MIT 6.837 - Ray Tracing

Ray Tracing

MIT EECS 6.837

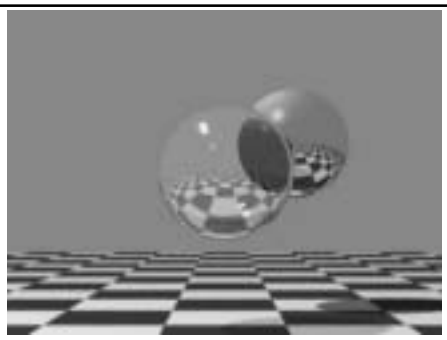
Most slides are taken from Frédo Durand and Barb Cutler

Some slides courtesy of Leonard McMillan

Ray Tracing

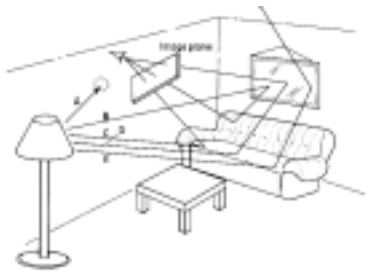
- Ray Tracing kills two birds with one stone:
  - Solves the Hidden Surface Removal problem
  - Evaluates an improved global illumination model
    - shadows
    - ideal specular reflections
    - ideal specular refractions
  - Enables direct rendering of a large variety of geometric primitives
- Book: A. Glassner, An Introduction to Ray Tracing
- Web: `http://www.cs.cf.ac.uk/Ray.Tracing`

Ray Tracing



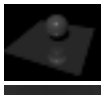
Recursive ray tracing: Turner Whitted, 1980

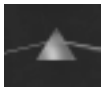
Backward Tracing

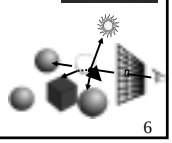


Overview of today

- Shadows
- Reflection
- Refraction
- Recursive Ray Tracing



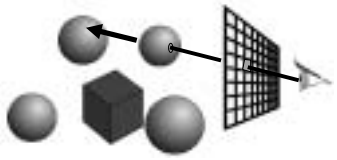




Ray Casting (a.k.a. Ray Shooting)

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

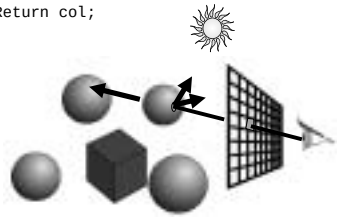
- Complexity?
  - O(n * m)
  - n: number of objects, m: number of pixels



7

Ray Casting with diffuse shading

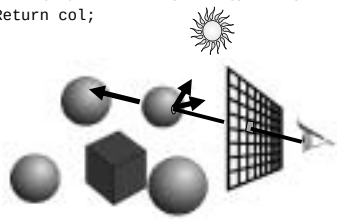
```
Color castRay(ray)
  Hit hit();
  For every object ob
      ob->intersect(ray, hit, tmin);
   Color col=ambient*hit->getColor();
   For every light L
      col=col+hit->getColorL()*L->getColor*
        L->getDir()->Dot3( hit->getNormal() );
   Return col;
```



8

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;
```



9

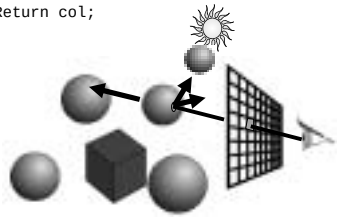
Questions?

- Image computed using the RADIANCE system by Greg Ward

10

How can we add shadows?

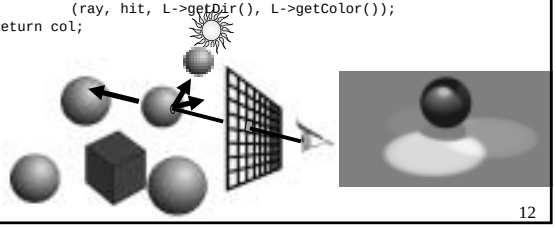
```
Color castRay(ray)
  Hit hit();
  For every object ob
      ob->intersect(ray, hit, tmin);
   Color col=ambient*hit->getMaterial()->getDiffuse();
   For every light L
      col=col+hit->getMaterial()->shade
        (ray, hit, L->getDir(), L->getColor());
   Return col;
```



11

Shadows

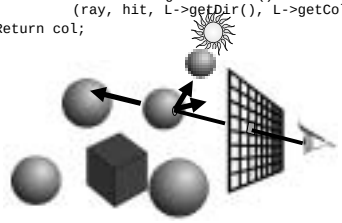
```
Color castRay(ray)
    Hit hit();
    For every object ob
        ob->intersect(ray, hit, tmin);
     Color col=ambient*hit->getMaterial()->getDiffuse();
     For every light L
        Ray ray2(hitPoint, L->getDir()); Hit hit2(L->getDist(),,)
        For every object ob
            ob->intersect(ray2,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



12

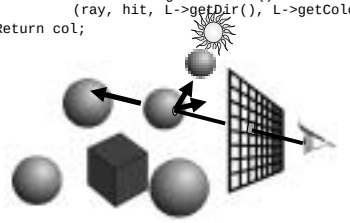
Shadows – problem?

```
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;
```



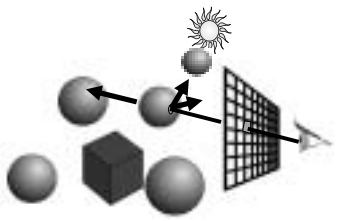
Avoiding self shadowing

```
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;
```



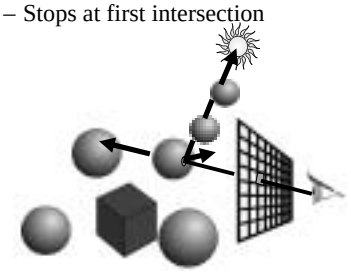
Shadow optimization

- 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

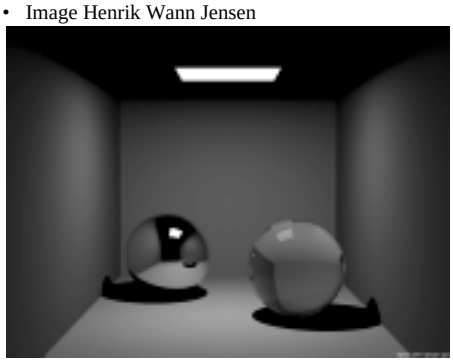


Shadow ray casting history

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

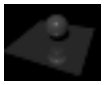
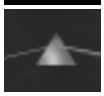
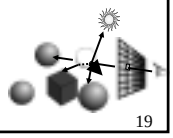
Questions?

- Image Henrik Wann Jensen



Overview of today

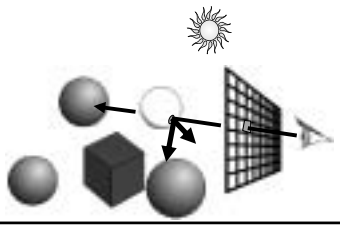
- Shadows
- Reflection
- Refraction
- Recursive Ray Tracing

19

Mirror Reflection

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

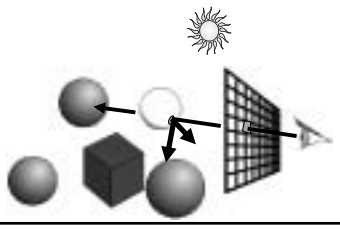



20

Mirror Reflection

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

Without epsilon

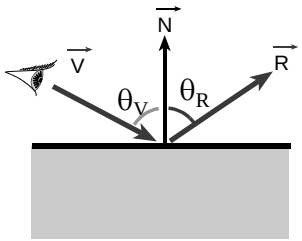



With epsilon

21

Reflection

- Reflection angle = view angle

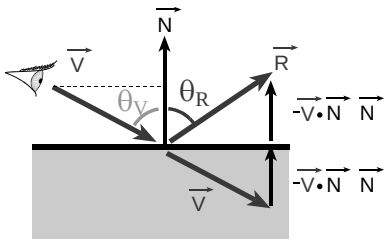


22

Reflection

- Reflection angle = view angle

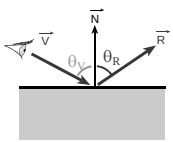
$$\vec{R} = \vec{V} - 2\left(\vec{V} \bullet \vec{N}\right)\vec{N}$$



23

Amount of Reflection

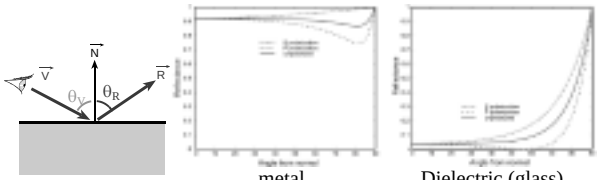
- Traditional (hacky) ray tracing
  - Constant coefficient `reflectionColor`
  - Component per component multiplication



24

Amount of Reflection

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



metal Dielectric (glass)

25

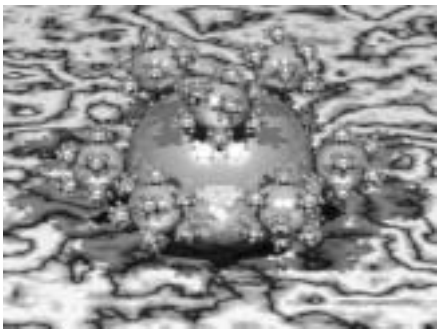
Fresnel reflectance demo

- Lafortune et al., Siggraph 1997

26

Questions?

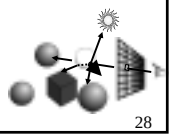
- Image by Henrik Wann Jensen



27

Overview of today

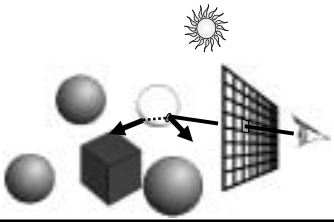
- Shadows
- Reflection
- Refraction
- Recursive Ray Tracing



28

Transparency

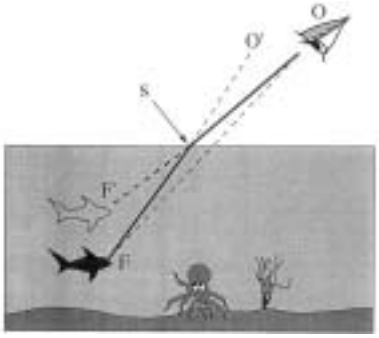
- Compute transmitted contribution
- Cast ray
  - In refracted direction
- Multiply by transparency coefficient (color)



29

Qualitative refraction

- From "Color and Light in Nature" by Lynch and Livingston



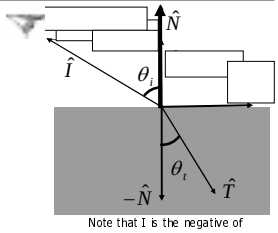
30

Refraction

Fig. 3. Refraction causes the ruler to appear bent in a glass of water.

Refraction

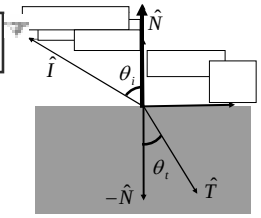
Snell-Descartes Law



Refraction

Snell-Descartes Law

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

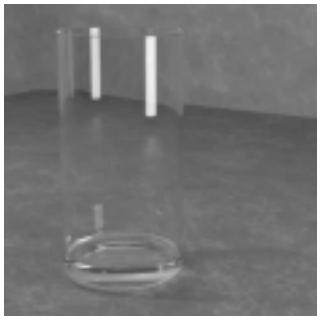


Total internal reflection

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

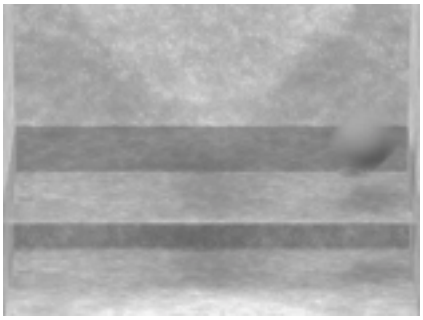
Cool refraction demo

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



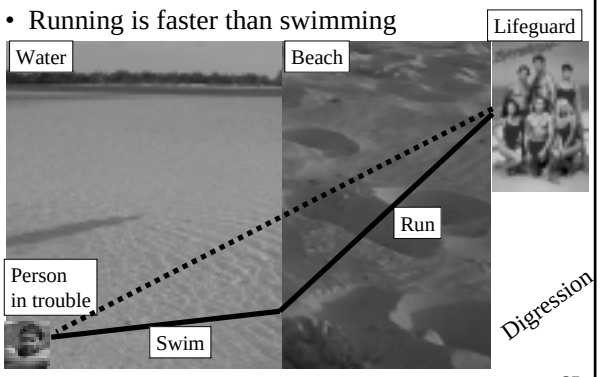
Cool refraction demo

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



Refraction and the lifeguard problem

- Running is faster than swimming



Digression

37

Wavelength

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

Pink Floyd, *The Dark Side of the Moon*

Pittoni, 1725, Allegory to Newton

Rainbow

- From "Color and Light in Nature" by Lynch and Livingstone

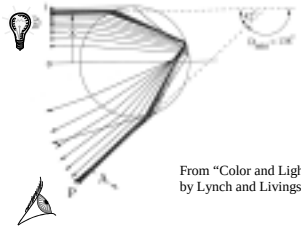
Digression

39

Rainbow

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

Digression



From "Color and Light in Nature" by Lynch and Livingstone

40

Questions?

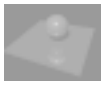
41

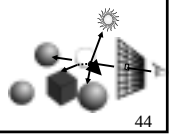
42

43

Overview of today

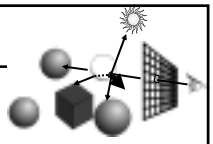
- Shadows
- Reflection
- Refraction
- Recursive Ray Tracing



44

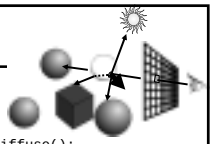
Recap: Ray Tracing



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

45

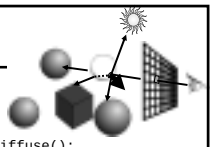
Recap: Ray Tracing



```
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;
```

46

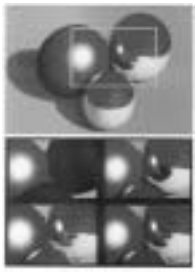
Does it end?



```
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;
```

47

The depth of reflection



48

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

0 recursion 1 recursion 2 recursions

Ray-Surface Intersection

- Implicit surfaces: $f(x, y, z) = 0$
  - Use a parametric representation for the ray:

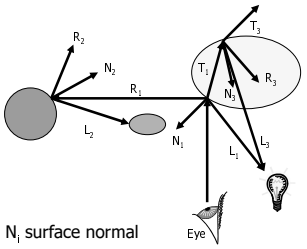
$$R(t) = O + tD$$
$$R_x(t) = O_x + tD_x$$
$$R_y(t) = O_y + tD_y$$
$$R_z(t) = O_z + tD_z$$

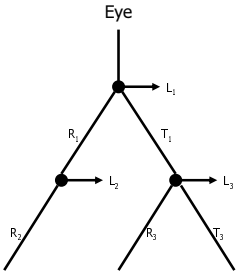
  - Substitute into the implicit equation:

$$f(O_x + tD_x, O_y + tD_y, O_z + tD_z) = 0$$

  - Solve the resulting equation
  - Examples: plane, sphere

The Ray Tree





N_i surface normal

R_i reflected ray

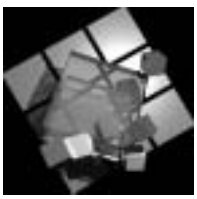
L_i shadow ray

T_i transmitted (refracted) ray

Kewl visualization

- Ben Garlick's SGI demo flyray
- On an Athena SGI O2:

```
add 6.837
cd /mit/6.837/demos/flyray/data
../flyray
```

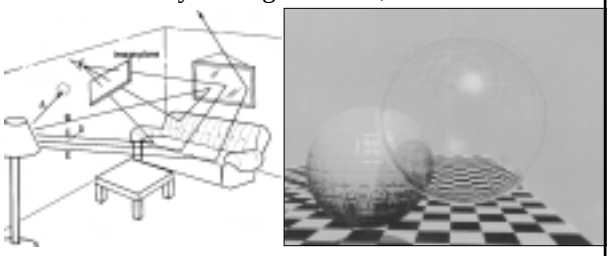


Real-time ray tracing

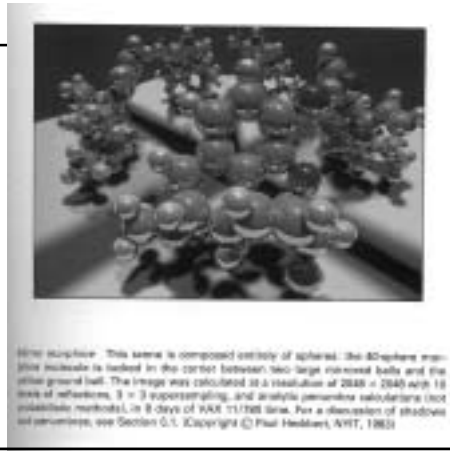
- Steve Parker et al. (U. of Utah)

Ray Tracing History

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



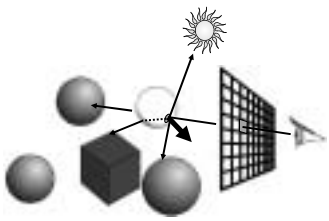
55



56

Does Ray Tracing simulate physics?

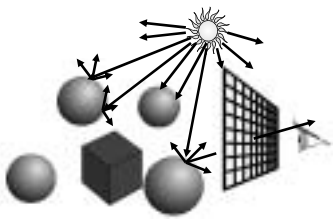
- Photons go from the light to the eye, not the other way
- What we do is backward ray tracing



57

Forward ray tracing

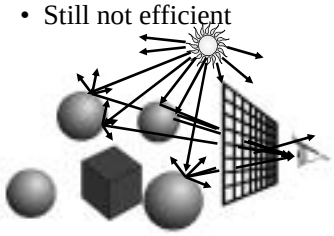
- Start from the light source
- But low probability to reach the eye
  - What can we do about it?

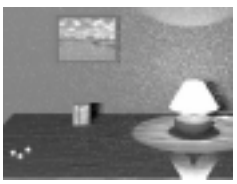


58

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

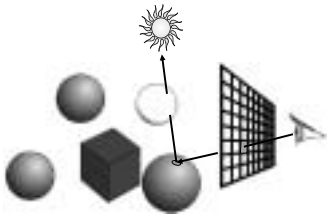




59

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



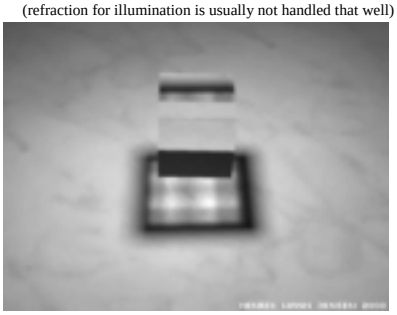
60

Correct transparent shadow

Animation by Henrik Wann Jensen

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

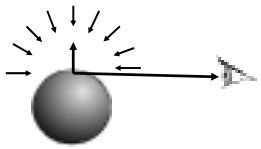
Digression



61

The Rendering equation

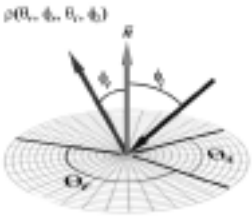
- Clean mathematical framework for light-transport simulation
- We'll see that in November
- At each point, outgoing light in one direction is the integral of incoming light in all directions multiplied by reflectance property



62

BRDF

- Reflectance properties, shading and BRDF
- Guest lecture by Wojciech Matusik



63