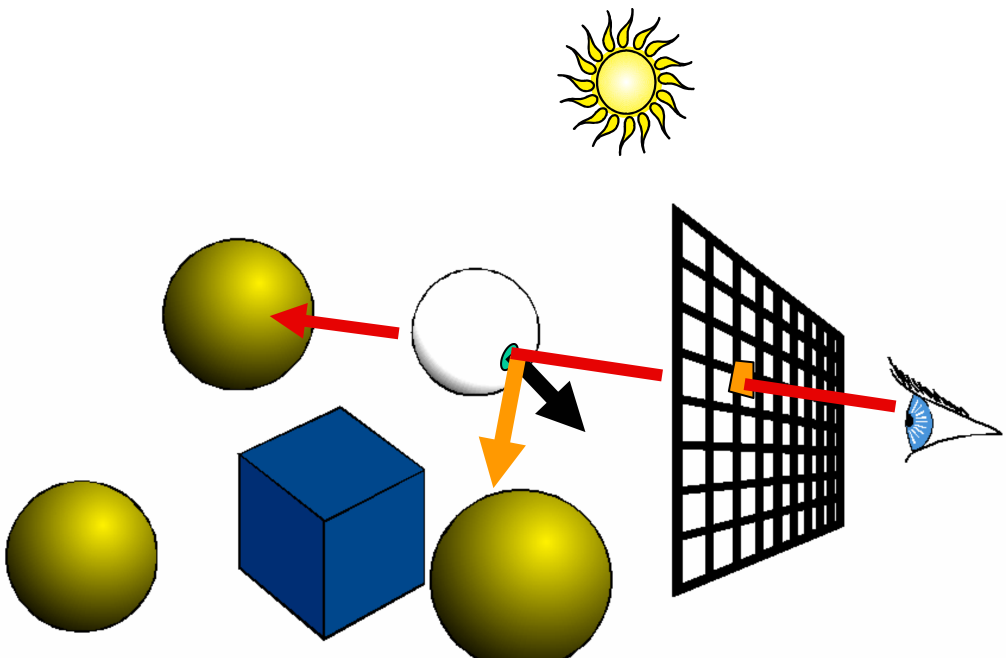


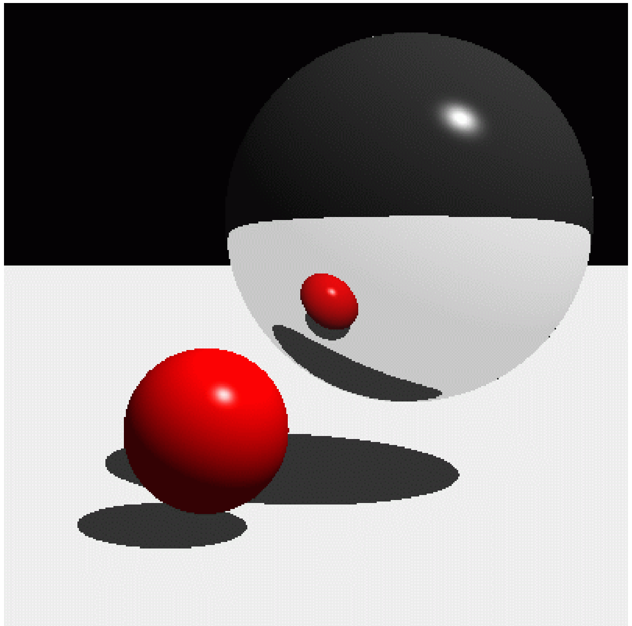




Mirror Reflection

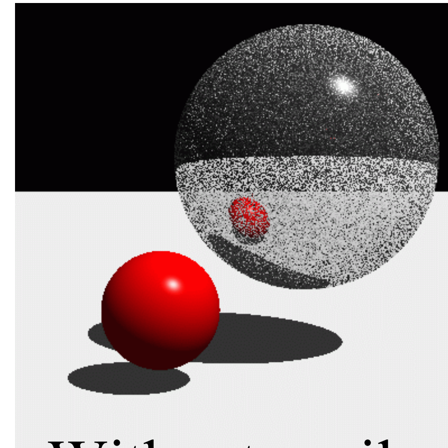
- Compute mirror contribution
- Cast ray
  - In direction symmetric wrt normal
- Multiply by reflection coefficient (color)



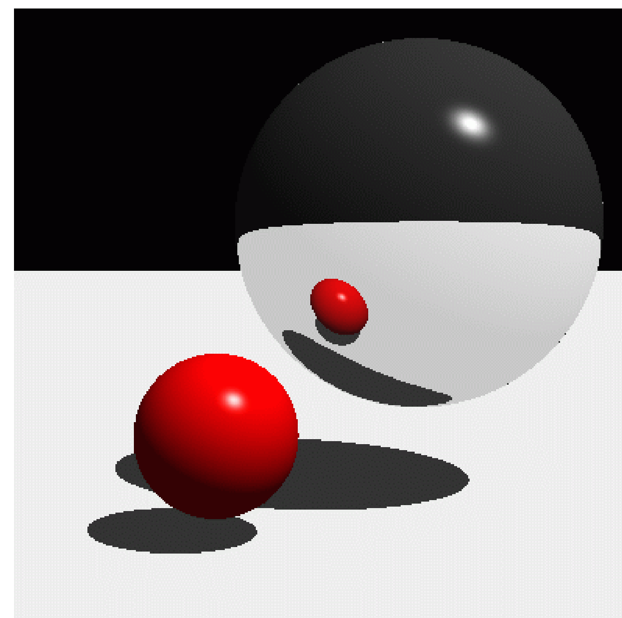


Mirror Reflection

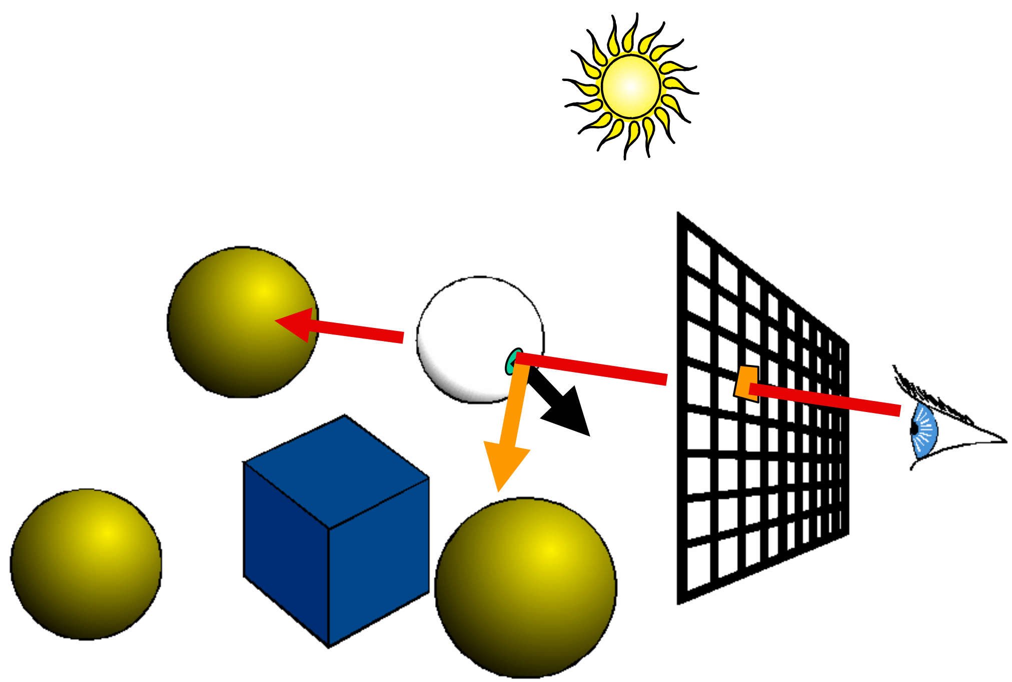
- Cast ray
  - In direction symmetric wrt normal
- Don't forget to add epsilon to the ray



Without epsilon



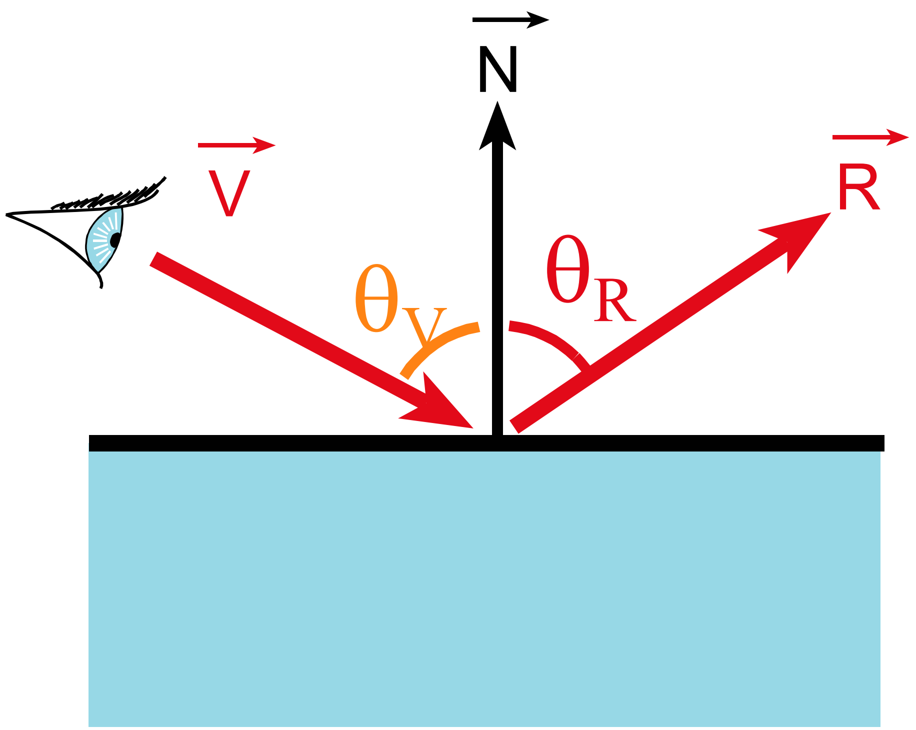
With epsilon



and Durand

Reflection

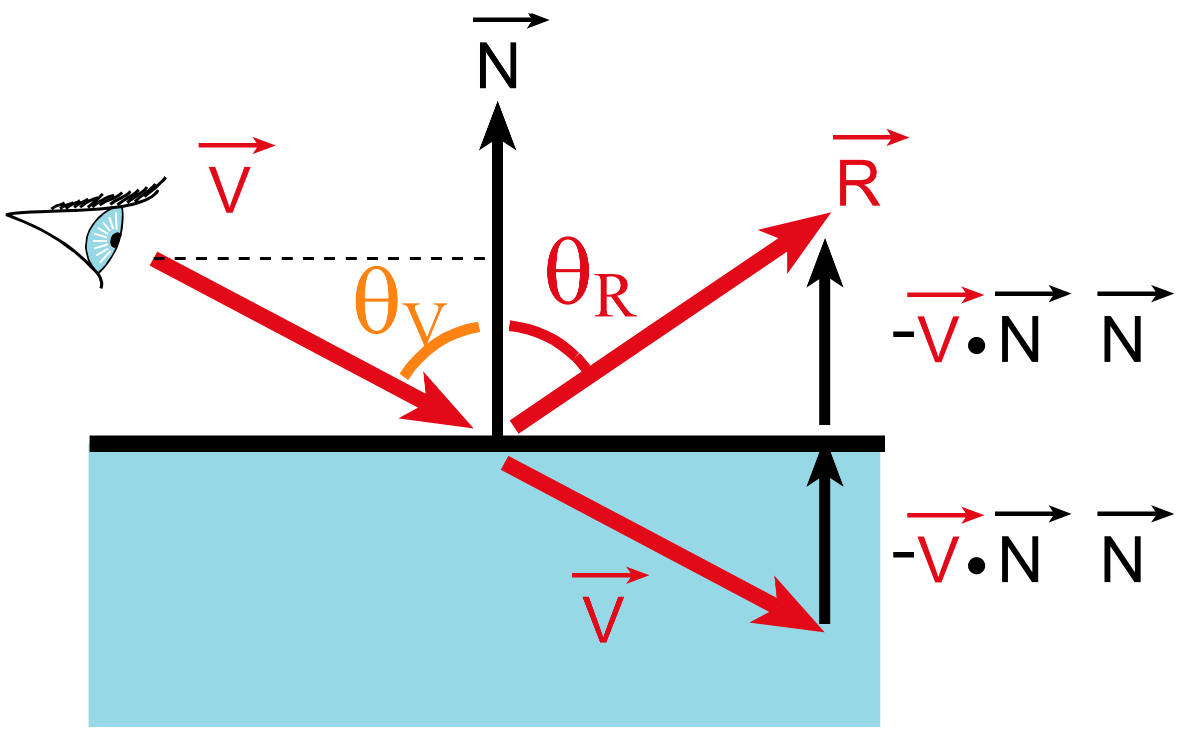
- Reflection angle = view angle



Reflection

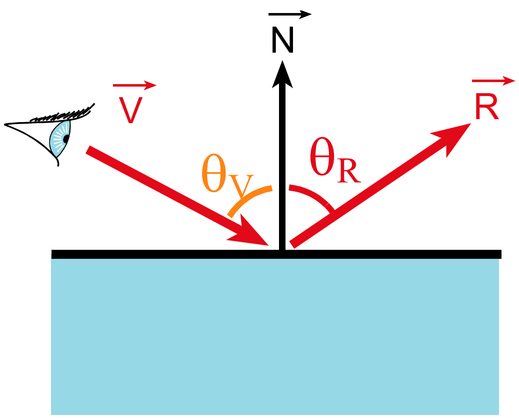
- Reflection angle = view angle

$$\vec{R} = \vec{V} - 2\left(\vec{V} \bullet \vec{N}\right)\vec{N}$$



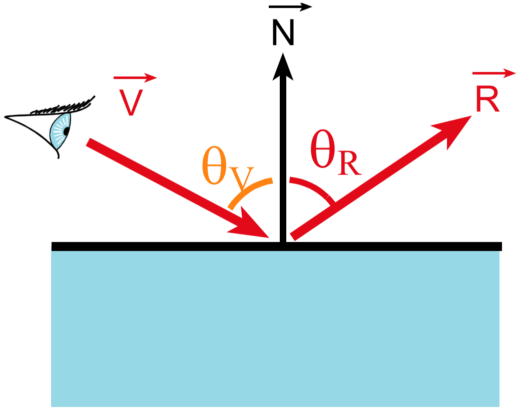
Amount of Reflection

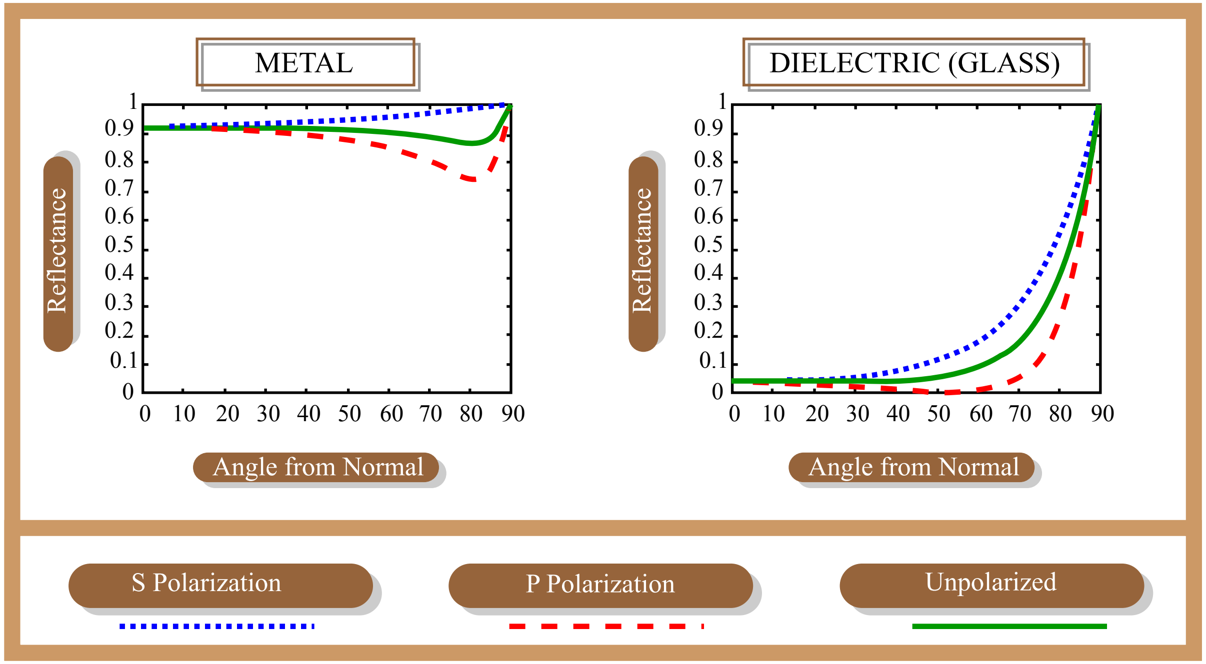
- Traditional (hacky) ray tracing
  - Constant coefficient `reflectionColor`
  - Component per component multiplication



Amount of Reflection

- More realistic:
  - Fresnel reflection term
  - More reflection at grazing angle
  - Schlick's approximation: $R(\theta)=R_0+(1-R_0)(1-\cos\theta)^5$





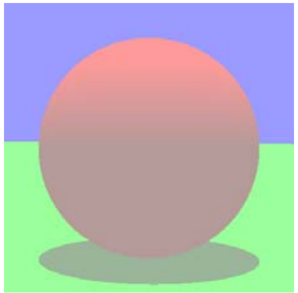
Fresnel reflectance demo

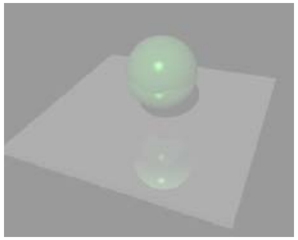
- Lafortune et al., Siggraph 1997

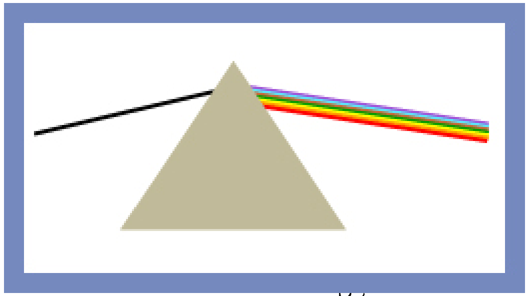
Image removed due to copyright considerations.

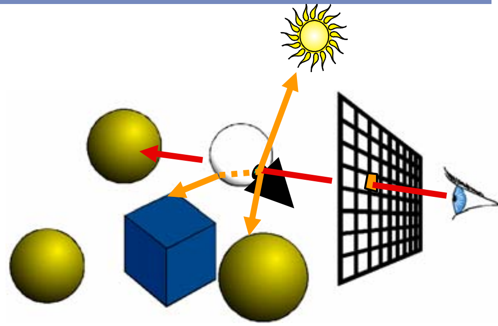
Overview of today

- Shadows
- Reflection
- Refraction
- Recursive Ray Tracing



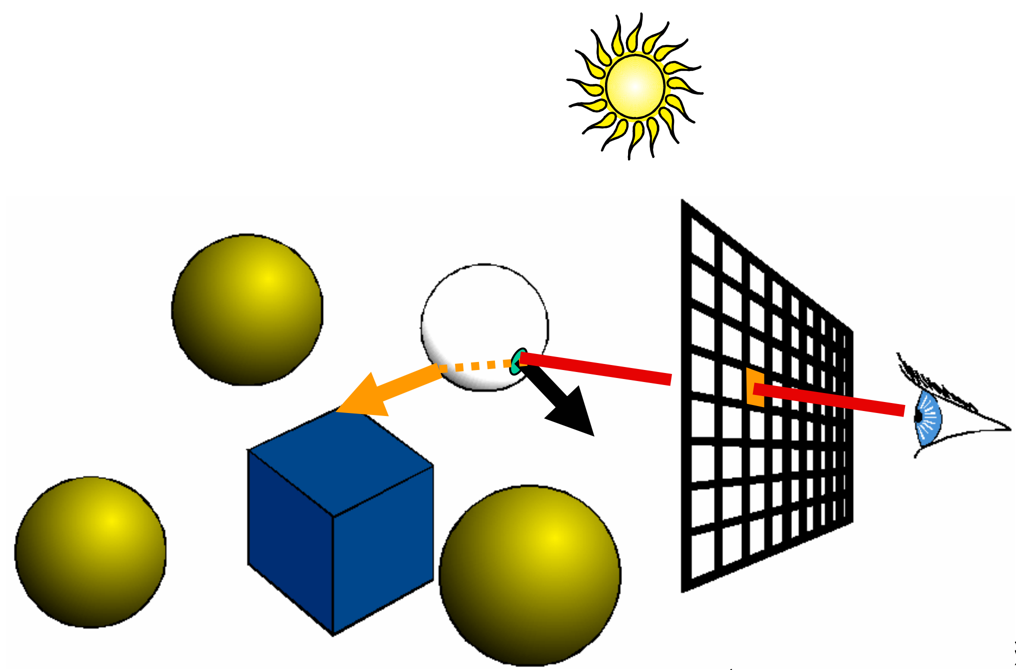






Transparency

- Compute transmitted contribution
- Cast ray
  - In refracted direction
- Multiply by transparency coefficient (color)



Qualitative refraction

- From “Color and Light in Nature” by Lynch and Livingston

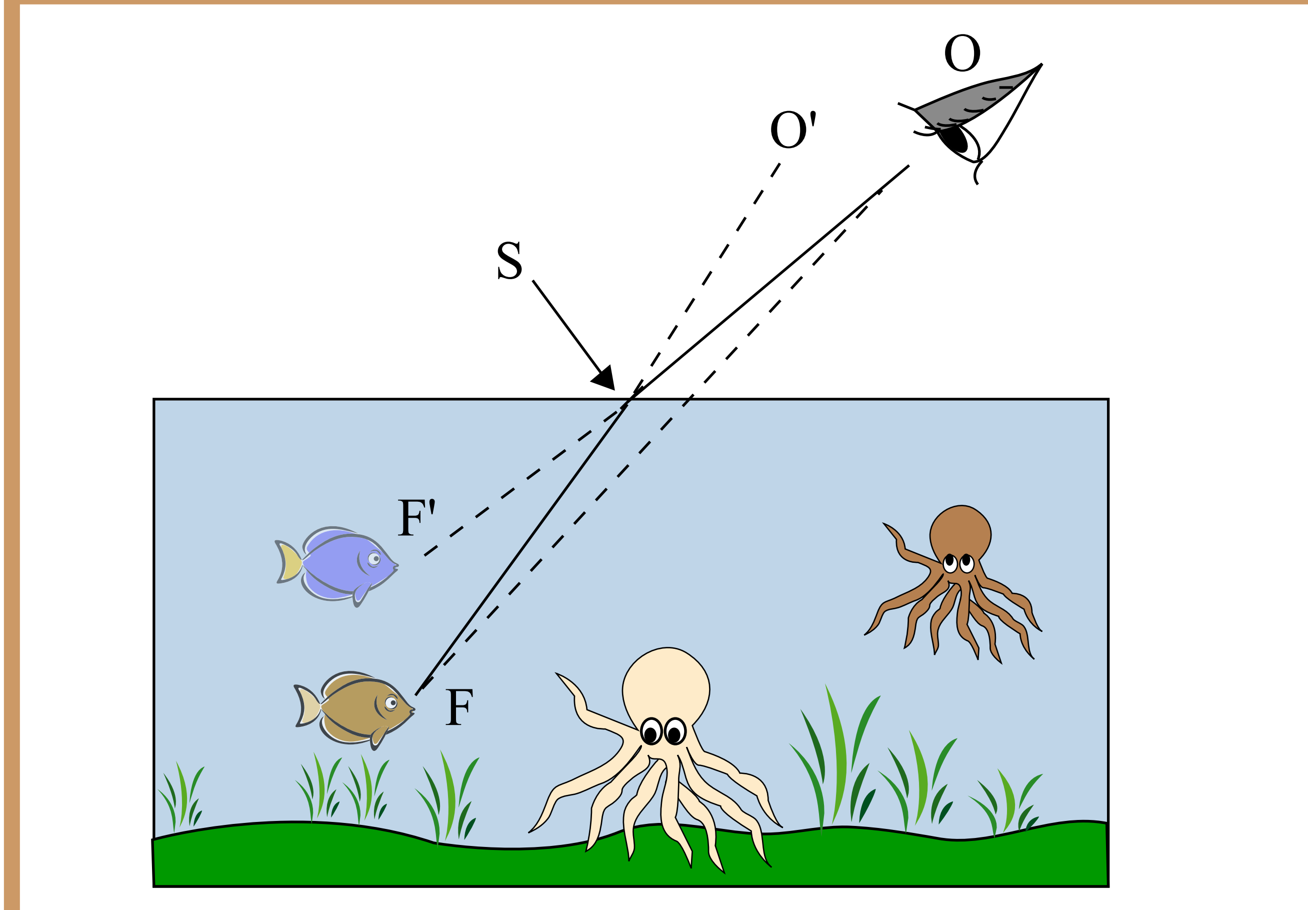
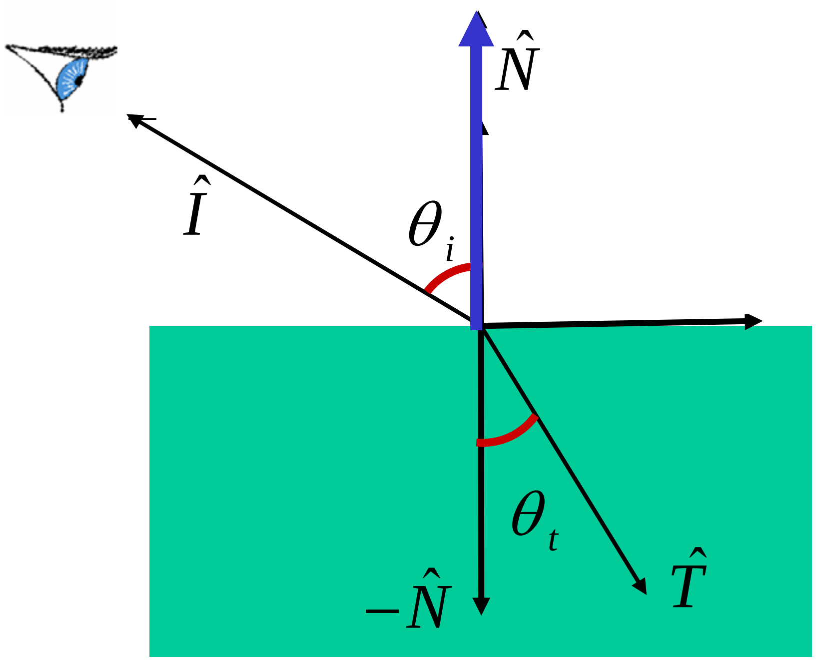


Image adapted from:
Lynch, David K. and William Livingston. *Color and Light in Nature.* Cambridge University Press. June 2001.
ISBN: 0-521-77504-3.

Refraction

Snell-Descartes Law

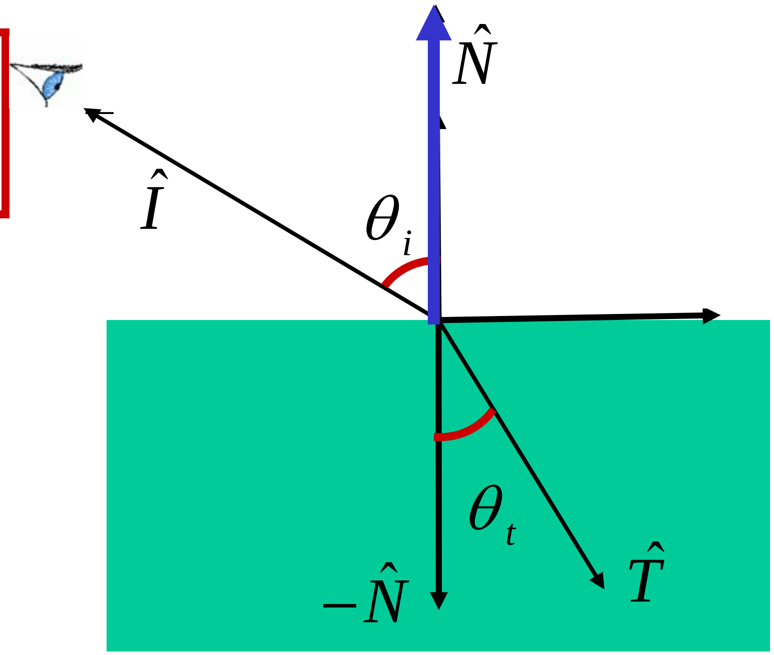


Note that I is the negative of the incoming ray

Refraction

Snell-Descartes Law

$$\boxed{\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_t}{\eta_i} = \eta_r}$$



Note that I is the negative of
the incoming ray

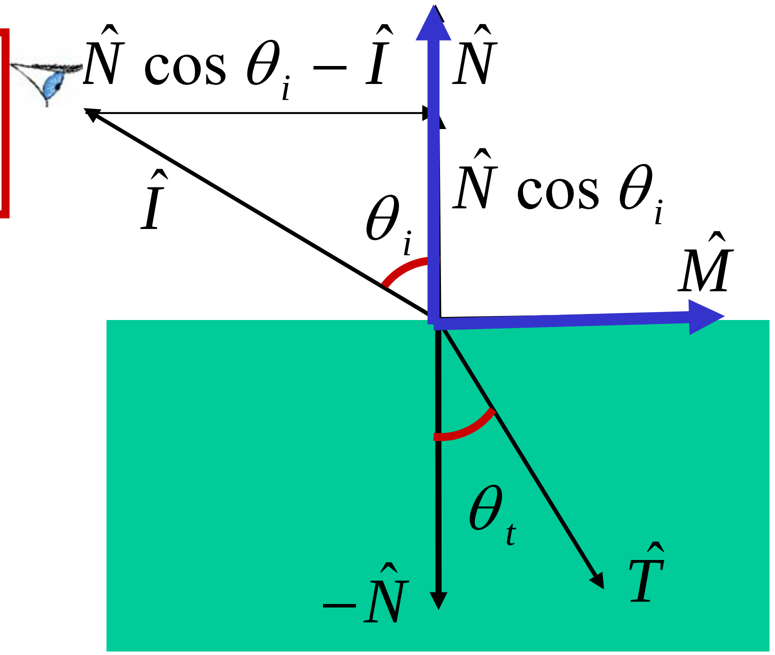
Refraction

Snell-Descartes Law

$$\boxed{\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_t}{\eta_i} = \eta_r}$$

$$\hat{T} = \sin\theta_t \hat{M} - \cos\theta_t \hat{N}$$

$$\hat{M} = \frac{(\hat{N}\cos\theta_i - \hat{I})}{\sin\theta_i}$$



Note that I is the negative of the incoming ray

Refraction

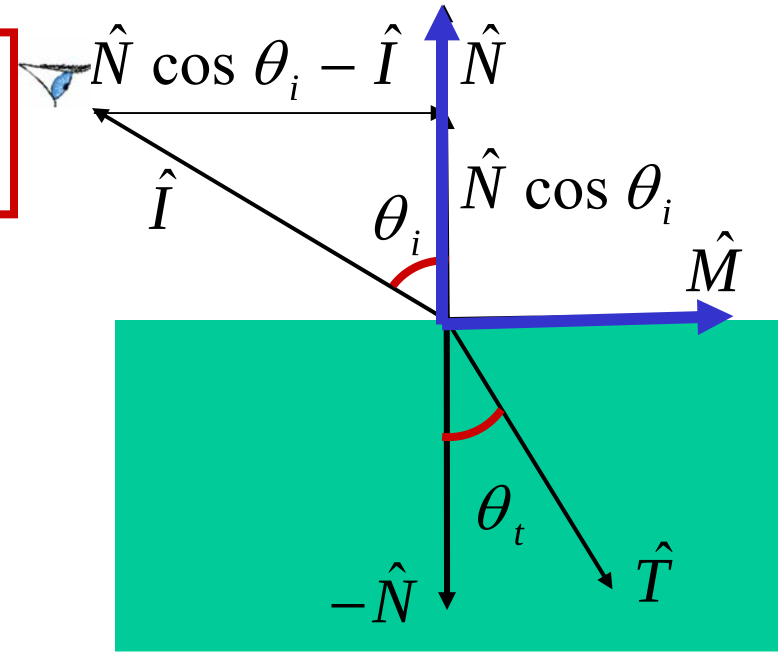
Snell-Descartes Law $\boxed{\dfrac{\sin\theta_t}{\sin\theta_i} = \dfrac{\eta_i}{\eta t} = \eta_r}$

$$\hat{T} = \sin\theta_t \hat{M} - \cos\theta_t \hat{N}$$

$$\hat{M} = \frac{(\hat{N}\cos\theta_i - \hat{I})}{\sin\theta_i}$$

$$\hat{T} = \frac{\sin\theta_t}{\sin\theta_i}(\hat{N}\cos\theta_i - \hat{I}) - \cos\theta_t \hat{N}$$

$$\hat{T} = (\eta_r \cos\theta_i - \cos\theta_t)\hat{N} - \eta_r \hat{I}$$

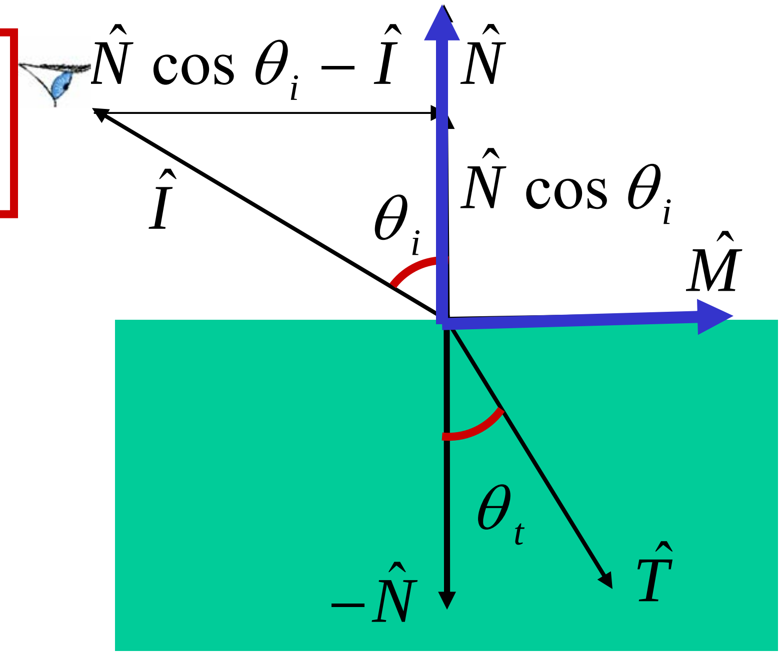


Note that I is the negative of the incoming ray

Refraction

Snell-Descartes Law

$$\boxed{\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_t}{\eta_i} = \eta_r}$$



Note that I is the negative of the incoming ray

$$\hat{T} = \sin\theta_t \hat{M} - \cos\theta_t \hat{N}$$

$$\hat{M} = \frac{(\hat{N}\cos\theta_i - \hat{I})}{\sin\theta_i}$$

$$\hat{T} = \frac{\sin\theta_t}{\sin\theta_i}(\hat{N}\cos\theta_i - \hat{I}) - \cos\theta_t \hat{N}$$

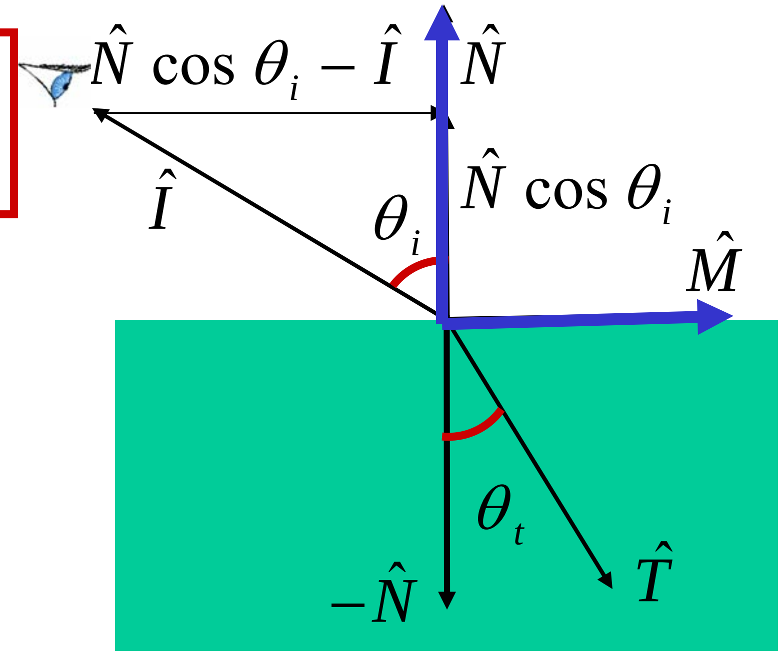
$$\hat{T} = (\eta_r\cos\theta_i - \cos\theta_t)\hat{N} - \eta_r\hat{I}$$

$$\cos\theta_i = \hat{N}\cdot\hat{I}$$

$$\cos\theta_t = \sqrt{1-\sin^2\theta_t} = \sqrt{1-\eta_r^2\sin^2\theta_i} = \sqrt{1-\eta_r^2(1-(\hat{N}\cdot\hat{I})^2)}$$

Refraction

Snell-Descartes Law $\boxed{\dfrac{\sin\theta_t}{\sin\theta_i} = \dfrac{\eta_i}{\eta_t} = \eta_r}$



Note that I is the negative of the incoming ray

$$\hat{T} = \sin\theta_t \hat{M} - \cos\theta_t \hat{N}$$

$$\hat{M} = \frac{(\hat{N}\cos\theta_i - \hat{I})}{\sin\theta_i}$$

$$\hat{T} = \frac{\sin\theta_t}{\sin\theta_i}(\hat{N}\cos\theta_i - \hat{I}) - \cos\theta_t \hat{N}$$

$$\hat{T} = (\eta_r\cos\theta_i - \cos\theta_t)\hat{N} - \eta_r \hat{I}$$

$$\cos\theta_i = \hat{N}\cdot\hat{I}$$

$$\cos\theta_t = \sqrt{1-\sin^2\theta_t} = \sqrt{1-\eta_r^2\sin^2\theta_i} = \sqrt{1-\eta_r^2(1-(\hat{N}\cdot\hat{I})^2)}$$

$$\boxed{\hat{T} = \left(\eta_r(\hat{N}\cdot\hat{I}) - \sqrt{1-\eta_r^2(1-(\hat{N}\cdot\hat{I})^2)}\right)\hat{N} - \eta_r\hat{I}}$$

Don't forget to normalize

Total internal reflection when the square root is imaginary

Total internal reflection

- From “Color and Light in Nature” by Lynch and Livingstone

THE OPTICAL MANHOLE. LIGHT FROM THE HORIZON (ANGLE OF INCIDENCE = 90^{O}) IS REFRACTED DOWNWARD AT AN ANGLE OF 48.6^{O}. THIS COMPRESSES THE SKY INTO A CIRCLE WITH A DIAMETER OF 97.2^{O} INSTEAD OF ITS USUAL 180^{O}.

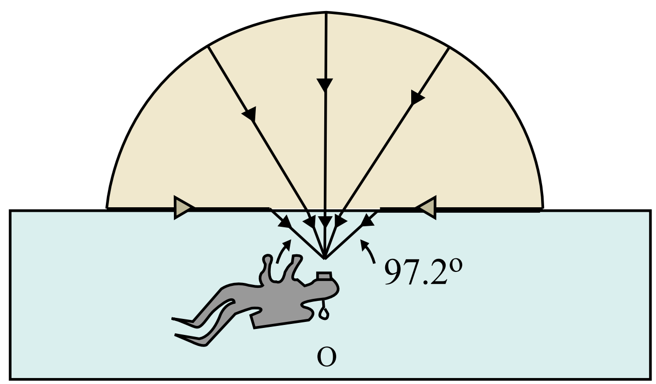


Image adapted from:
Lynch, David K. and William Livingston. *Color and Light in Nature.* Cambridge University Press. June 2001.
ISBN: 0-521-77504-3.

Cool refraction demo

- Enright, D., Marschner, S. and Fedkiw, R.,

Image removed due to copyright considerations.

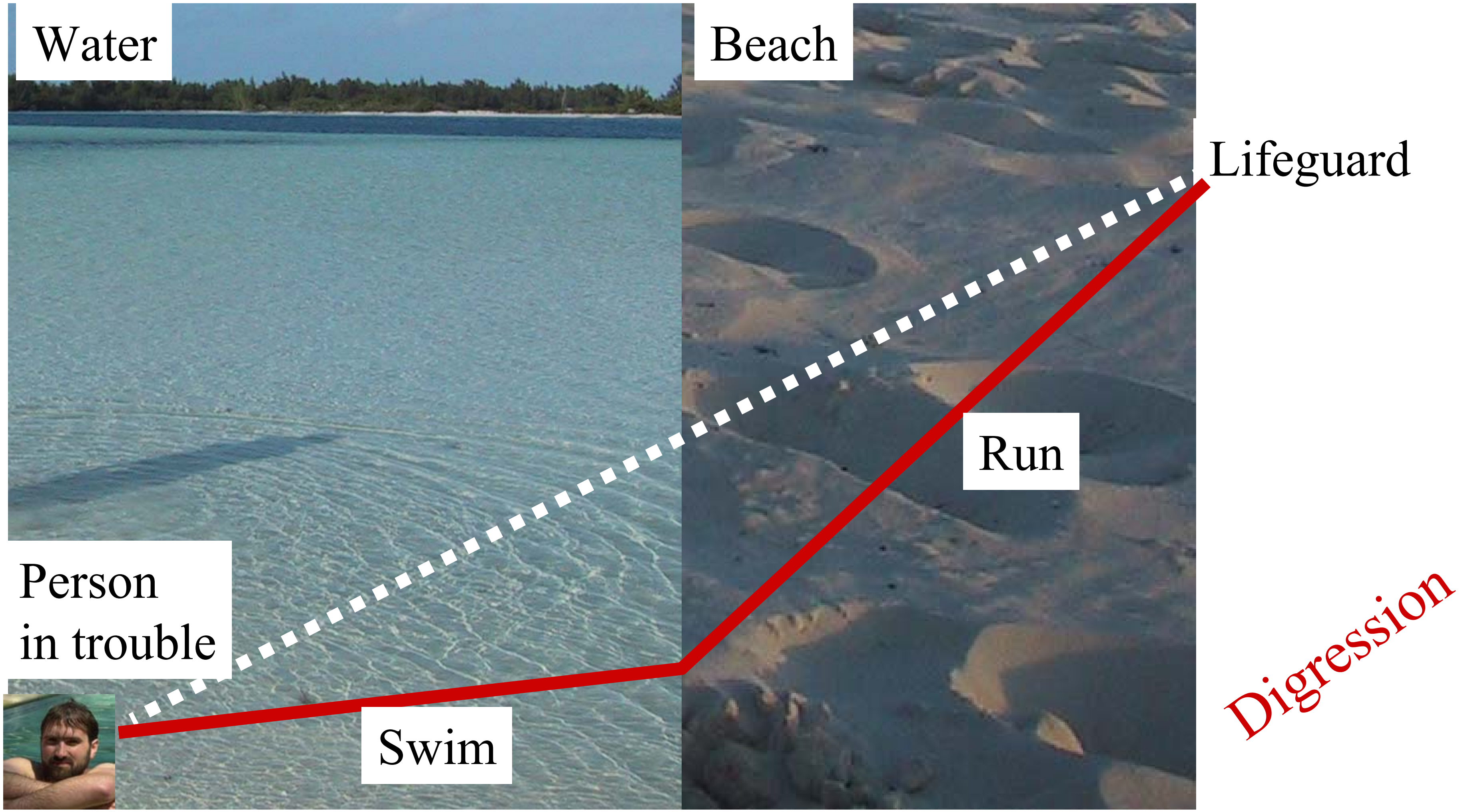
Cool refraction demo

- Enright, D., Marschner, S. and Fedkiw, R.,

Image removed due to copyright considerations.

Refraction and the lifeguard problem

- Running is faster than swimming



Wavelength

- Refraction is wavelength-dependent
- Newton’s experiment
- Usually ignored in graphics

Rainbow

Digression

- Refraction depends on wavelength
- Rainbow is caused by refraction+internal reflection+refraction
- Maximum for angle around 42 degrees

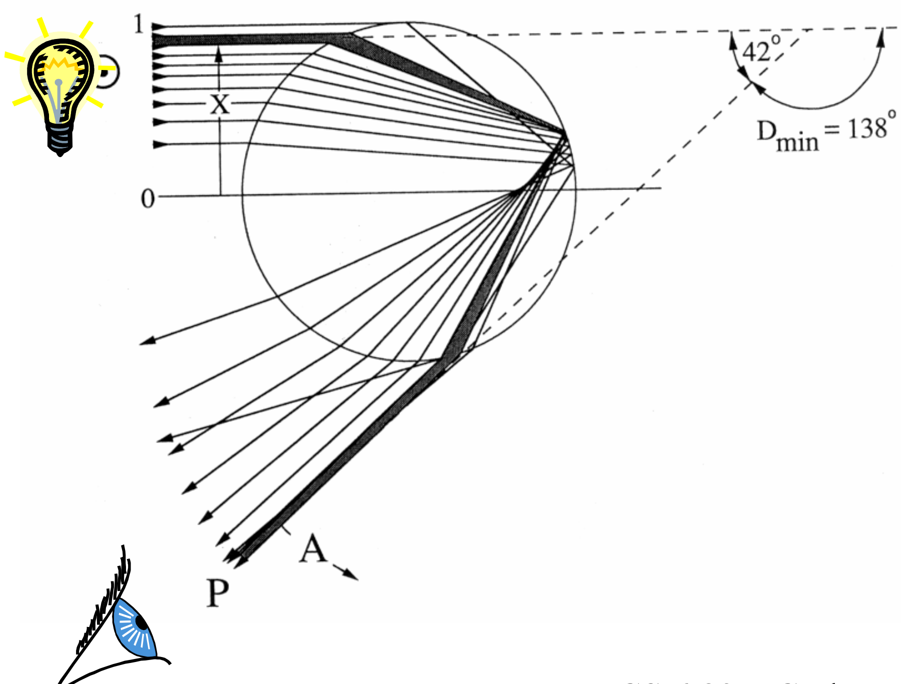
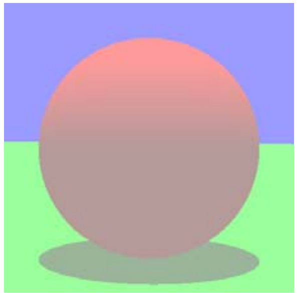
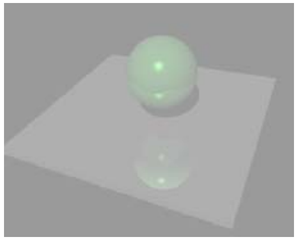


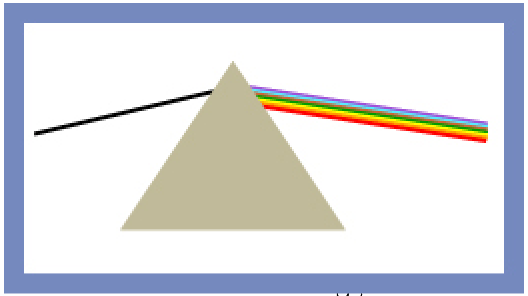
Image removed due to copyright considerations.

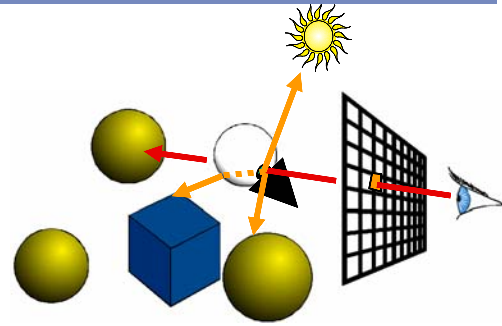
Overview of today

- Shadows
- Reflection
- Refraction
- Recursive Ray Tracing

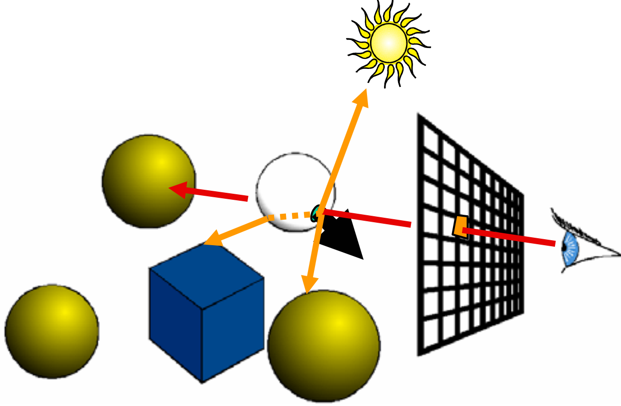






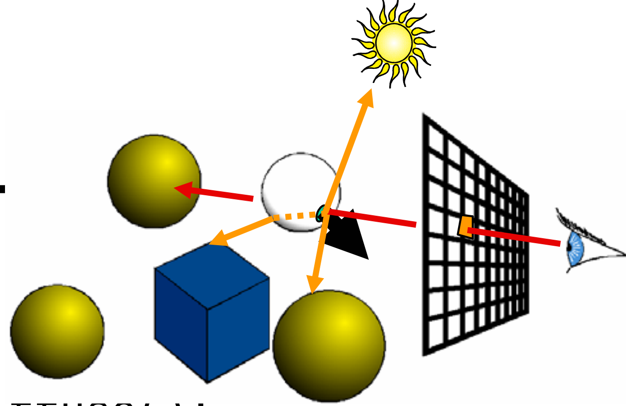


Recap: Ray Tracing



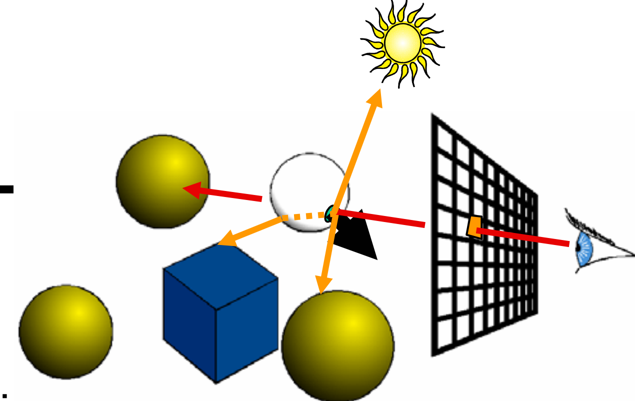
```
traceRay
   Intersect all objects
   Ambient shading
   For every light
      Shadow ray
      shading
   If mirror
      Trace reflected ray
   If transparent
      Trace transmitted ray
```

Recap: Ray Tracing



```
Color traceRay(ray)
   For every object ob
        ob->intersect(ray, hit, tmin);
    Color col=ambient*hit->getMaterial()->getDiffuse();
    For every light L
        If ( not castShadowRay( hit->getPoint(), L->getDir())
            col=col+hit->getMaterial()->shade
              (ray, hit, L->getDir(), L->getColor());
    If (hit->getMaterial()->isMirror())
        Ray rayMirror (hit->getPoint(),
          getMirrorDir(ray->getDirection(), hit->getNormal());
        Col=col+hit->getMaterial->getMirrorColor()
          *traceRay(rayMirror, hit2);
    If (hit->getMaterial()->isTransparent()
        Ray rayTransmitted(hit->getPoint(),
          getRefracDir(ray, hit->getNormal(), curentRefractionIndex,
          hit->Material->getRefractionIndex());
        Col=col+hit->getMaterial->getTransmittedColor()
          *traceRay(rayTransmitted, hit3);
    Return col;
```

Does it end?



```
Color traceRay(ray)
   For every object ob
        ob->intersect(ray, hit, tmin);
    Color col=ambient*hit->getMaterial()->getDiffuse();
    For every light L
        If ( not castShadowRay( hit->getPoint(), L->getDir())
            col=col+hit->getMaterial()->shade
              (ray, hit, L->getDir(), L->getColor());
    If (hit->getMaterial()->isMirror())
        Ray rayMirror (hit->getPoint(),
          getMirrorDir(ray->getDirection(), hit->getNormal());
        Col=col+hit->getMaterial->getMirrorColor()
          *traceRay(rayMirror, hit2);
    If (hit->getMaterial()->isTransparent()
        Ray rayTransmitted(hit->getPoint(),
          getRefracDir(ray, hit->getNormal(), curentRefractionIndex,
          hit->Material->getRefractionIndex());
        Col=col+hit->getMaterial->getTransmittedColor()
          *traceRay(rayTransmitted, hit3);
    Return col;
```

Avoiding infinite recursion

Stopping criteria:

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

Usually do both

```
Color traceRay(ray)
      For every object ob
                ob->intersect(ray, hit, tmin);
        Color col=ambient*hit->getMaterial()->getDiffuse();
        For every light L
                If ( not castShadowRay( hit->getPoint(), L->getDir())
                        col=col+hit->getMaterial()->shade
                            (ray, hit, L->getDir(), L->getColor());
        If (hit->getMaterial()->isMirror())
                Ray rayMirror (hit->getPoint(),
                    getMirrorDir(ray->getDirection(), hit->getNormal());
                Col=col+hit->getMaterial->getMirrorColor()
                    *traceRay(rayMirror);
        If (hit->getMaterial()->isTransparent()
                Ray rayTransmitted(hit->getPoint(),
                    getRefracDir(ray, hit->getNormal(),
                    curentRefractionIndex, hit->Material-
                    >getRefractionIndex());
                Col=col+hit->getMaterial->getTransmittedColor()
                    *traceRay(rayTransmitted);
        Return col;
```

Recursion for reflection

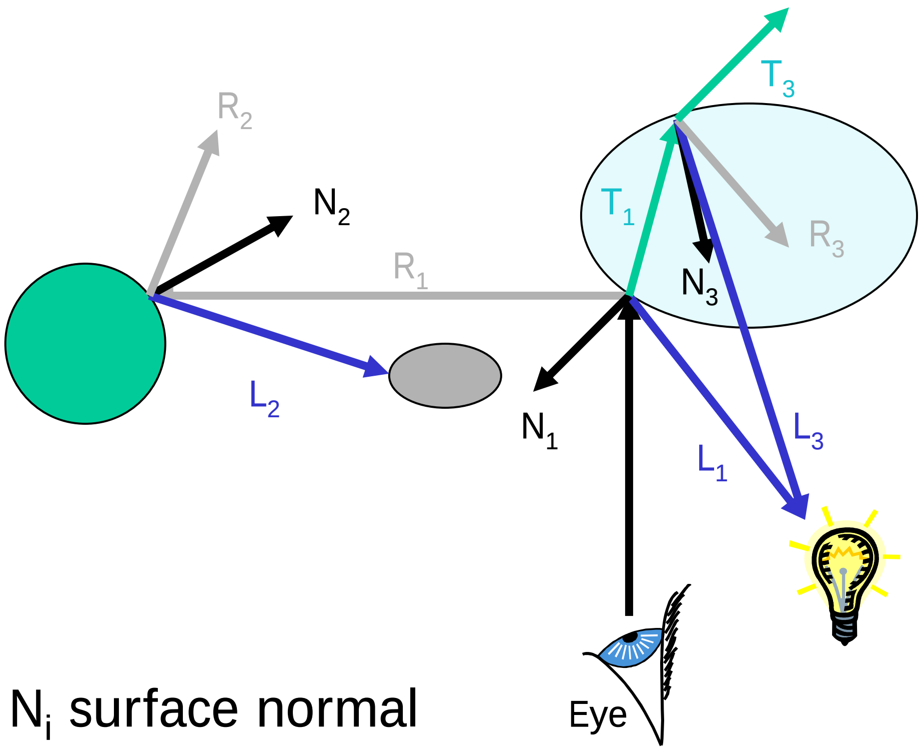
(Images removed due to copyright considerations.)

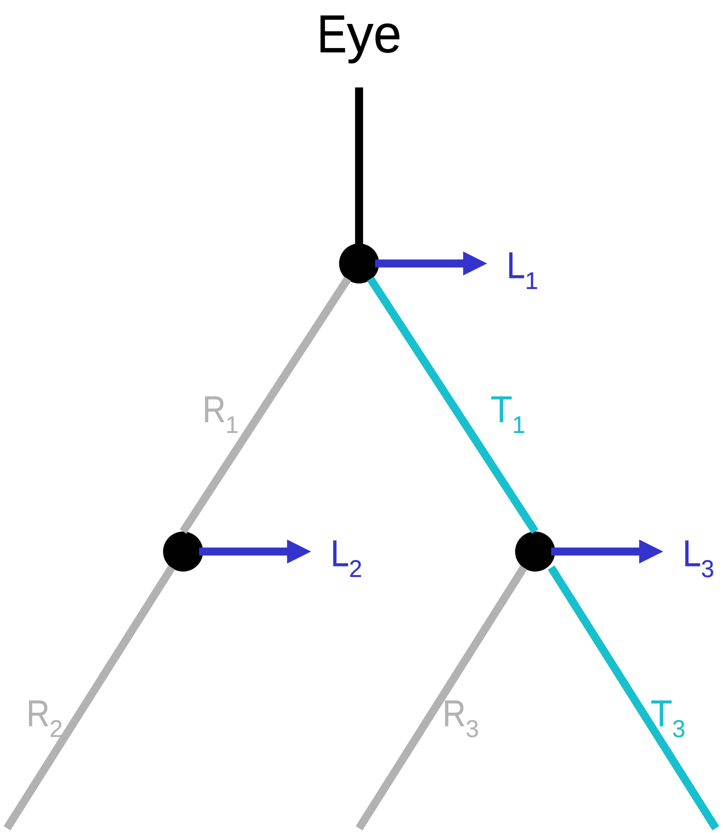
0 recursion

1 recursion

2 recursions

The Ray Tree





N_i surface normal

R_i reflected ray

L_i shadow ray

T_i transmitted (refracted) ray

Ray Tracing History

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

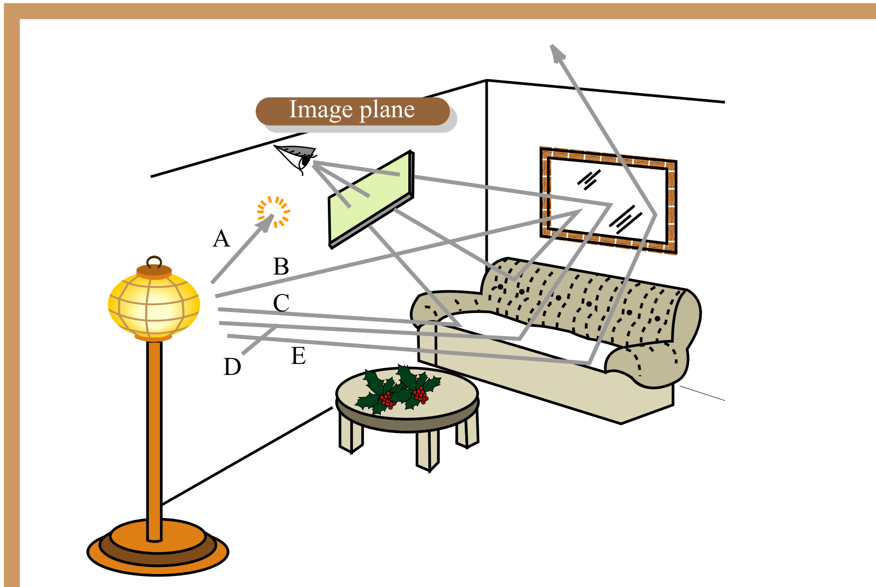
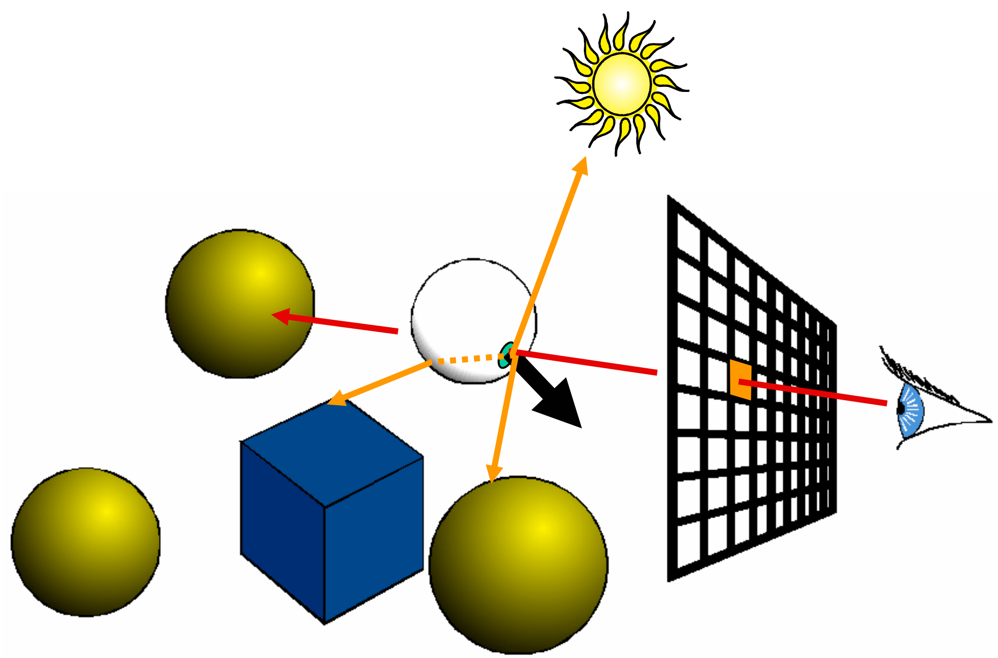


Image adapted from:
Appel, A. "Some Techniques for Shading Machine Renderings of Solids." *Proceedings of the Spring Joint Computer Conference*. 1968, pp. 37-45.

Image removed due to copyright considerations.

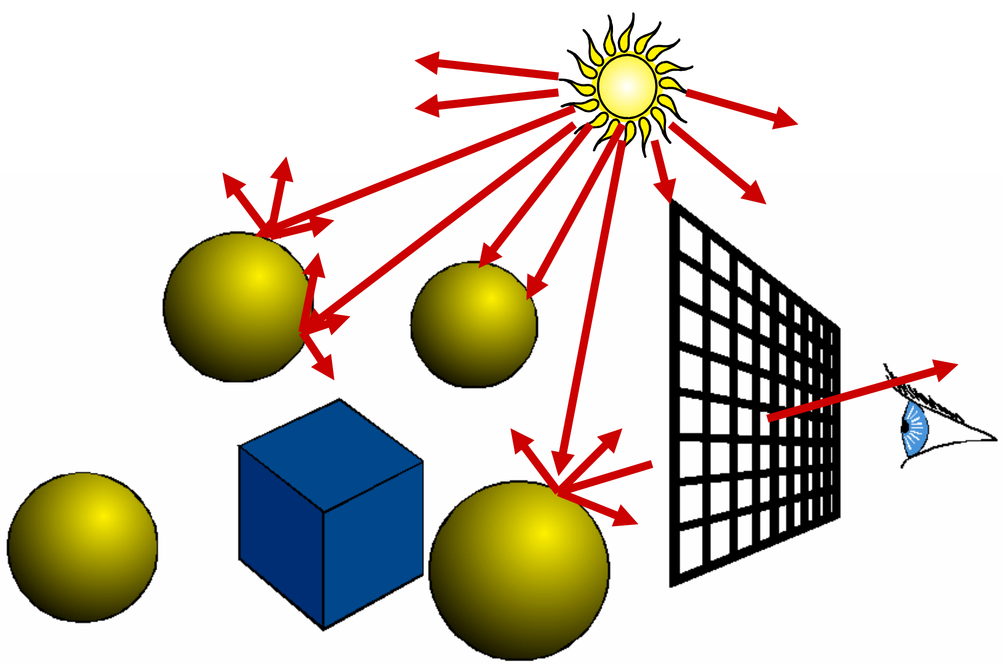
Does Ray Tracing simulate physics?

- Photons go from the light to the eye, not the other way
- What we do is backward ray tracing



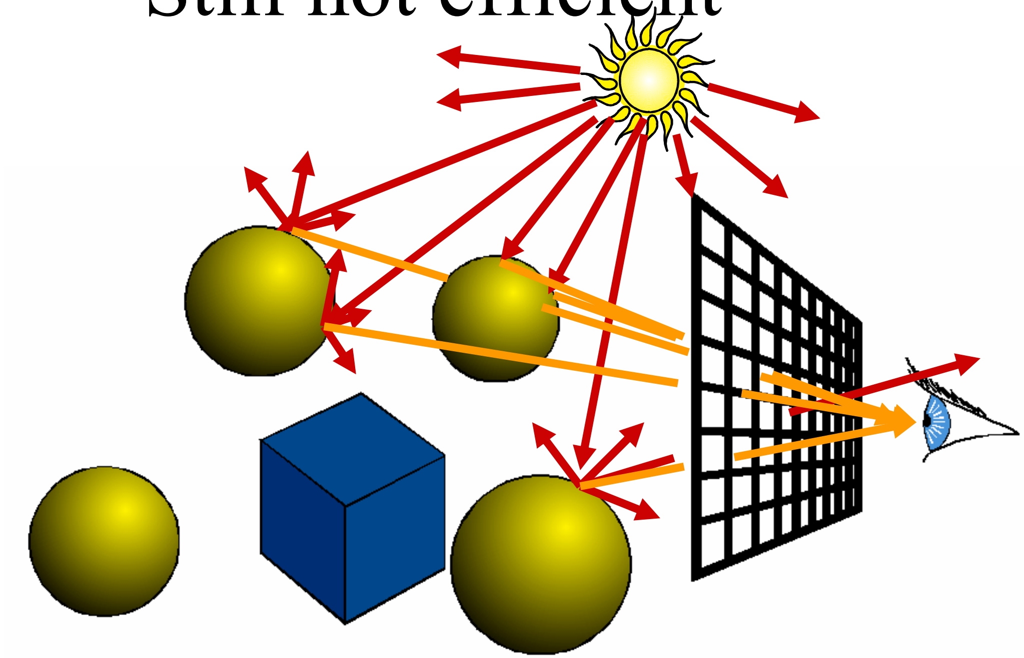
Forward ray tracing

- Start from the light source
- But low probability to reach the eye
  - What can we do about it?



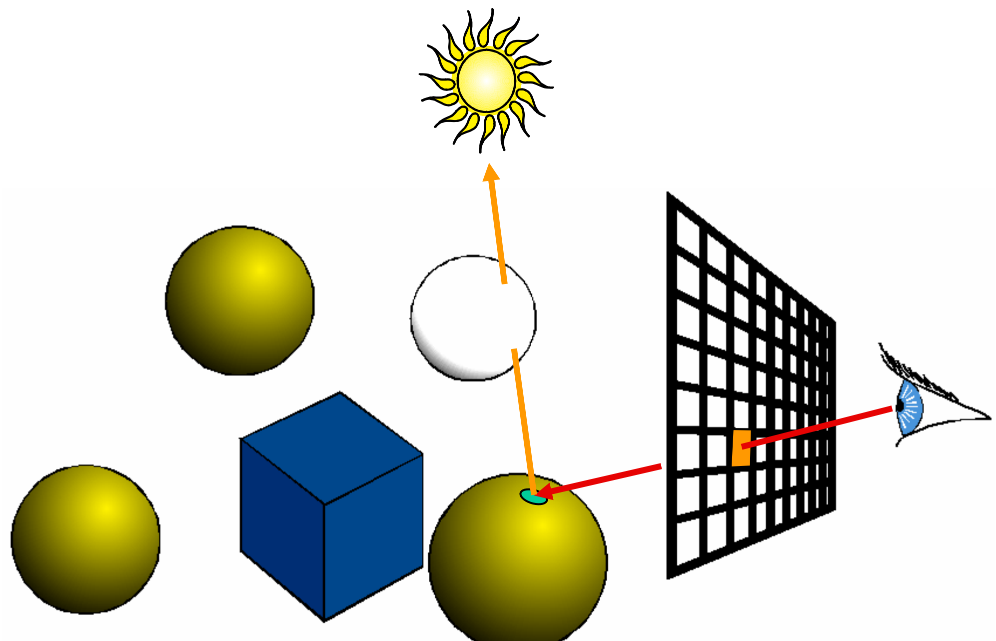
Forward ray tracing

- Start from the light source
- But low probability to reach the eye
  - What can we do about it?
  - Always send a ray to the eye
- Still not efficient



Does Ray Tracing simulate physics?

- Ray Tracing is full of dirty tricks
- e.g. shadows of transparent objects
  - Dirtiest: opaque
  - Still dirty: multiply by transparency color
    - But then no refraction



Correct transparent shadow

Animation by Henrik Wann Jensen

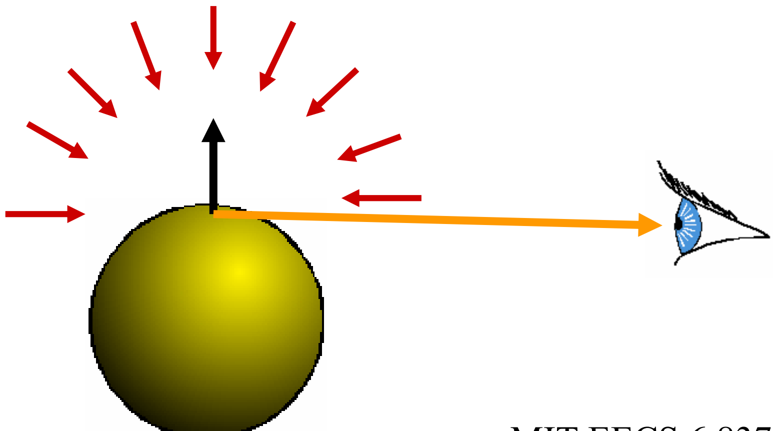
Digression

Using advanced refraction technique
(refraction for illumination is usually not handled that well)

(Image removed due to copyright considerations.)

The Rendering equation

- Clean mathematical framework for light-transport simulation
- We'll see that in November
- At each point, outgoing light in one direction is the integral of incoming light in all directions multiplied by reflectance property



Thursday

- Reflectance properties, shading and BRDF
- Guest lecture