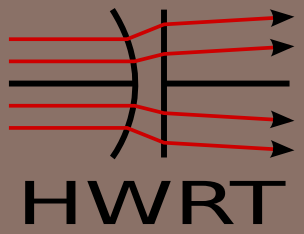


CS 6958
LECTURE 5
TM ANALYSIS
SHADING

January 20, 2014

Clarification

- Avoid global variables

- class/struct types will cause compiler to fail
  - What I meant was global *instances* of objects
  - You can definitely define and use classes/structs

Pass By Reference

- Do this whenever possible
- Copying arguments is slow
  - Especially for large data types
- Passing by const reference keeps the original data safe

Issue Statistics

- Issue Rate:
  - Want this as high as possible
- iCache conflicts
  - Tracked separately from “resource” conflicts
- thread*cycles of resource conflicts
  - FU conflicts, L1, L2, DRAM
- thread*cycles of data dependence
  - Nothing you can do about this (for now)
  - Includes RF conflicts

Default Areas (square mm)

- FPADD .003
- FPMIN .00072
- FPCMP .00072
- INTADD .00066
- FPMUL .0165
- INTMUL .0117
- FPINV .112
- CONV .001814
- BLT .00066
- BITWISE .00066

Instruction Caches

- Instruction caches are actually "double pumped"
  - Each bank can service 2 requests every cycle
- Is N banks as good as N caches?
  - Is N caches reasonable?
- Is N banks > T threads useful?

trax.hpp

- Some of the useful functions (more as we need them)
  - invsqrt(float f)
  - sqrt(float f)
  - min(float a, float b)
  - max(float a, float b)
  - GetXRes()
  - GetYRes()
  - GetFrameBuffer()

Ray tracer design

The major components in a ray tracer are:

- Camera (Pixels to Rays)
- Objects (Rays to intersection info)
- Materials (Intersection info and light to color)
- Lights
- Background (Rays to Color)

- All together: a Scene

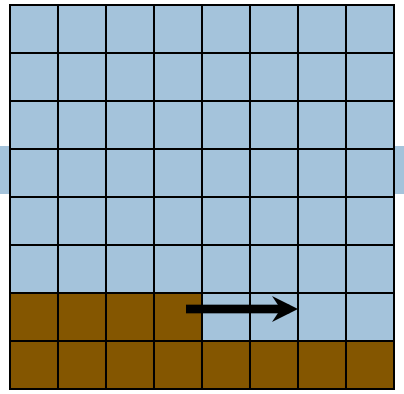
Ray tracing algorithm

```
foreach frame
  foreach pixel
    foreach sample
      generate ray
      intersect ray with objects
      shade intersection point
```

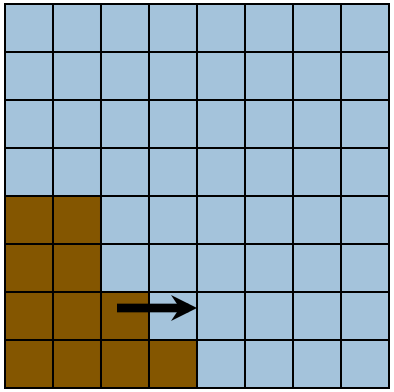
Starting simple

foreach frame	Ignore for now
foreach pixel	Atomic Increment
foreach sample	Ignore for now
generate ray	Camera
intersect ray with objects	Spheres, more soon
shade intersection point	Material

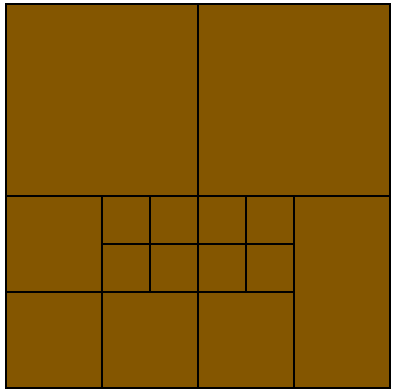
Foreach pixel



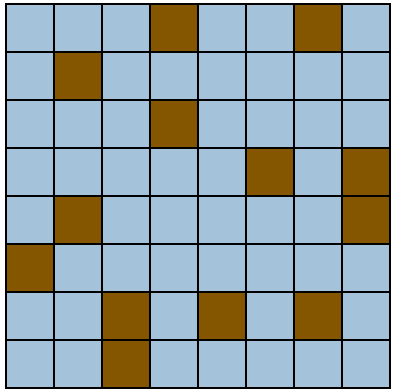
Row-major order



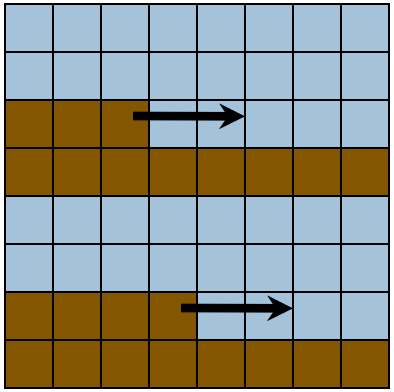
Tiled



Progressive



Frameless rendering



Parallel

Atomic Increment

- atomicinc(0)
  - Atomically increments global register 0
  - All threads have access to this register

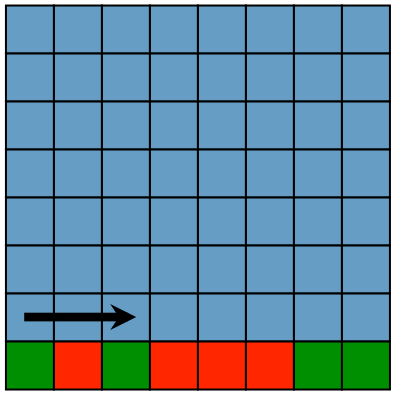
```
for(int pix = atomicinc(0); pix < xres*yres; pix = atomicinc(0))
  int i = pix / xres;
  int j = pix % xres;
```

- Compare to

```
for(int i=0; i < xres; i++)
   for(int j=0; j < yres; j++)
```

Atomic Increment

```
for(int pix = atomicinc(0); pix < xres*yres; pix = atomicinc(0))
  int i = pix / xres;
  int j = pix % xres;
```



Potential pixel assignments

Thread 1	Thread 2
0	1
2	3
6	4
7	5

Cameras – Coming Soon

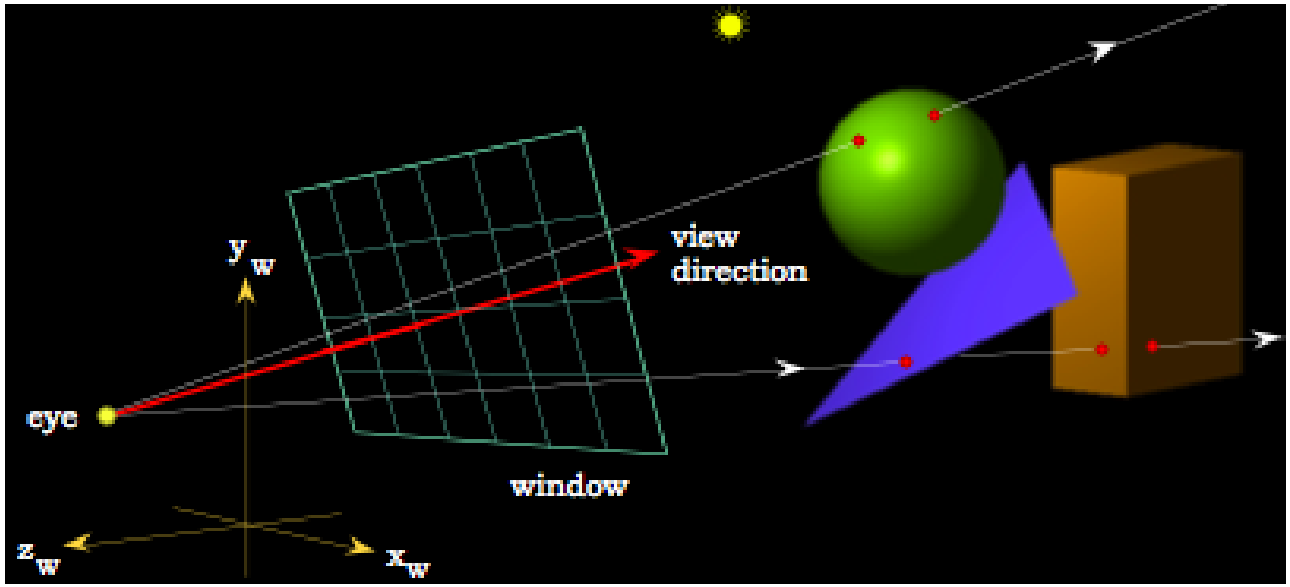
foreach frame

  foreach pixel

    foreach sample

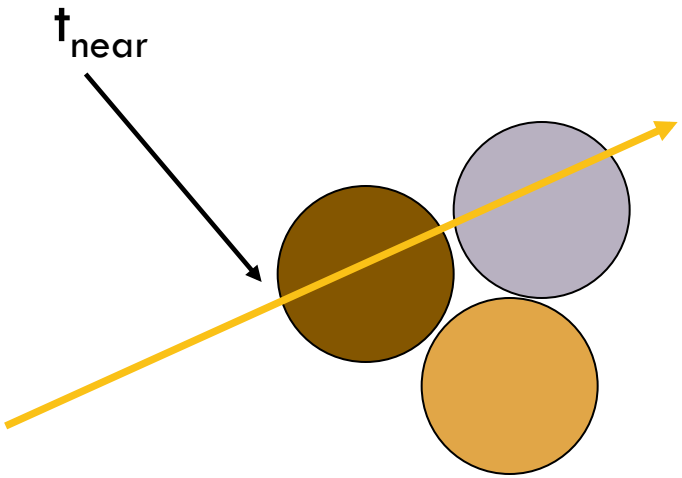
      generate ray

Camera



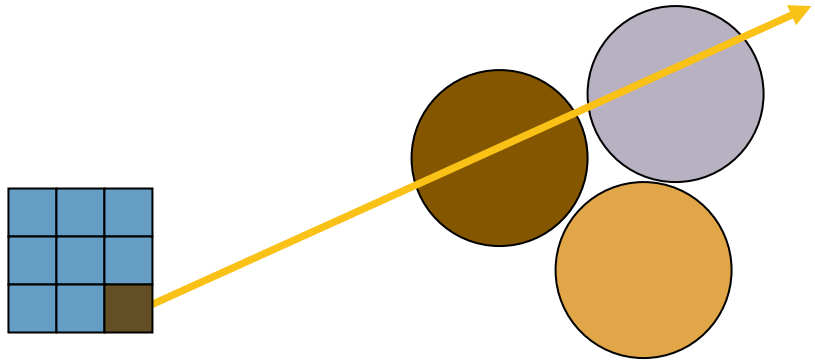
Find Closest Object

foreach frame
  foreach pixel
    foreach sample
      generate ray
      intersect ray with objects Spheres, more soon



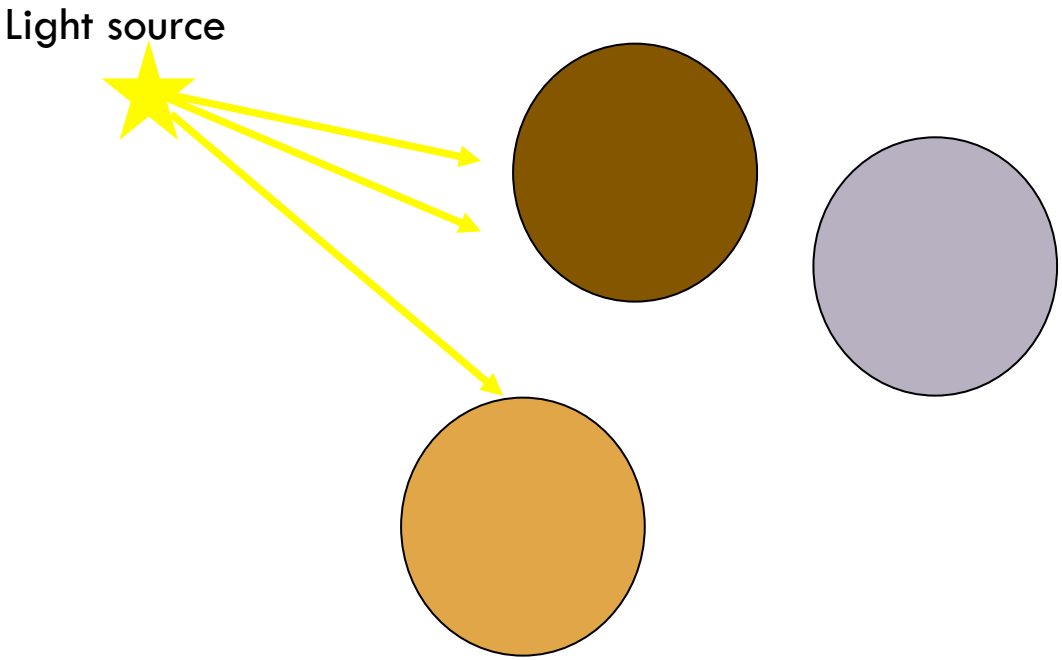
Shading (can get very complex)

foreach frame
  foreach pixel
    foreach sample
      generate ray
      intersect ray with objects
      shade intersection point Pixel gets a color



Shading

- Path tracing (and other global techniques)
  - Consider light from all sources
- Starting simple
  - Consider light from direct source(s)



Color Multiplication

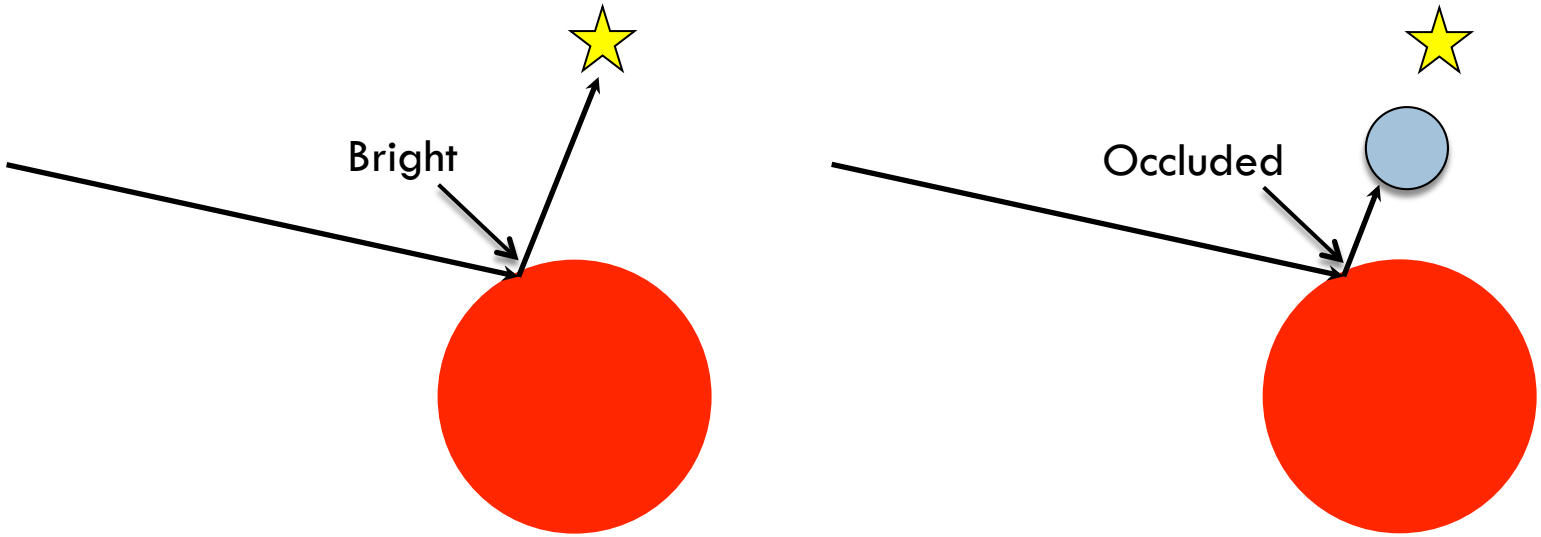
- C1 * C2 =

  <C1.R * C2.R, C1.G * C2.G, C1.B * C2.B>
- If white=<1,1,1>, red=<1,0,0>, green=<0,1,0>
- white * c = c
- red * green = black

- Remember, colors in range [0 .. 1]
  - Can only get darker by reflecting off surfaces

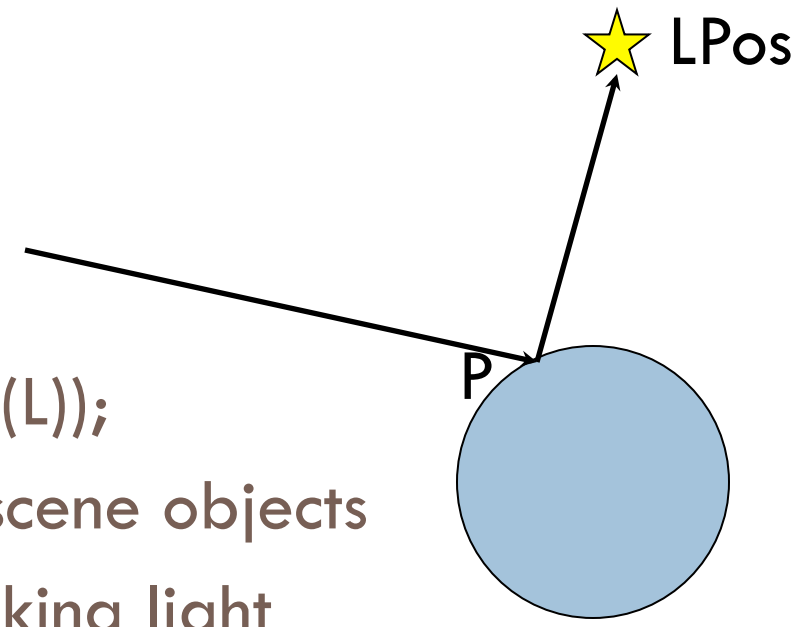
Direct Light (and thus shadows)

- If there is line of sight from hit point to light source, add light's contribution
  - Pixel color += object color * light color
- Else it is in shadow
  - Do nothing



Computing Direct Light

- First we need a vector from hit point to light
  - P = hit point
  - LPos = location of light
  - L = LPos – P



Ray shadowRay(P, normalize(L));

intersect shadowRay with scene objects

determine if anything blocking light

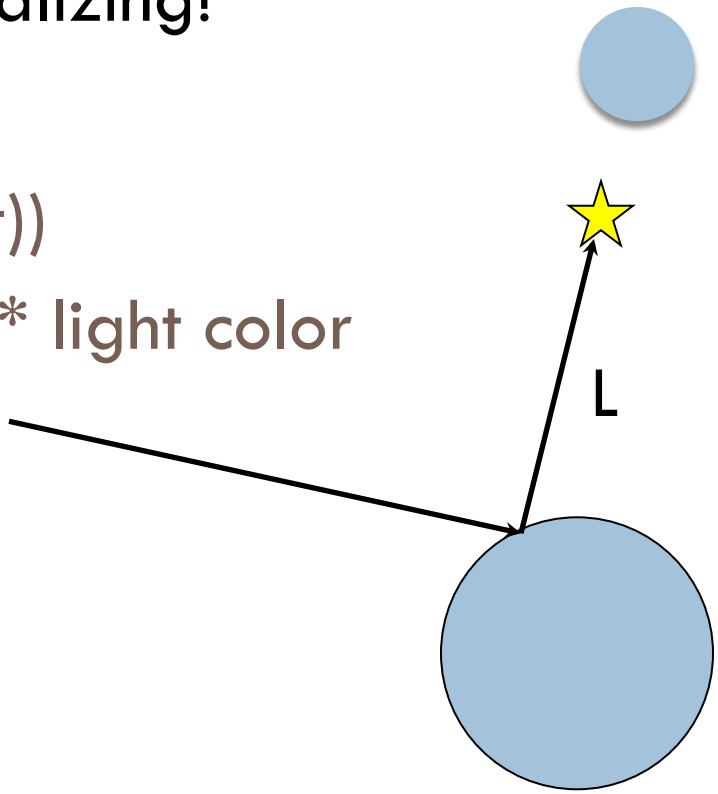
Computing Direct Light

- We don't care about intersections behind the light!

max_t = |L| ← before normalizing!

If (! (hit && 0 < hit_t < max_t))

Pixel color += object color * light color

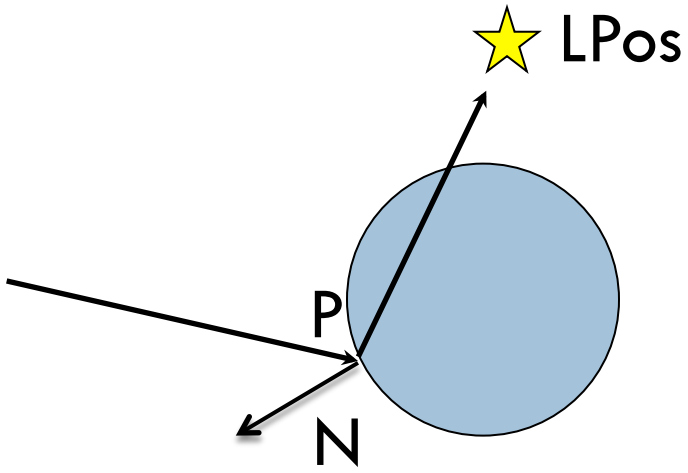


Computing Direct Light

- Sometimes we don't need to cast a ray
  - Hit surface is on opposite side of light
  - Angle between normal and L > 90
    - $\cos\theta = N \cdot \text{normalize}(L)$
    - N = surface normal direction

if($\cos\theta < 0$)

skip shadow ray



Flipped Normals

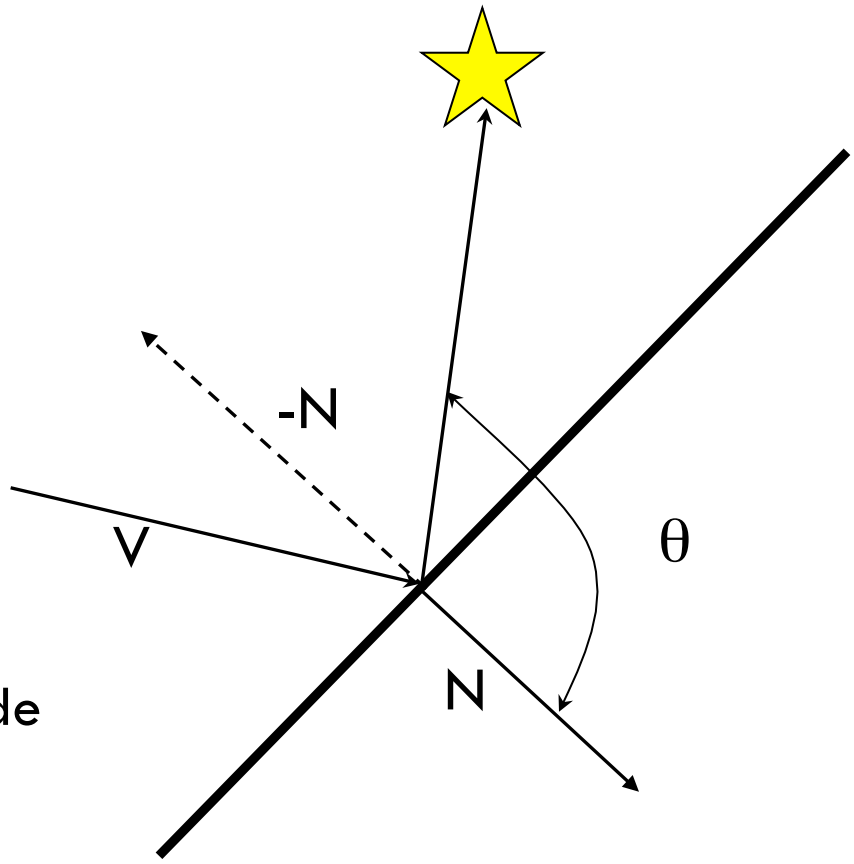
- Normals don't always point the right way
  - Depends on how N calculated
  - θ = wrong angle!

- V = camera ray direction

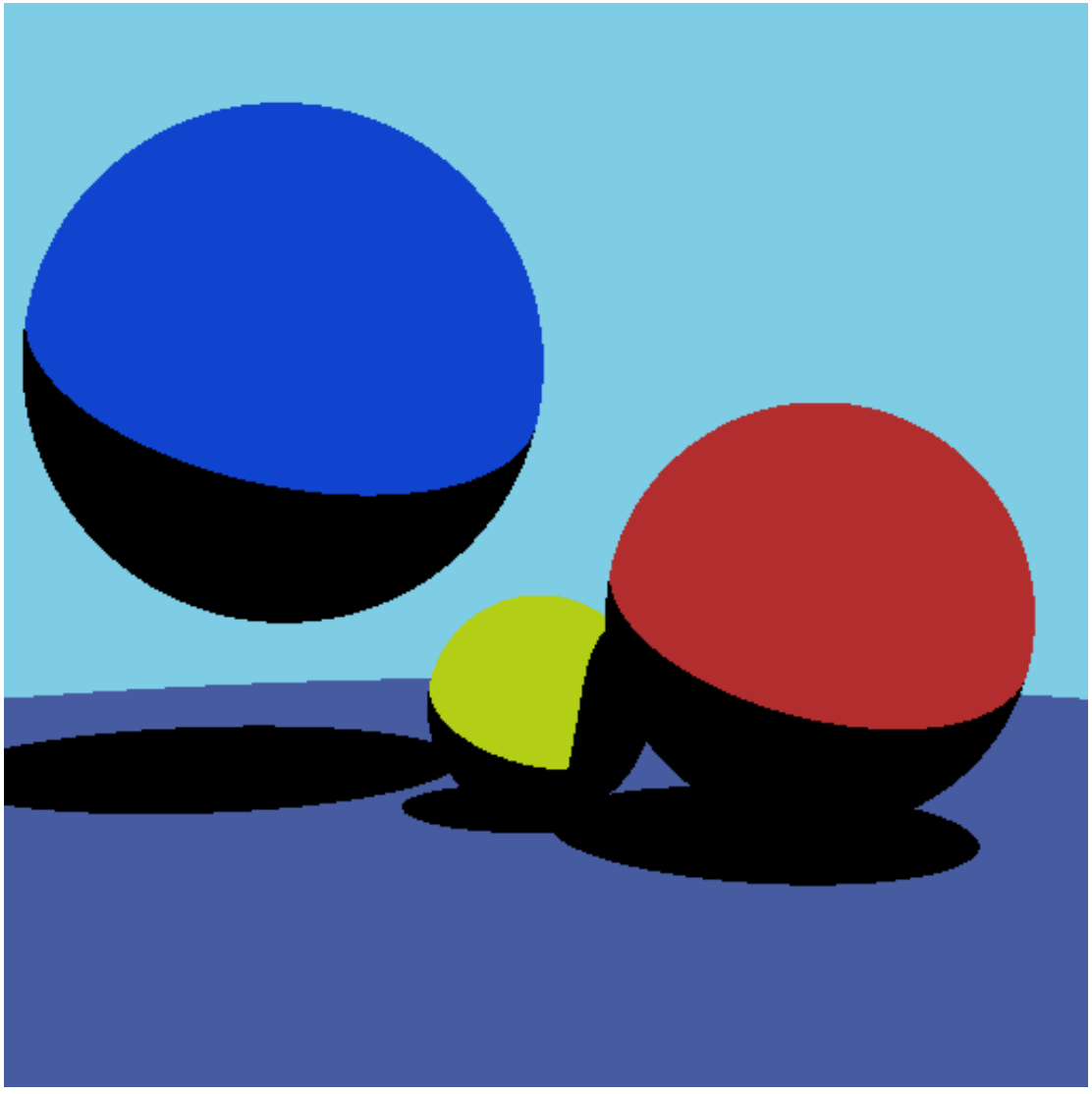
if(V · N > 0)

N = -N

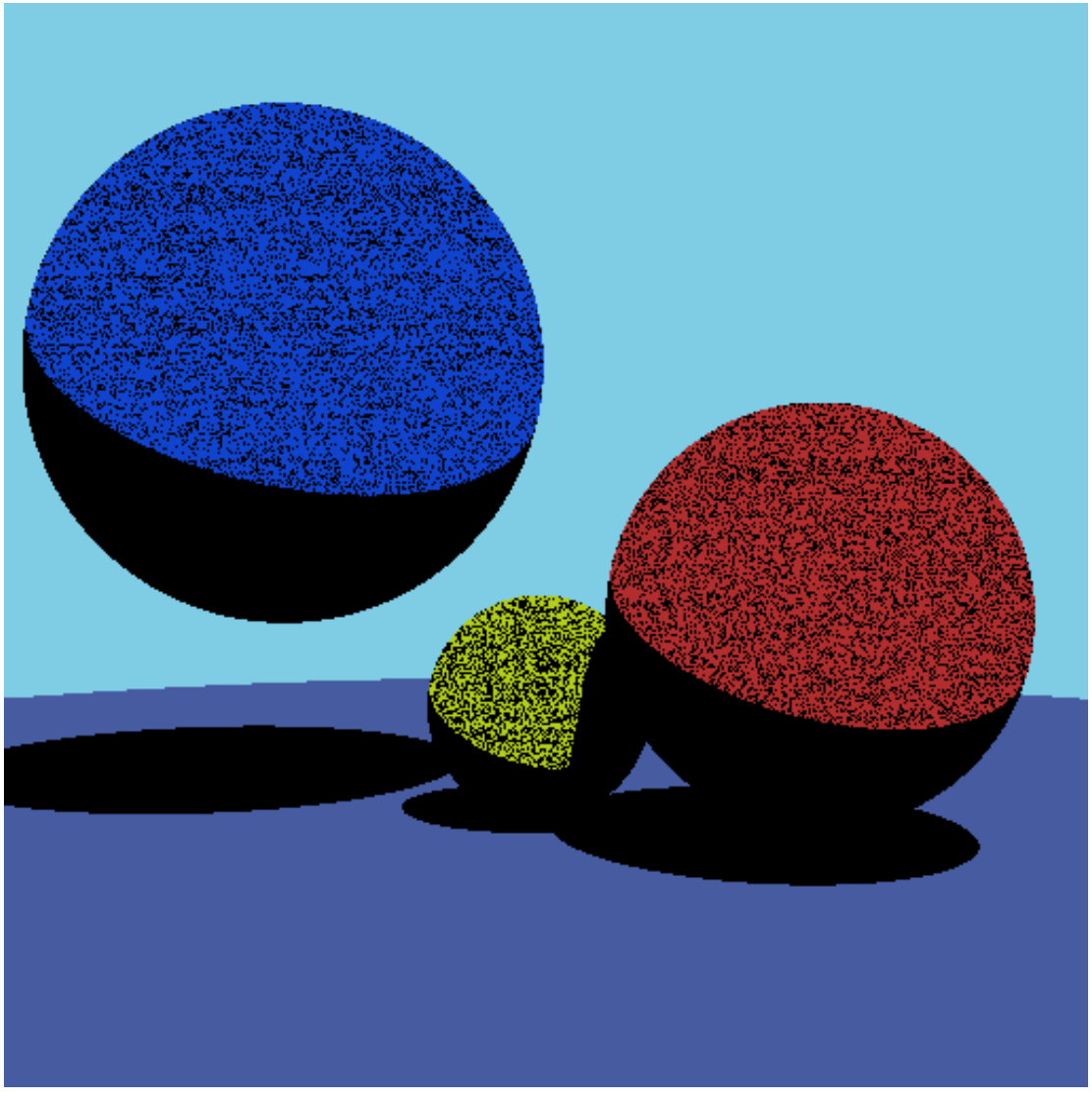
Always flip normals to be on same side as incoming ray



Shadows

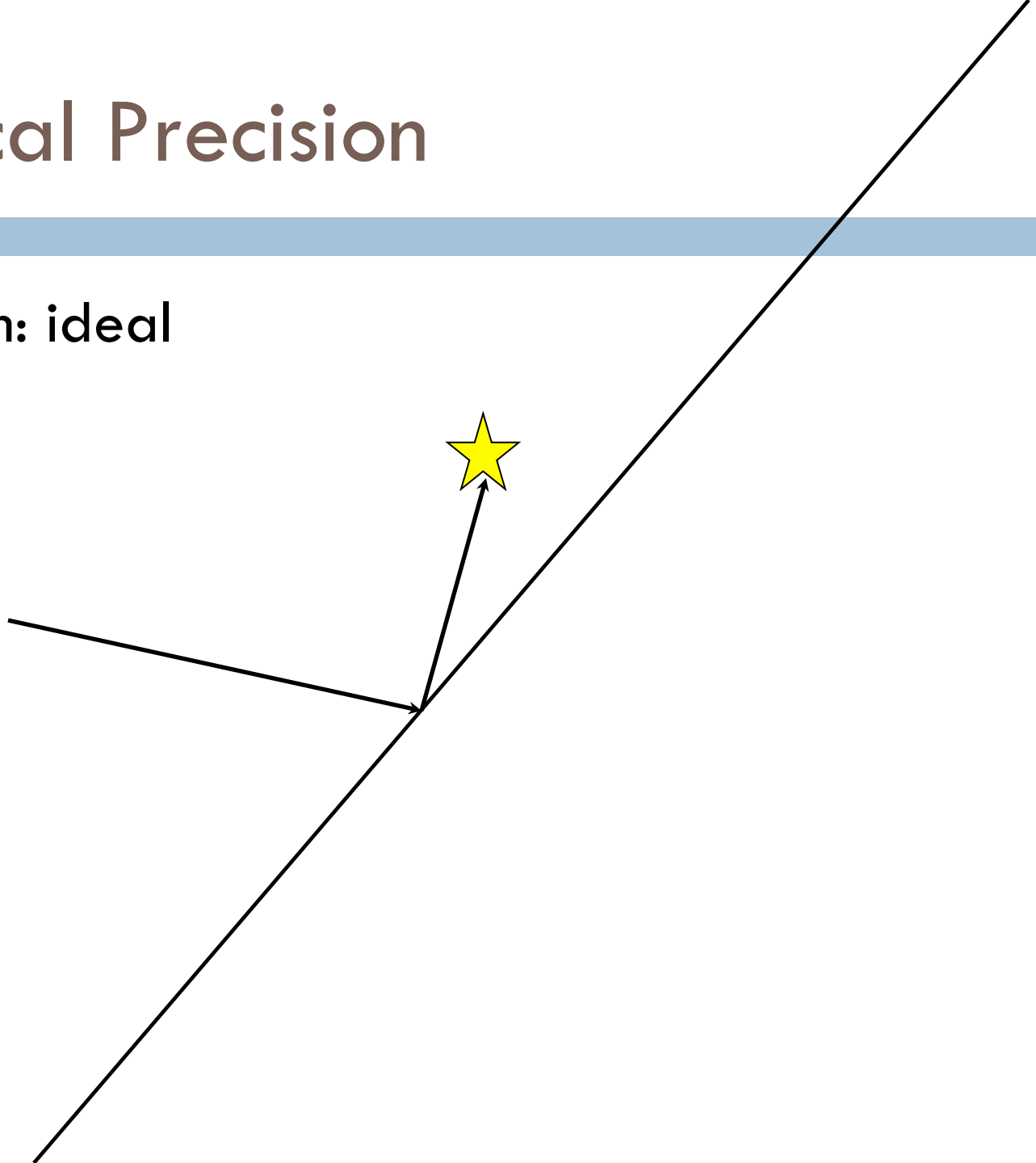


What's Wrong Here?



Numerical Precision

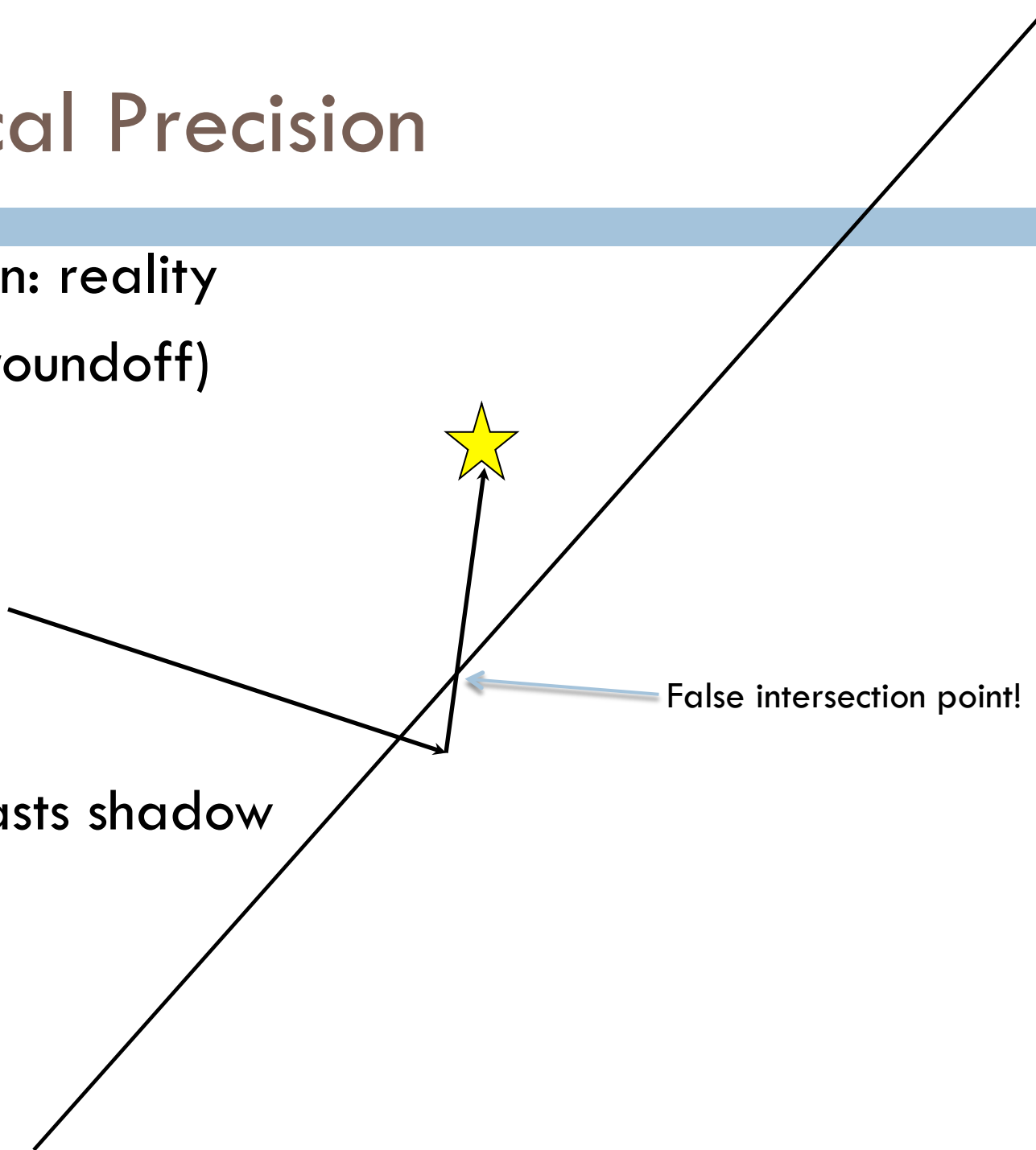
- Zoomed in: ideal



Numerical Precision

- Zoomed in: reality

(numerical roundoff)

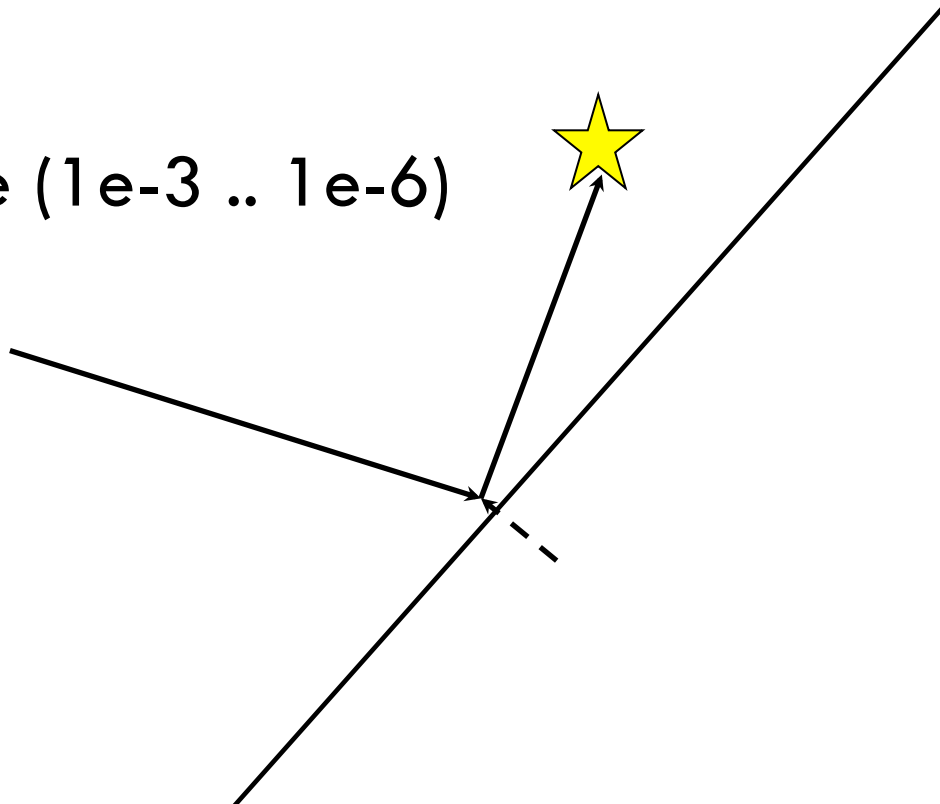


- Object casts shadow

  on itself

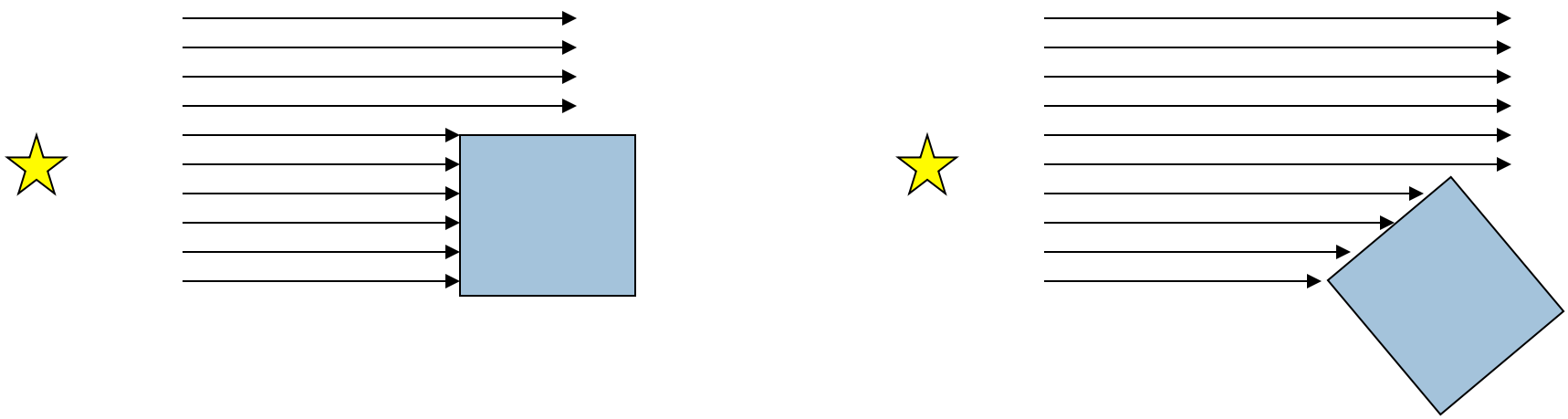
Solution

- Offset shadow ray origin in normal direction
  - P += N*epsilon
  - epsilon = some small number
- Epsilon depends on scene (1e-3 .. 1e-6)



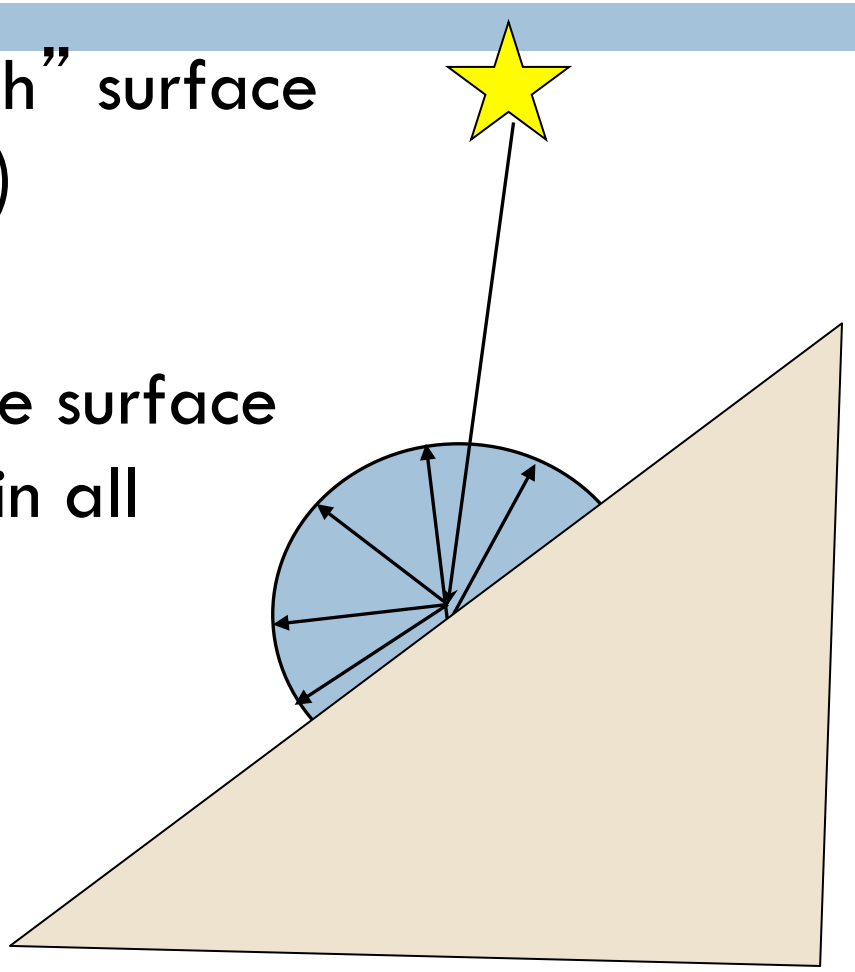
Lambert's Cosine Law

- Light reaching surface is proportional to projected visible area: cos θ
- θ = angle between light and normal



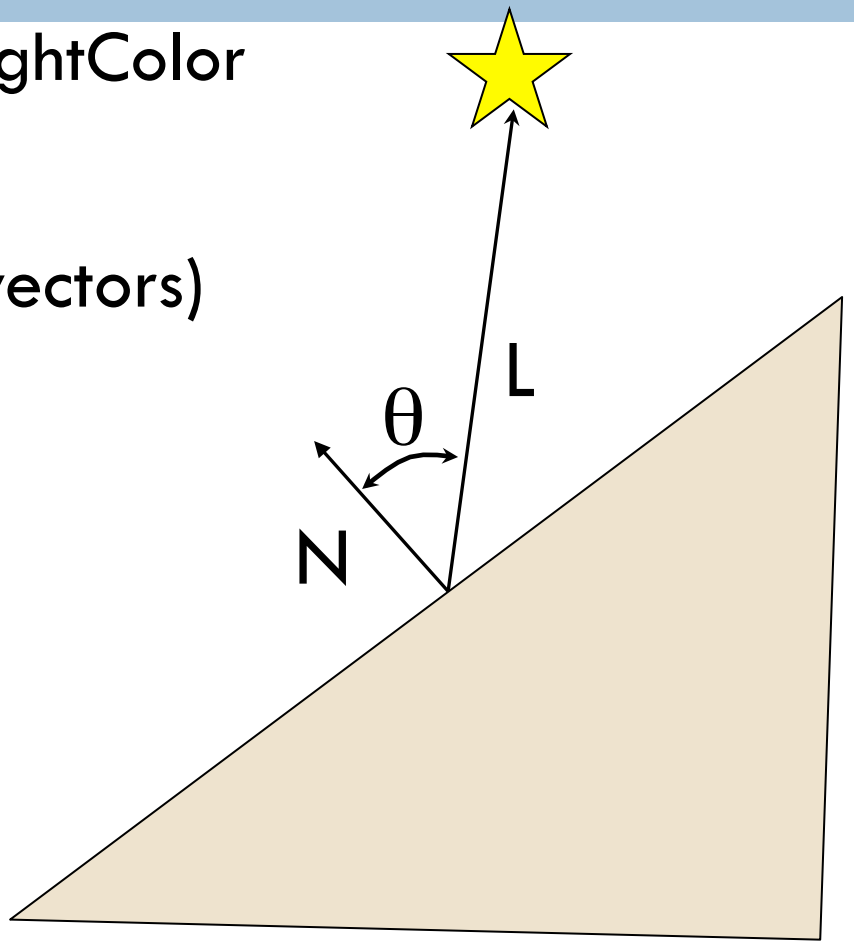
Lambertian shading

- Comes from a "rough" surface (at microscopic level)
- Light that reaches the surface is reflected equally in all directions

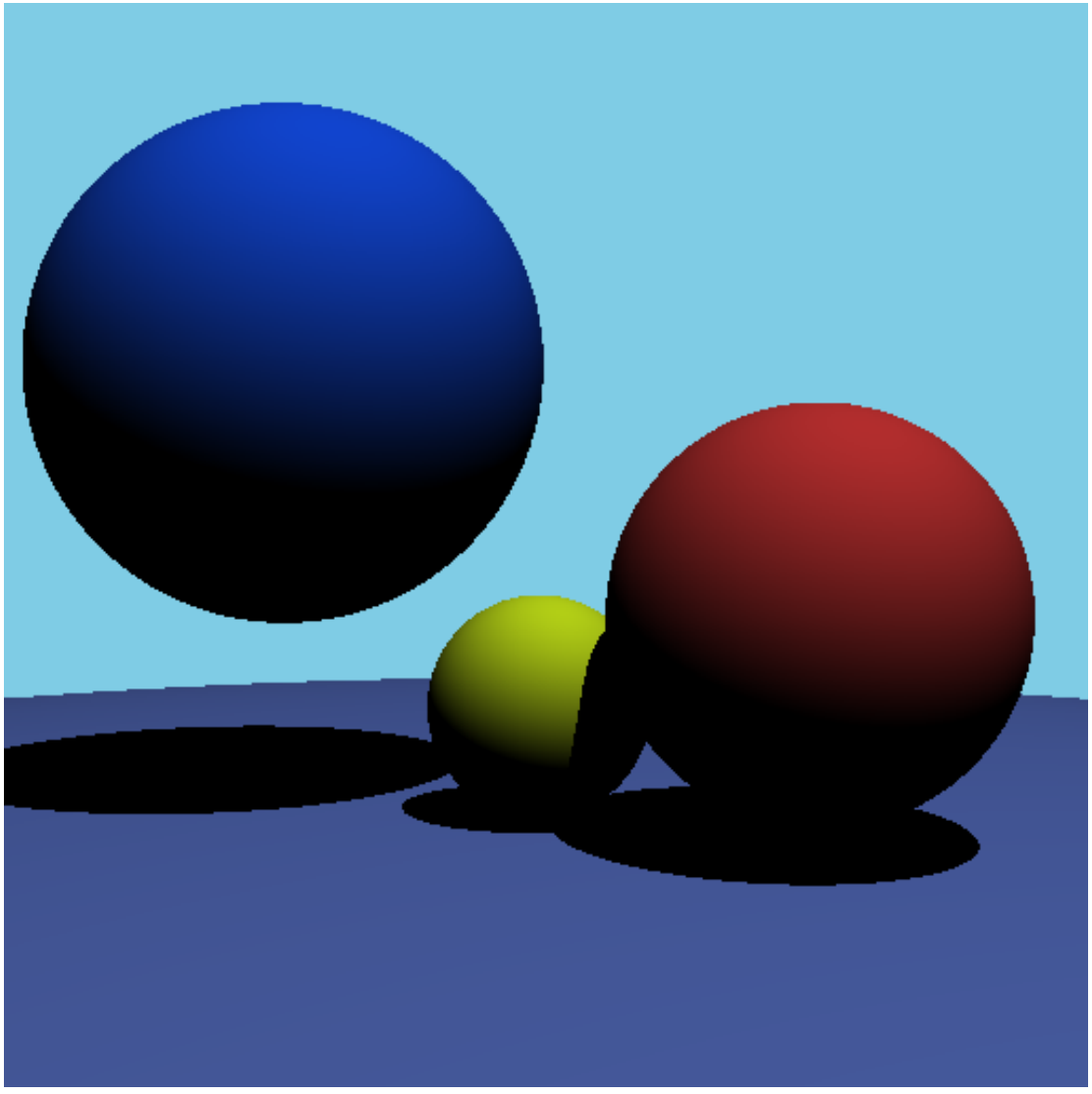


Lambertian shading

- Color at surface: cosθ * lightColor
  - cosθ = (N · L)
- (where N and L are unit vectors)



Direct Light



Ambient light

- With this mechanism, the light in a shadowed region is 0 (black)
- To avoid this, use "ambient" lighting

- For each scene, define Kd, Ka
  - Kd + Ka = 1
- Kd light comes from direct sources
- Ka light comes from "ambient" sources

Ambient light

- Pixel color =
  - objectColor * [(N · L)*lightColor*Kd + ambientColor*Ka]

- Define some ambientColor for the scene
  - Based on how bright scene is, background, etc…
  - Artistic choice (since it is a hack)

Direct + Ambient Light

- Ambient = <.6, .6, .6>
- Kd = .7
- Ka = .3