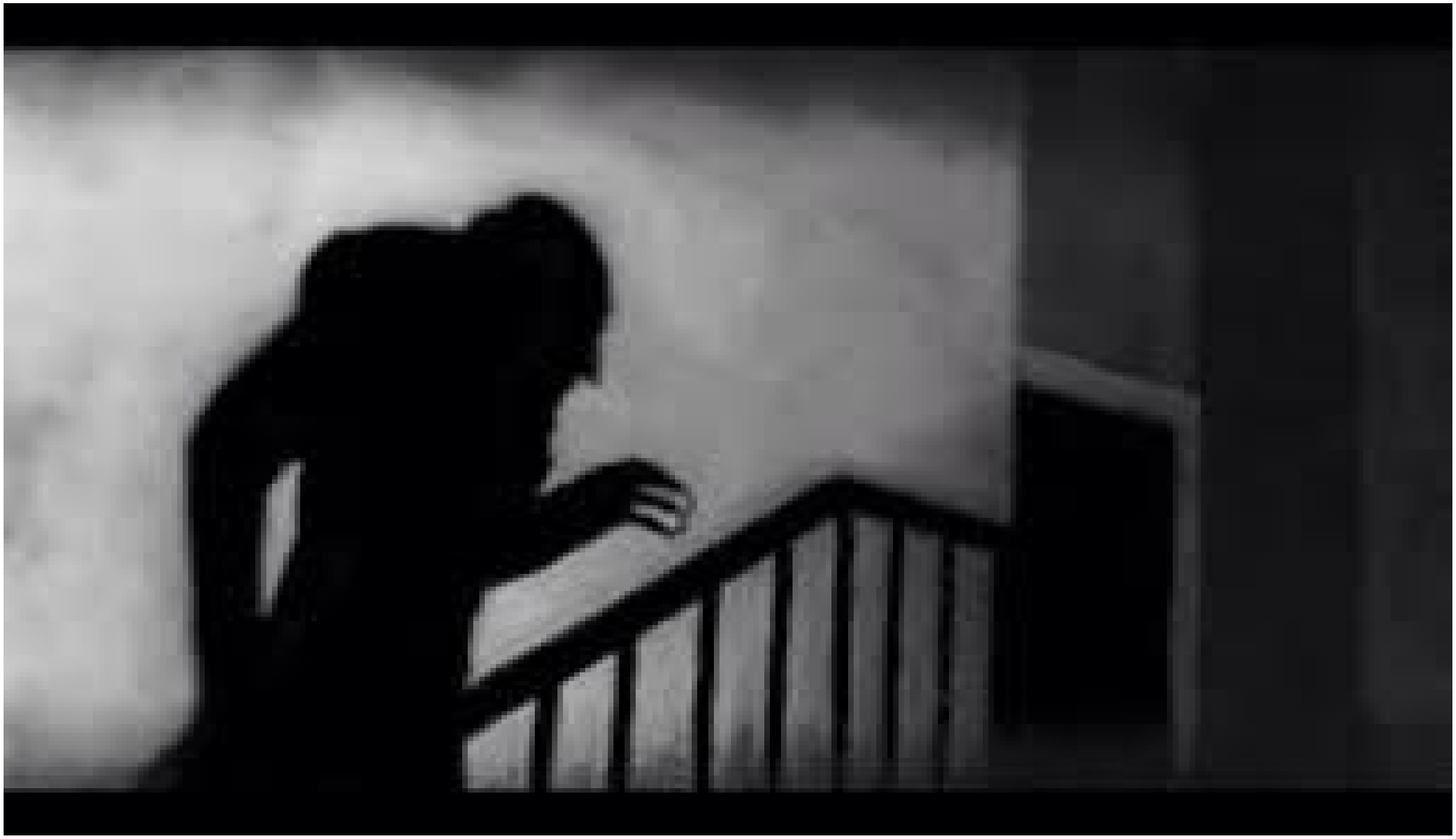
lecture 19

Shadows

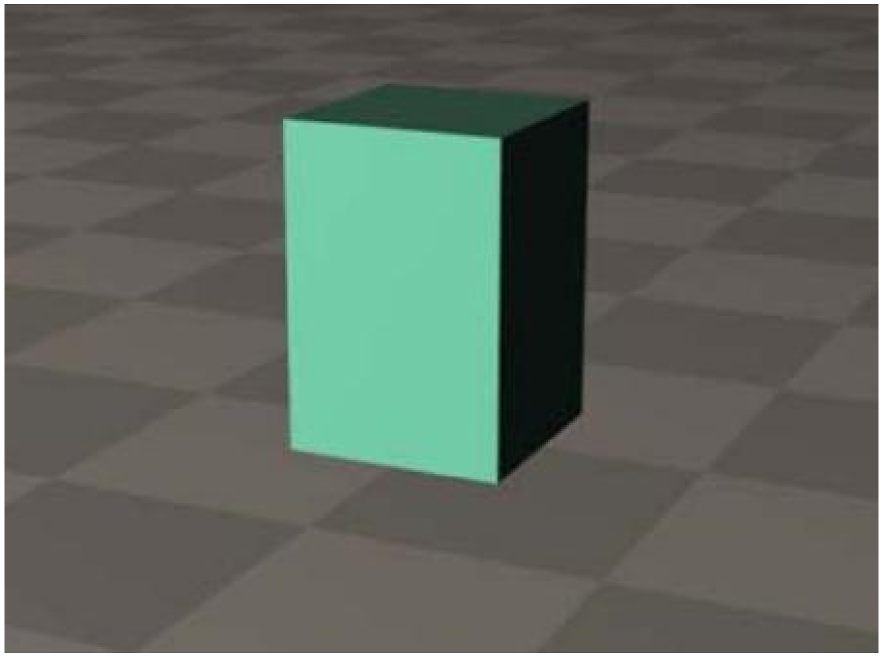
- ray tracing
- shadow mapping
- ambient occlusion

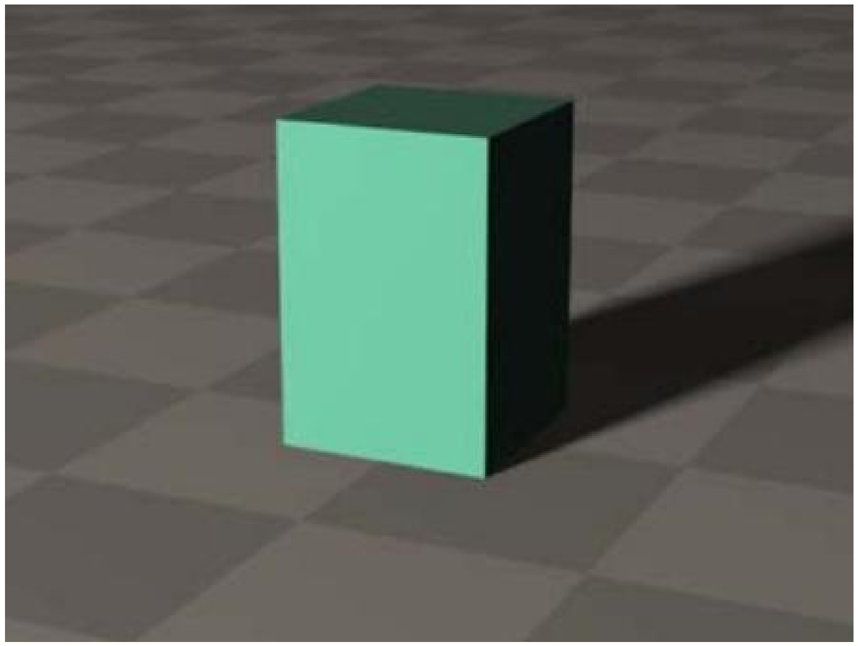
Interreflections

In cinema and photography, shadows are important for setting mood and directing attention.



Shadows indicate spatial relationships between objects e.g. contact with floor.

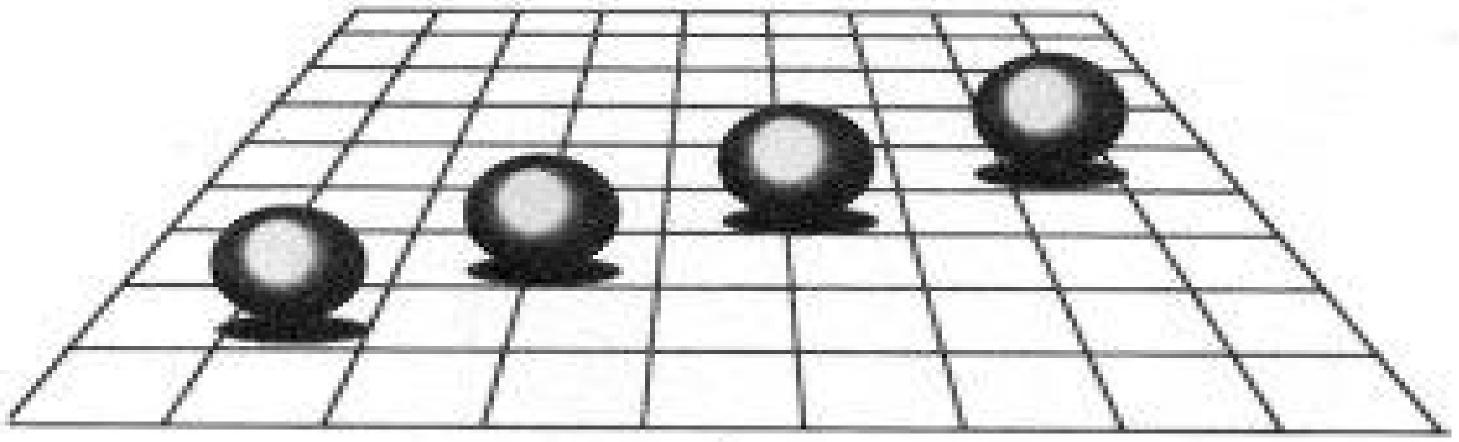


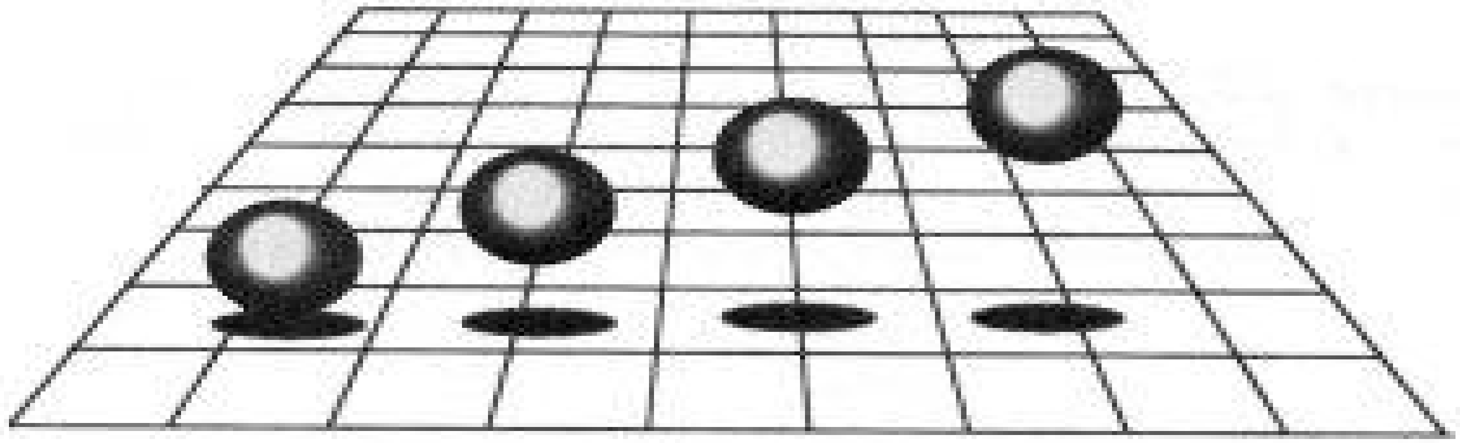


without shadow

with shadow

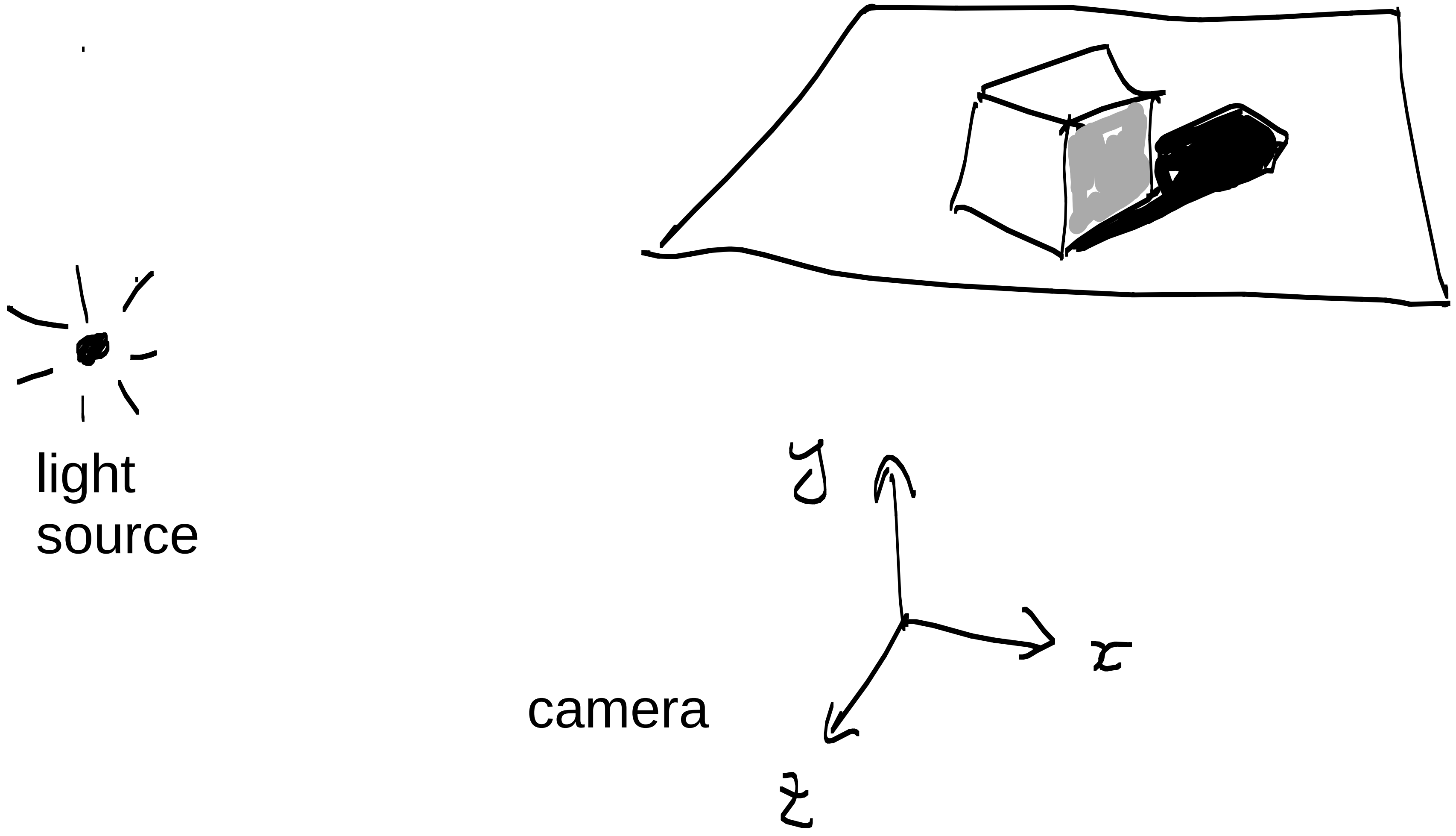
http://www.cs.utah.edu/percept/papers/Madison:2001:UIS.pdf





Two types of shadow :

- "attached" (n . l < 0)
- "cast"



Blinn-Phong Model **with Shadows**

.

Let the function **S(x)** = 1 when the light source is visible from x, and let **S(x)** = 0 when it not (i.e. shadow).

$$I(\mathbf{x}) \; = \; S(\mathbf{x}) \, \{ \; I_l(\mathbf{x}) \; k_d(\mathbf{x}) \; \mathbf{n}(\mathbf{x}) \cdot \mathbf{l}(\mathbf{x}) + I_{specular}(\mathbf{x}) \; \}$$

$$+ \; I_{amb} \; k_a(\mathbf{x})$$

Ray Tracing with Shadows (Assignment 3)

```
for each pixel   {

    cast a ray through that pixel into the scene to find an
         intersection point (x,y,z)

    compute RGB ambient light component at (x,y,z)

    for each point light source{

         cast a ray from (x,y,z) to light source
              // check if light is visible,   called "shadow testing"

         if light source is visible
                  add RGB contribution from that light source
    }
}
```

Shadow Mapping (basic idea)

Instead of asking:

"for each point in the scene, which lights are visible ?"

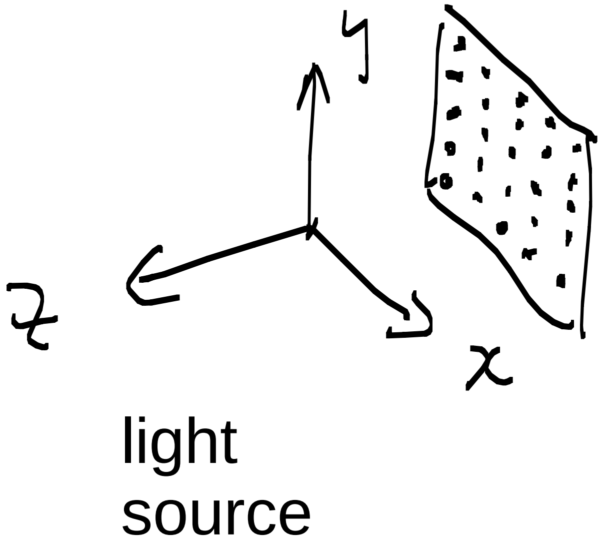
we ask

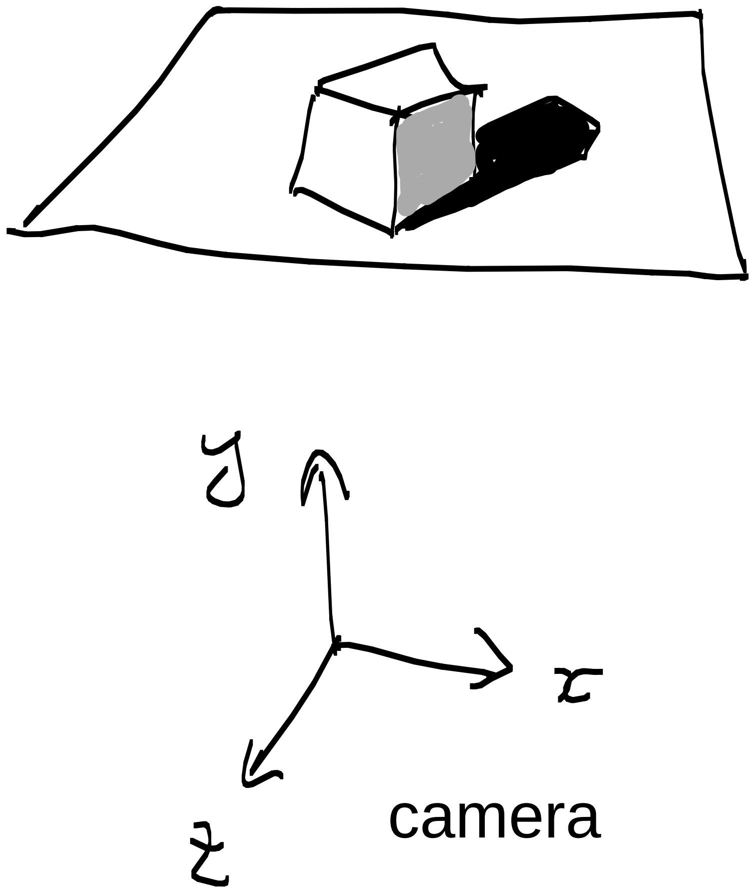
"what is seen from the light source's viewpoint ?"

"Shadow map" [Haines and Greenberg 1986]

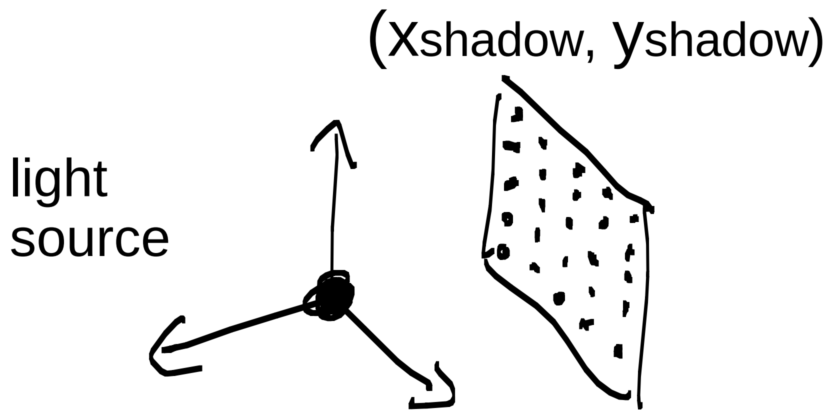
= a depth map as seen from the light source

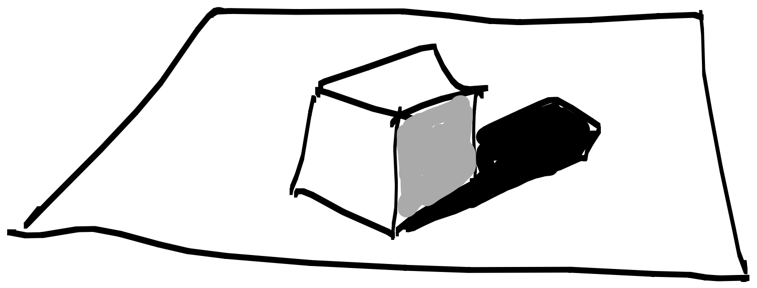
This term is potentially confusing. A surface is seen from the light source when it is NOT in shadow.





Coordinate Systems



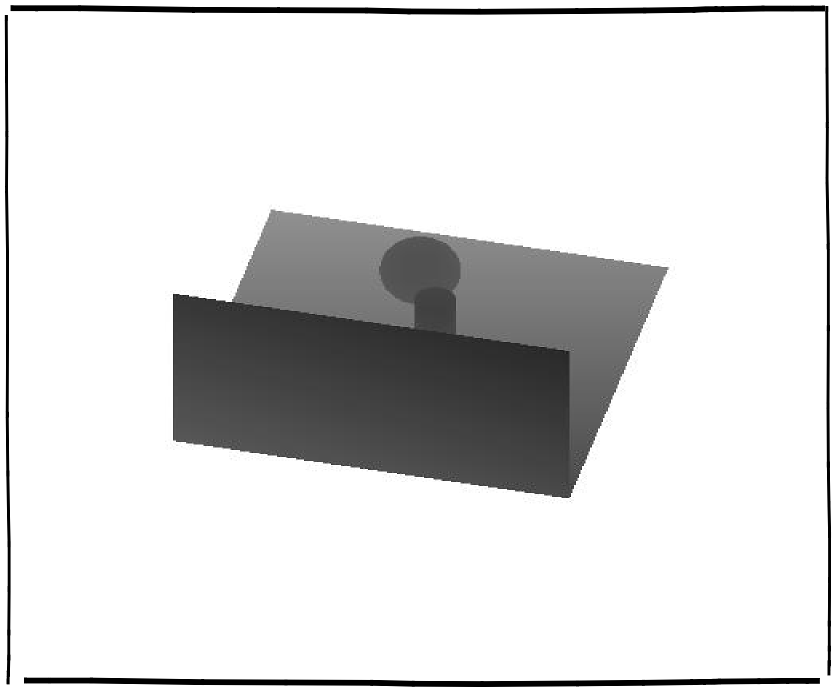


Notation:

Let $(x_{light}, y_{light}, z_{light})$ be ***continuous*** light source coordinates.

Let $(x_{shadow}, y_{shadow}, z_{shadow})$ be ***discrete*** shadow map coordinates with z_{shadow} having 16 bits per pixel.

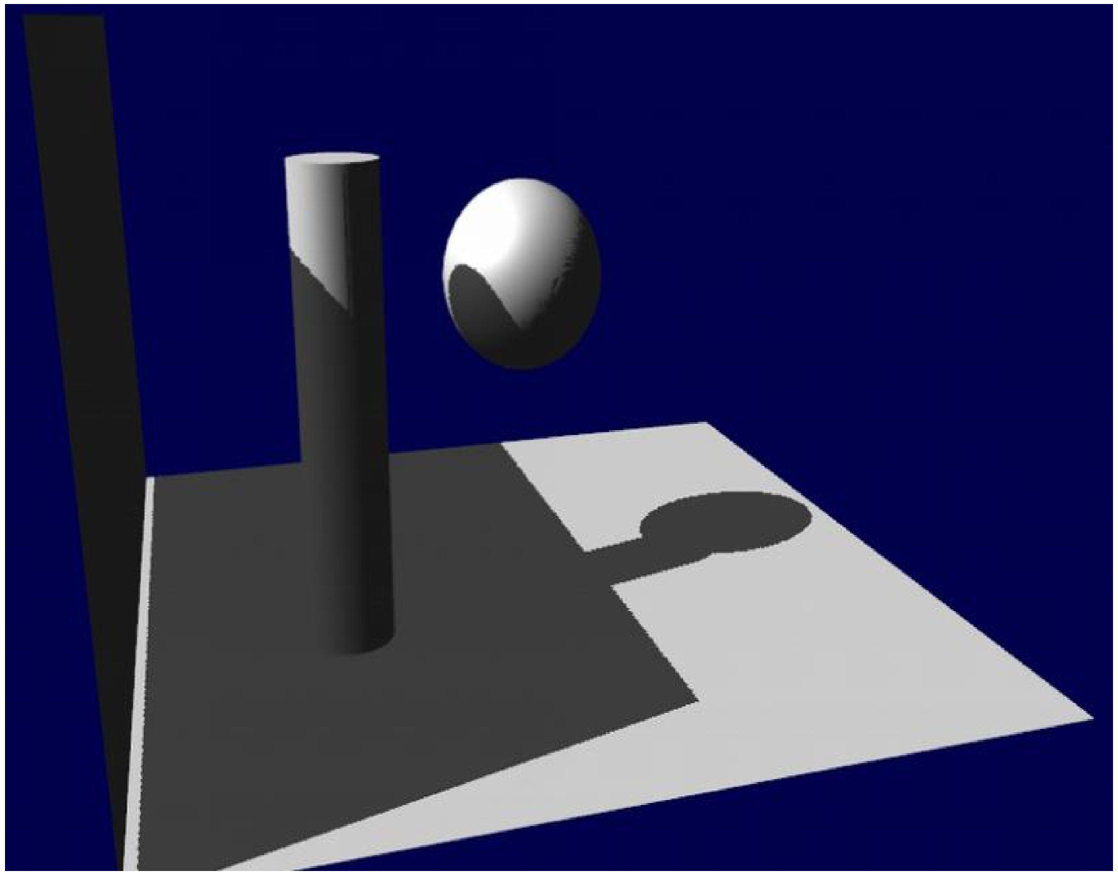
In both cases, assume the points have been projectively transformed, and coordinates are normalized to $[0, 1] \times [0, 1] \times [0, 1]$.



$z_{shadow}(x_{shadow}, y_{shadow})$

"shadow map"

light source viewpoint

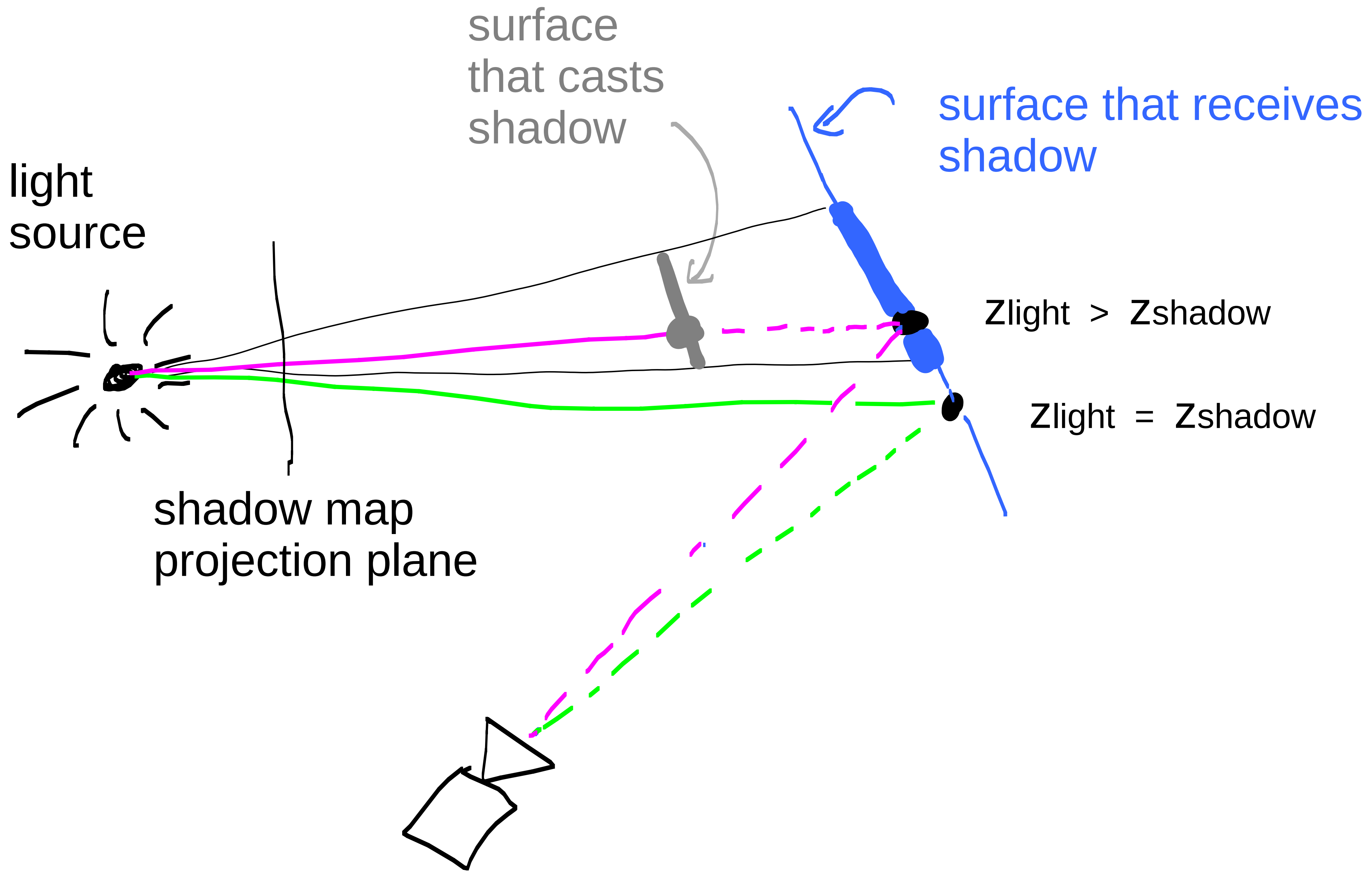


$I(x_p, y_p)$

RGB image (with shadows)

camera viewpoint

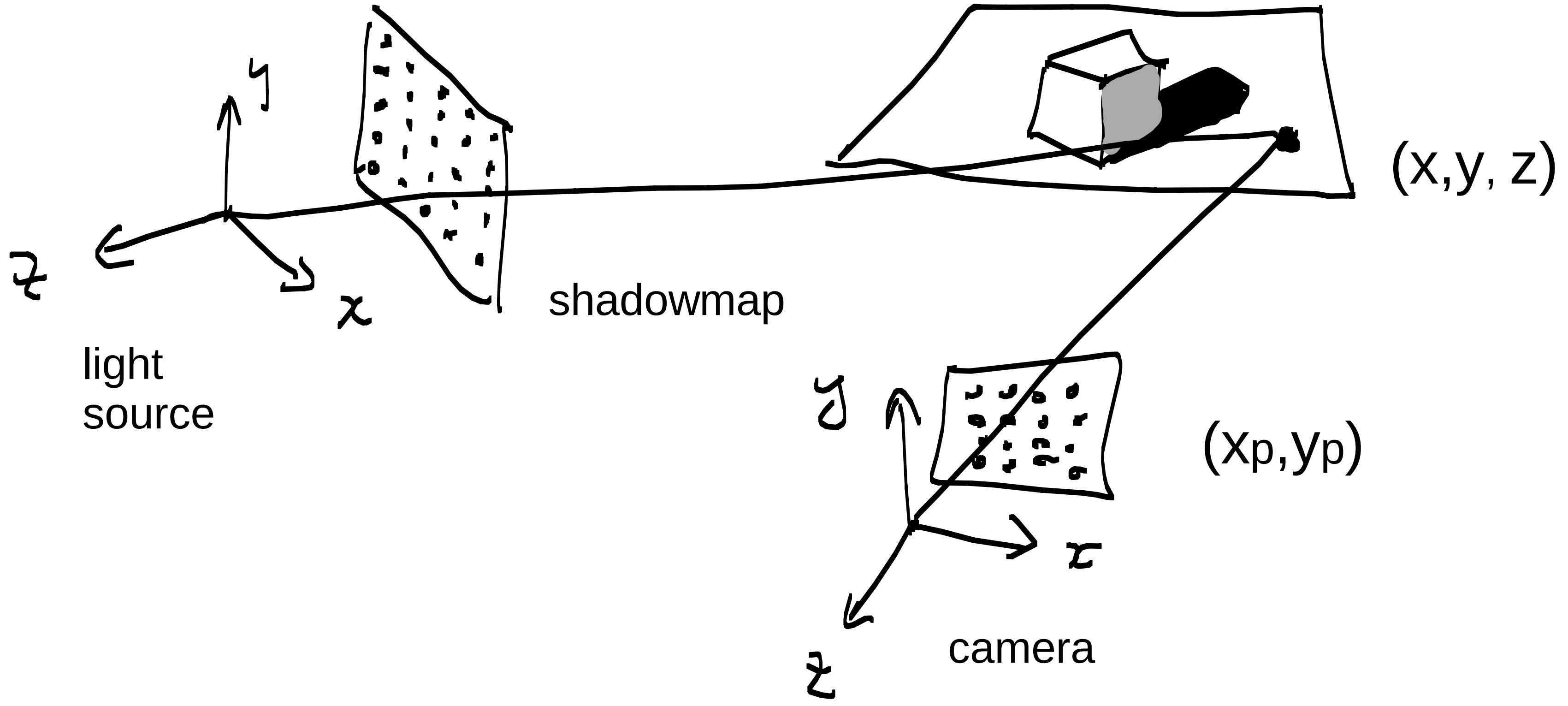
http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/


surface that casts shadow
surface that receives shadow
light source
Zlight > Zshadow
Zlight = Zshadow
shadow map projection plane

This illustration shows perspective view, but in fact the comparisons are done in normalized coordinates (i.e. after projective transform).

Shadow Mapping algorithm (sketch)

```
for each camera image pixel ($x_p$,$y_p$)  {
    find depth  $z_p$   of closest visible surface   //  using whatever method
    transform ($x_p$, $y_p$, $z_p$) to light coordinates ( $x_{light}$, $y_{light}$, $z_{light}$ )
    compare $z_{light}$  to  $z_{shadow}$( $x_{shadow}$, $y_{shadow}$ )  to decide if
          3D point is in shadow
    compute RGB
}
```



Shadow Mapping: a two-pass algorithm (and more details)

Pass 1:

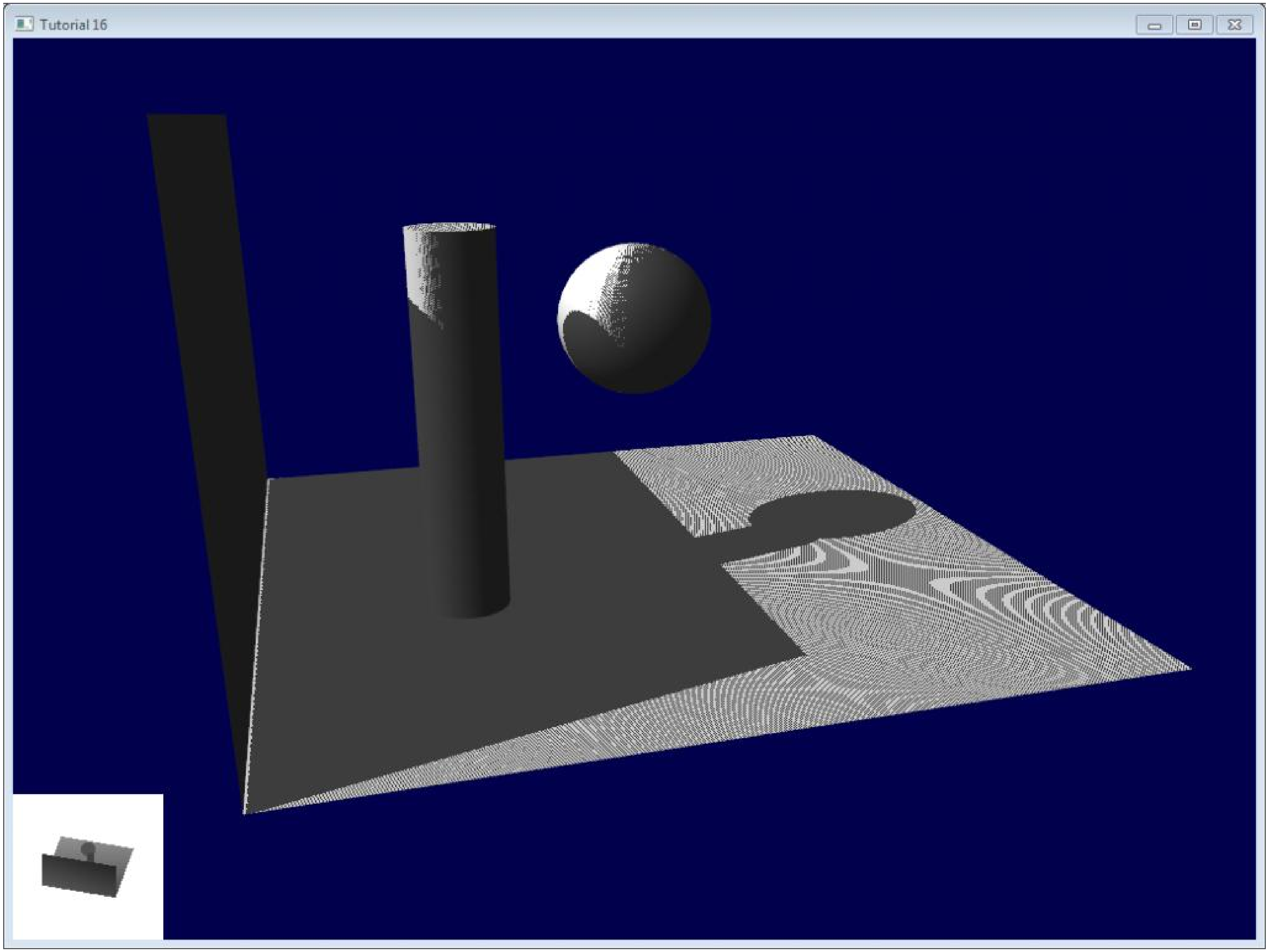
Compute only a shadow map z_{shadow}.

// Assume just one light source (can be generalized)

Pass 2:

```
for each camera image pixel $(x_p, y_p)$ {
    find depth $z_p$ of closest visible surface
    transform $(x_p, y_p, z_p)$ to light coordinates $(x_{light}, y_{light}, z_{light})$
    $(x_{shadow}, y_{shadow})$ = discretize$(x_{light}, y_{light})$    // to pixel positions
                                                                      // in shadow map
    if $z_{light} > z_{shadow}(x_{shadow}, y_{shadow})$
        $S(x_p, y_p) = 0$   // light source is not visible from point
    else                    //                i.e. point in shadow
        $S(x_p, y_p) = 1$   // light source is visible from point

    calculate RGB value e.g. using Blinn-Phong model with shadow
}
```

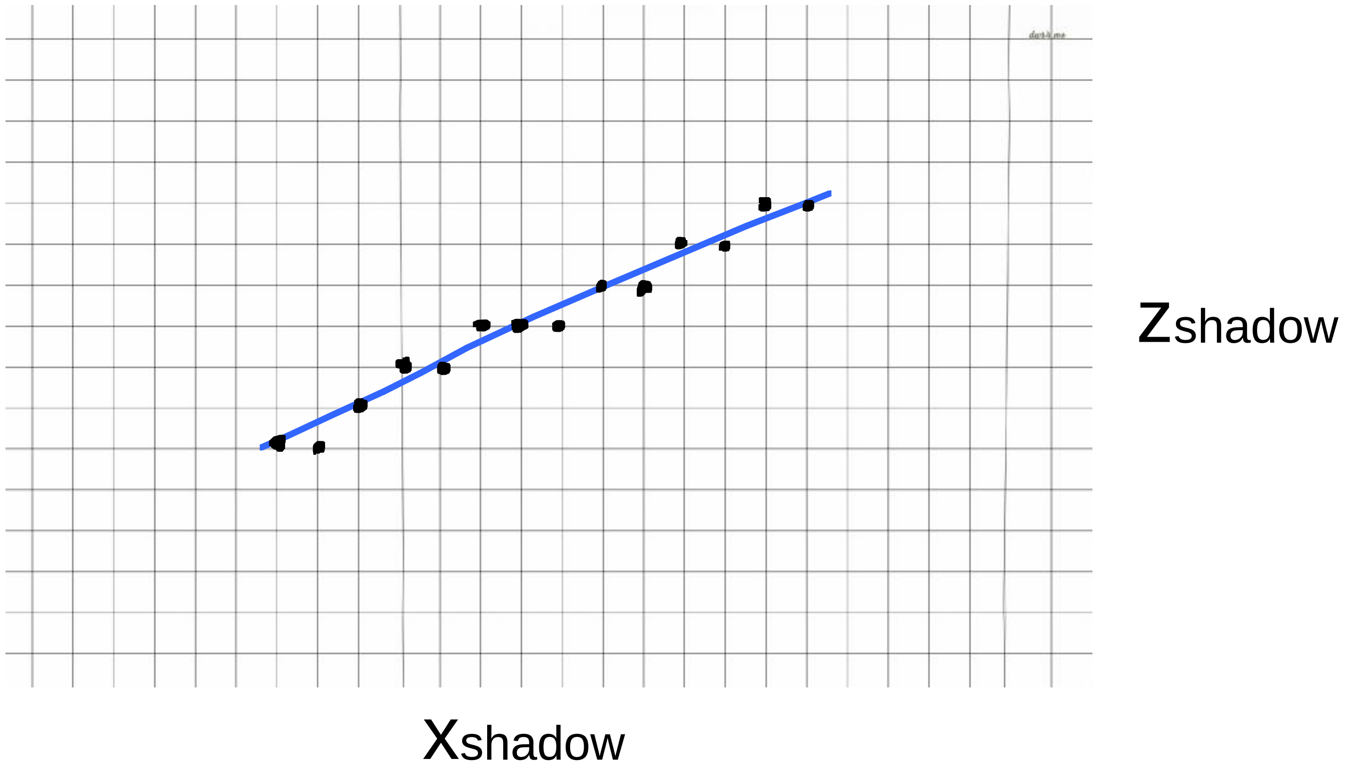
Shading Mapping and Aliasing



http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/

What causes the aliasing shown on the previous slide? Consider the 2D xz example below.

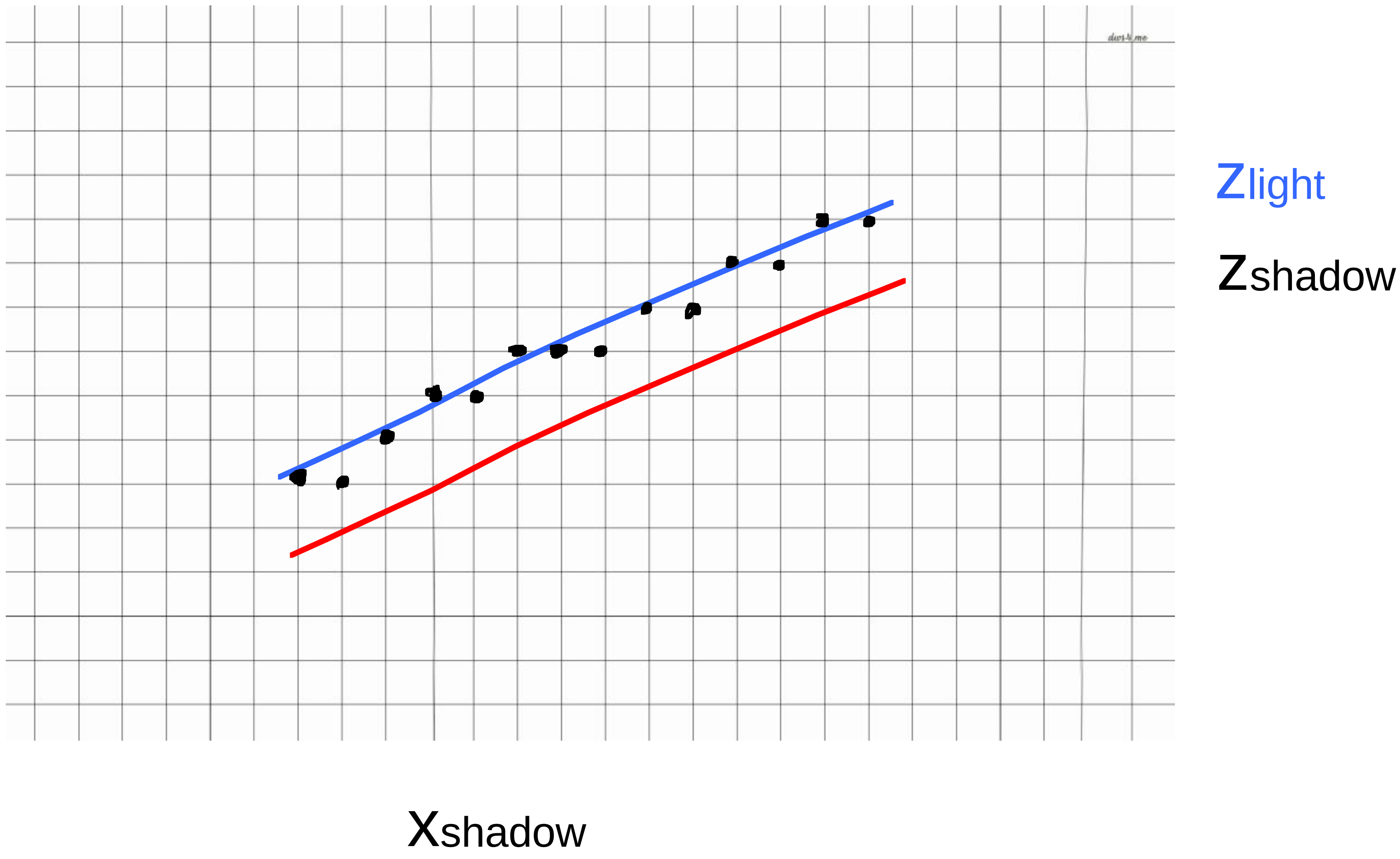
(x_{light}, z_{light}) is on a **continuous** visible surface (blue curve).
(x_{shadow}, z_{shadow}) is in the **discretized** shadow map (black points).
The shadow condition, $z_{shadow}(x_{shadow}) < z_{light}$, is supposed to be false because all points on the surface are visible. However, because of discretization, the condition is often true and the algorithm mistakenly concludes that some points are shadowed.

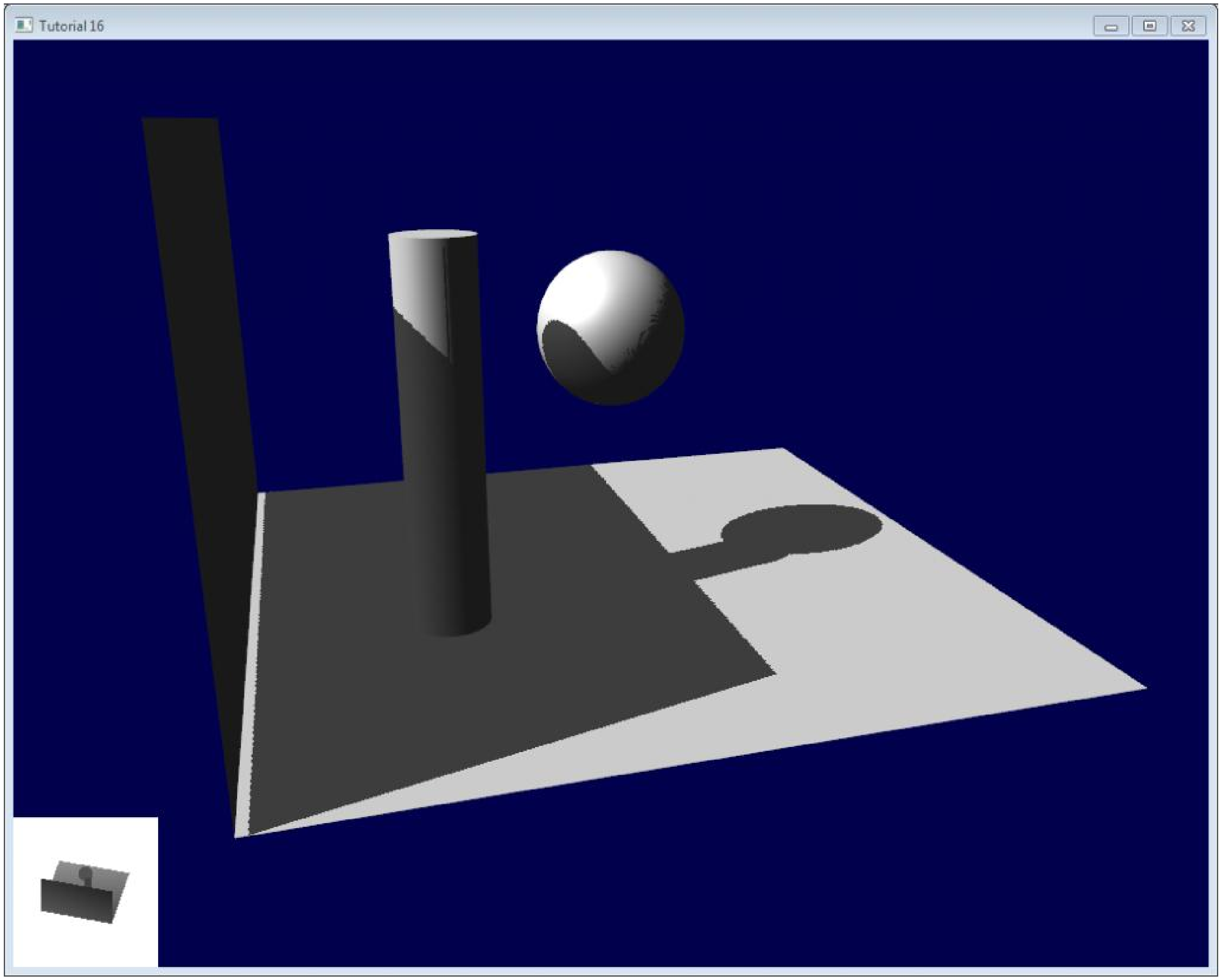


To conclude that (x_{light} , z_{light}) is in shadow, in the presence of discretization, we require that a stronger condition is met:

$$z_{shadow}(x_{shadow}) < z_{light} - \varepsilon .$$

However, as we show on next slide, this can lead us to conclude that a point is not in shadow when in fact it is in shadow.

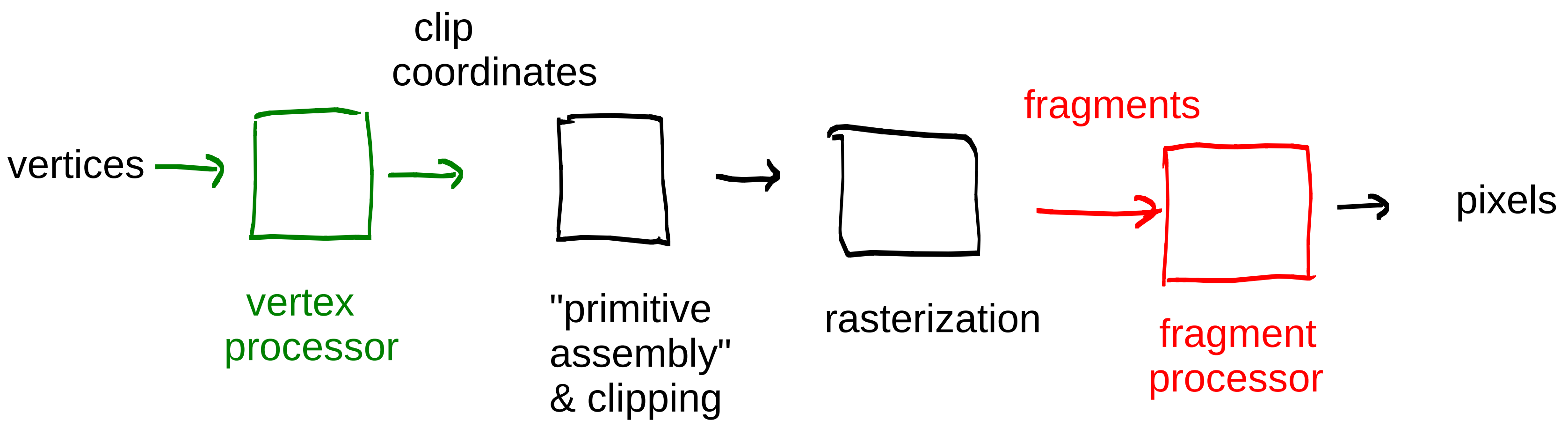




In fact, there is no gap between ground and vertical wall.
Yet algorithm allows light to leak under the wall.
It fails to detect this shadow.

Real Time Rendering

How are shadows computed in the OpenGL pipeline ?



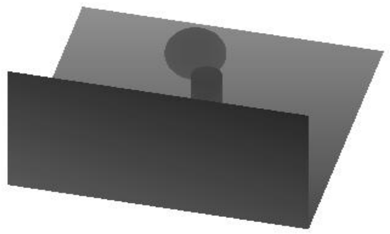
Pass 1:   make shadow map and store as a texture

Pass 2:   make RGB image using shadow map

Pass 1 (Compute shadow map)

vertex shader - transform vertices to light coordinates
$(x_{light}, y_{light}, z_{light})$

rasterizer - for each light source pixel (x_{shadow}, y_{shadow})
find depth z_{shadow} of closest surface



fragment shader
- store depths as a texture $z_{shadow}(x_{shadow}, y_{shadow})$

Pass 2 (make RGB image with shadows)

vertex shader

- transform vertex to camera coordinates (x_p, y_p, z) <u>and</u> to light source coordinates $(x_{light}, y_{light}, z_{light})$ // as in pass 1

rasterizer

- generate fragments

  (each fragment needs $(\underbrace{(x_p, y_p), n, z}_{\text{camera}}, \underbrace{x_{shadow}, y_{shadow}, z_{light}}_{\text{light}})$

The rasterizer does not have access to the shadow map computed in the first pass.
The fragment shader (below) has access to it.

fragment shader

```
- if  $z_{shadow}(x_{shadow}, y_{shadow}) + \varepsilon < z_{light}$
      compute RGB using ambient only   //  in shadow
  else
      compute RGB using Blinn-Phong    //  not in shadow
```

lecture 19

shadows and interreflections

- shadow maps

- ambient occlusion

- global illumination: interreflections

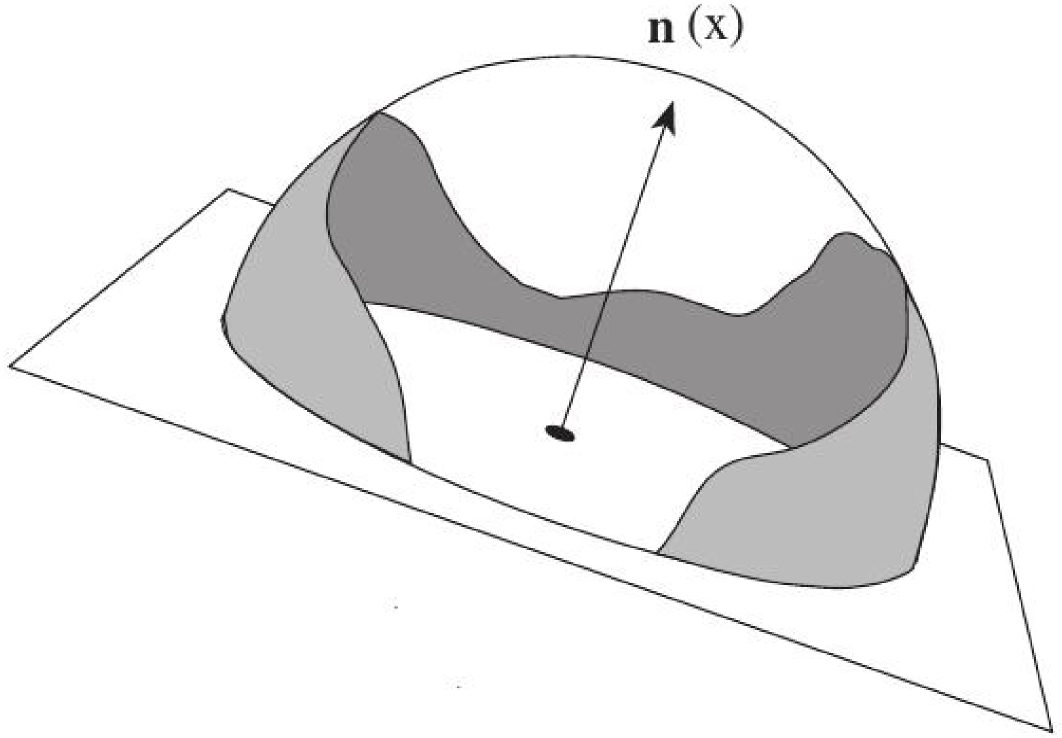
How to handle 'diffuse lighting' ?

- outdoors on an overcast day

- uniformly illuminated indoor scene

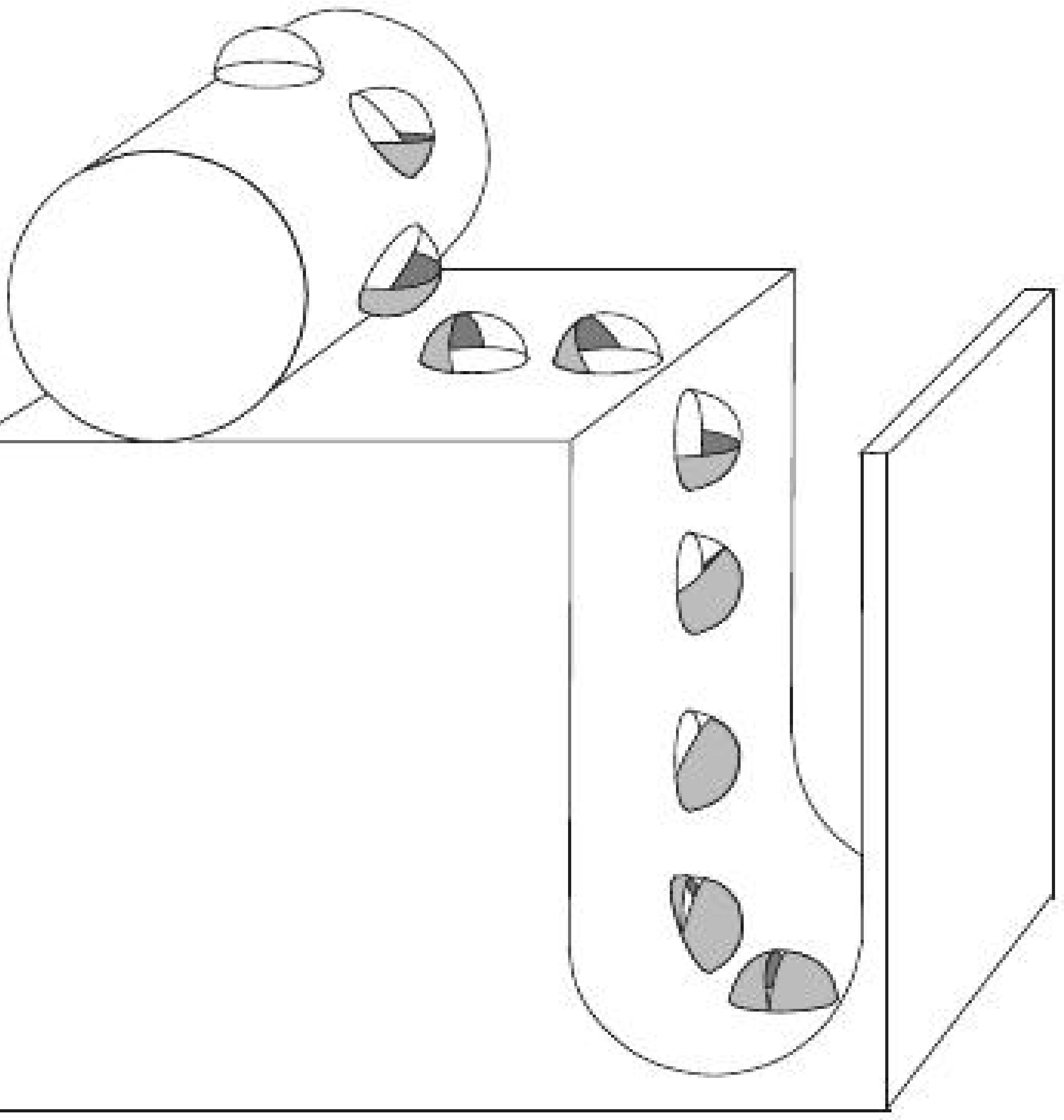
  e.g. classroom, factory, office, retail store

Visibility of the "sky"



$$I_{diffuse}(x) = \int_{S(x,\ell)} n(x) \cdot \ell \, d\ell$$

"Sky" visibility varies throughout scene



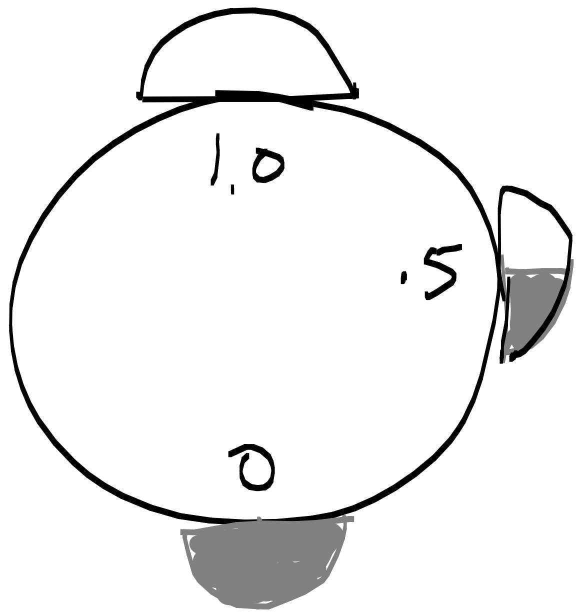
Solution 1: (cheap) use attached shadows only.

I skipped this in the lecture because I thought I was running out of time. See Exercises.

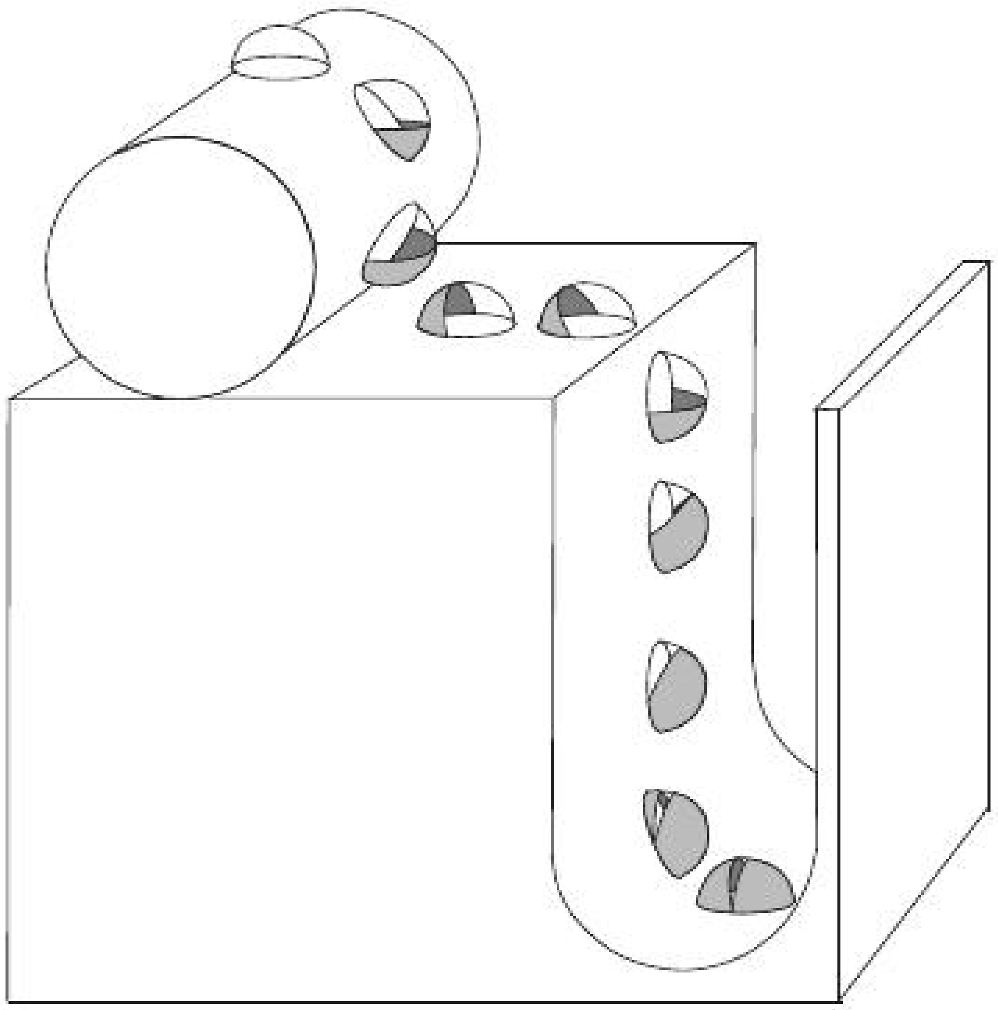
Assume :

- the light source is a uniform distant hemisphere
- the fraction of the source is determined only by the surface normal.

$$I_{diffuse}(x) = \frac{1 + n(x) \cdot \hat{y}}{2}$$



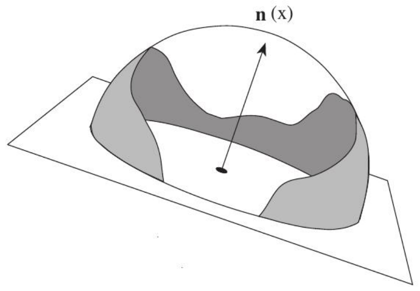
What is the differences of this model and "sunny day" ?



Key limitation: the model on the previous slide cannot account for cast shadows, e.g. illumination variations along the ground plane or along the planar side of the gully, since the normal is constant on a plane.

Solution 2: Ray tracing (expensive)

```
for each pixel in the image {

     cast a ray through the scene point to find the nearest
          surface (x,y,z)

     shoot out rays from (x, y, z) into the hemisphere
         and check which of them reach the "sky"
         i.e. infinity  or some finite distance

     add up environment contribution of rays that reach the "sky"
         //  you could have a non-uniform sky
}
```



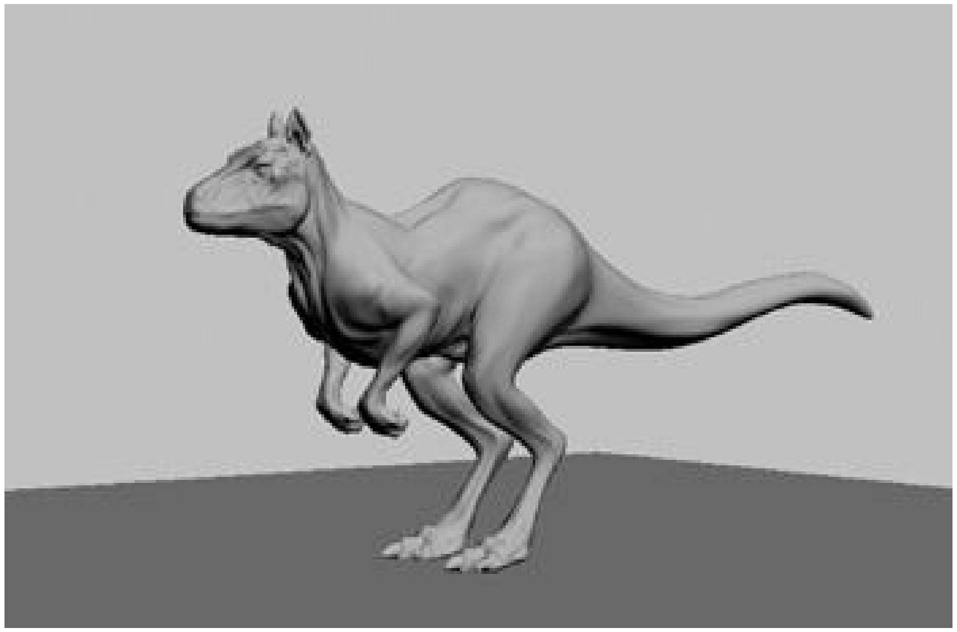
Solution 3: Ambient Occlusion [Zhukov,1998]

```
//   precompute

for each vertex  x {

        shoot out rays into the hemisphere and calculate the
            fraction of rays,   S(x) in [0, 1],  that reach the "sky"

        //  S(x) is an attribute of  x,   along with n and material

        //   If you are willing to use more memory,  then store the
        //   a boolean map  S(x, l),   where l is direction of light
        //    See Exercises.
}


//   We say that S(x) is "baked" into the surface.
```

http://http.developer.nvidia.com/GPUGems/gpugems_ch17.html

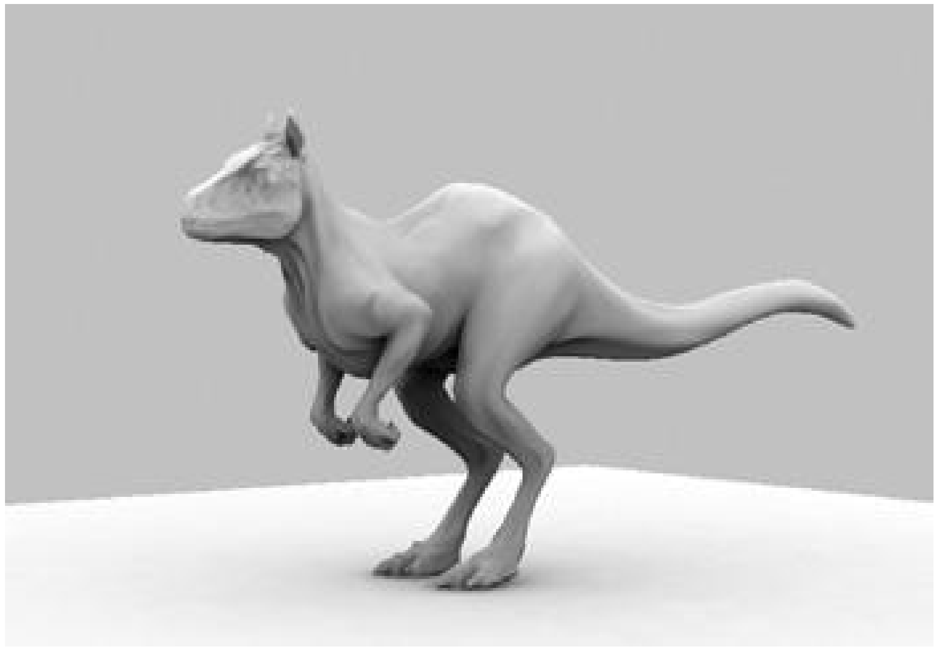
$I_{\text{diffuse}}(x) = n(x) \cdot I$

This example has no shadows, point source at upper right, and uniform reflectance



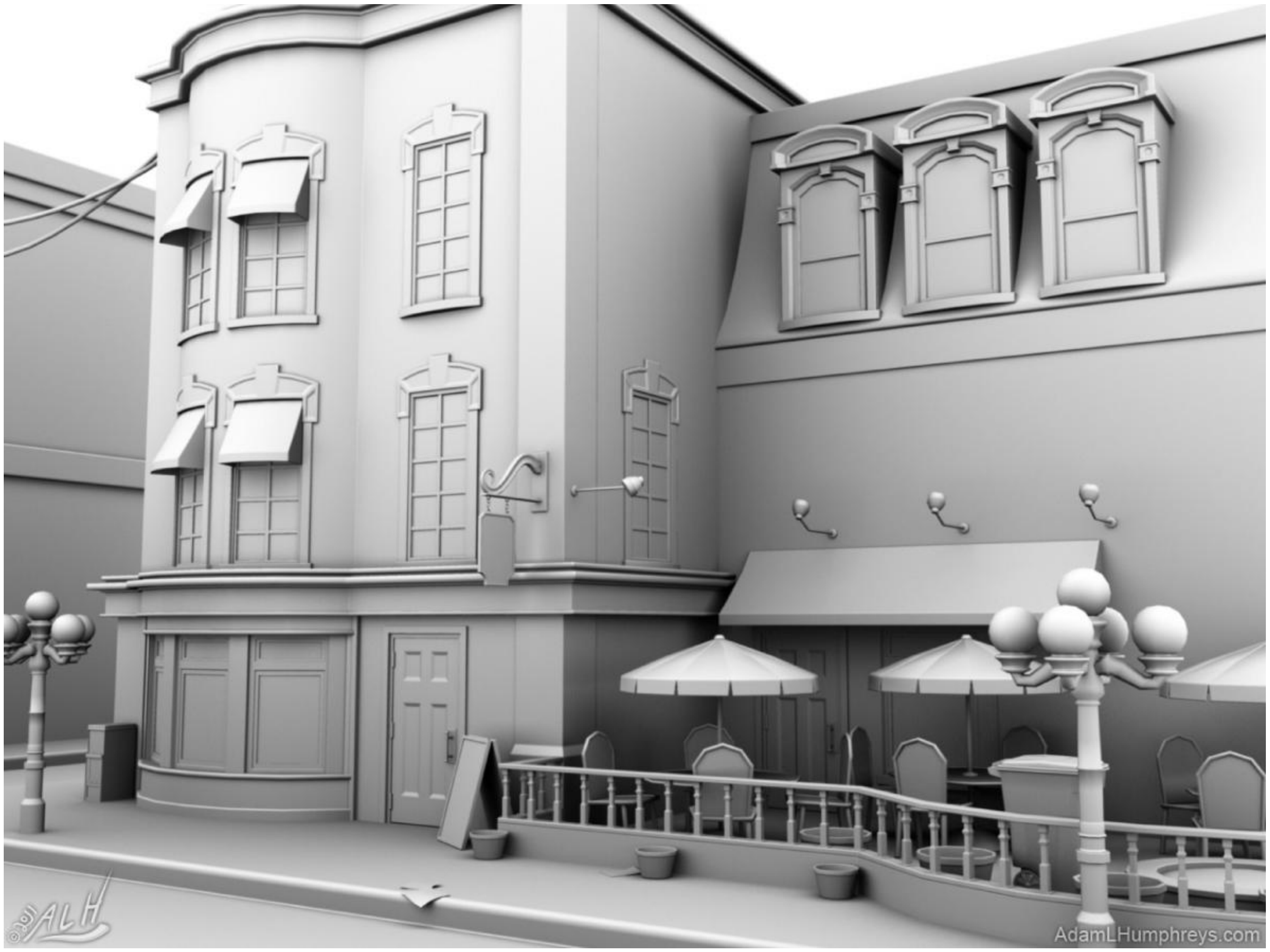
$I_{\text{diffuse}}(x) = S(x)$

Ambient occlusion can replace n(x).I term in more general Blinn-Phong model instance.
Compare the far leg here with the above example.



This allows for real-time rendering (moving the camera).

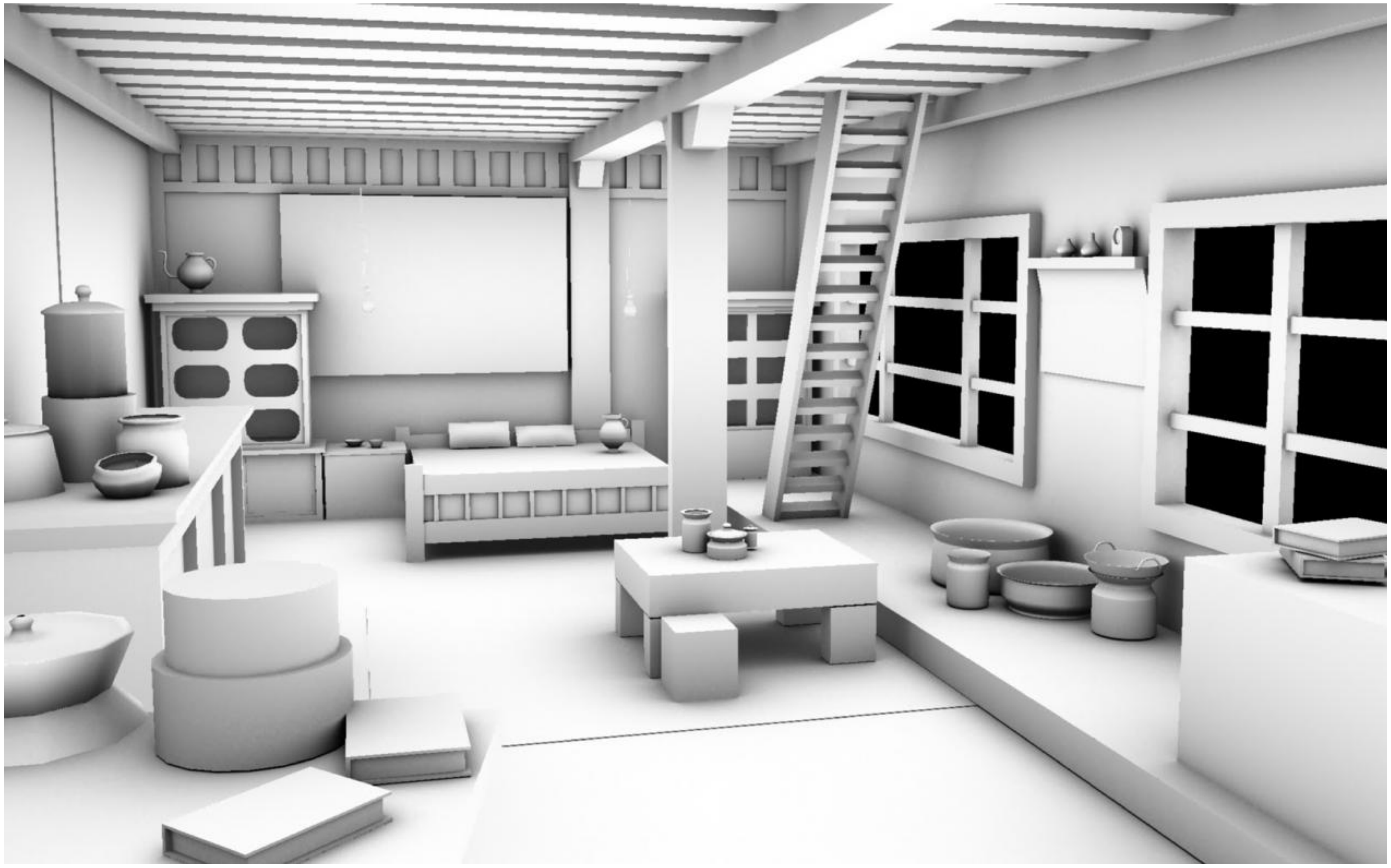
Examples of ambient occlusion

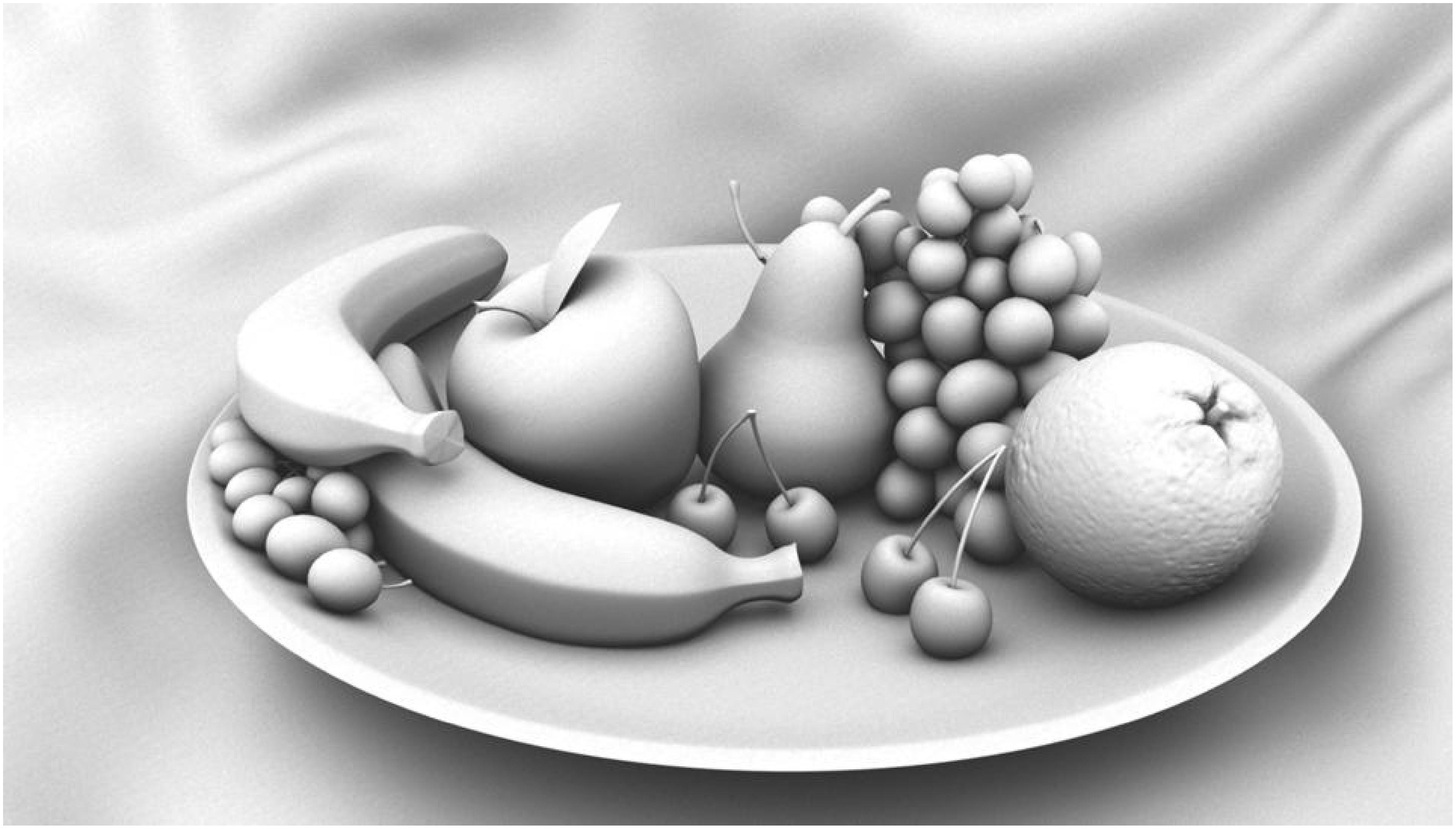


http://www.adamlhumphreys.com/gallery/lighting_ca/19

Q: How to do ambient occlusion with indoor scenes ?

A: For each vertex, compute $S(x)$ in $[0, 1]$ by considering only surfaces within some distance of x.





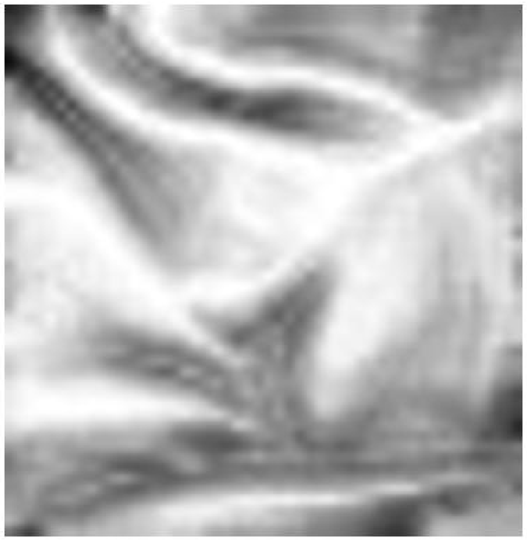
Ambient Occlusion in My Own Research

No, I will not ask about this (or the next two slides) on the final exam.

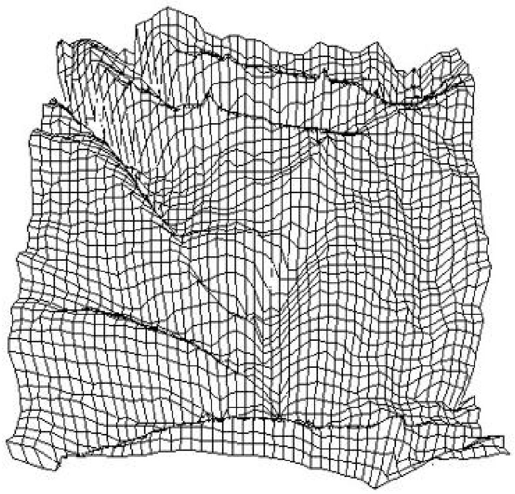
Ph.D. thesis

- "Shape from shading on a cloudy day" (Langer & Zucker, 1994)

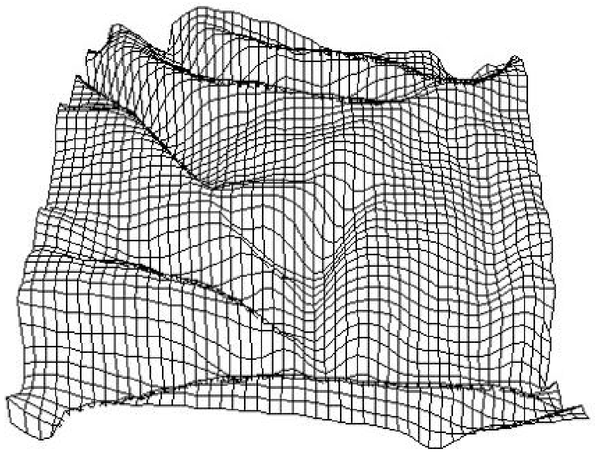
I independently discovered the principle of ambient occlusion.
I used it to introduce a new version of a classic computer vision problem (shape from shading).



input image (rendered)



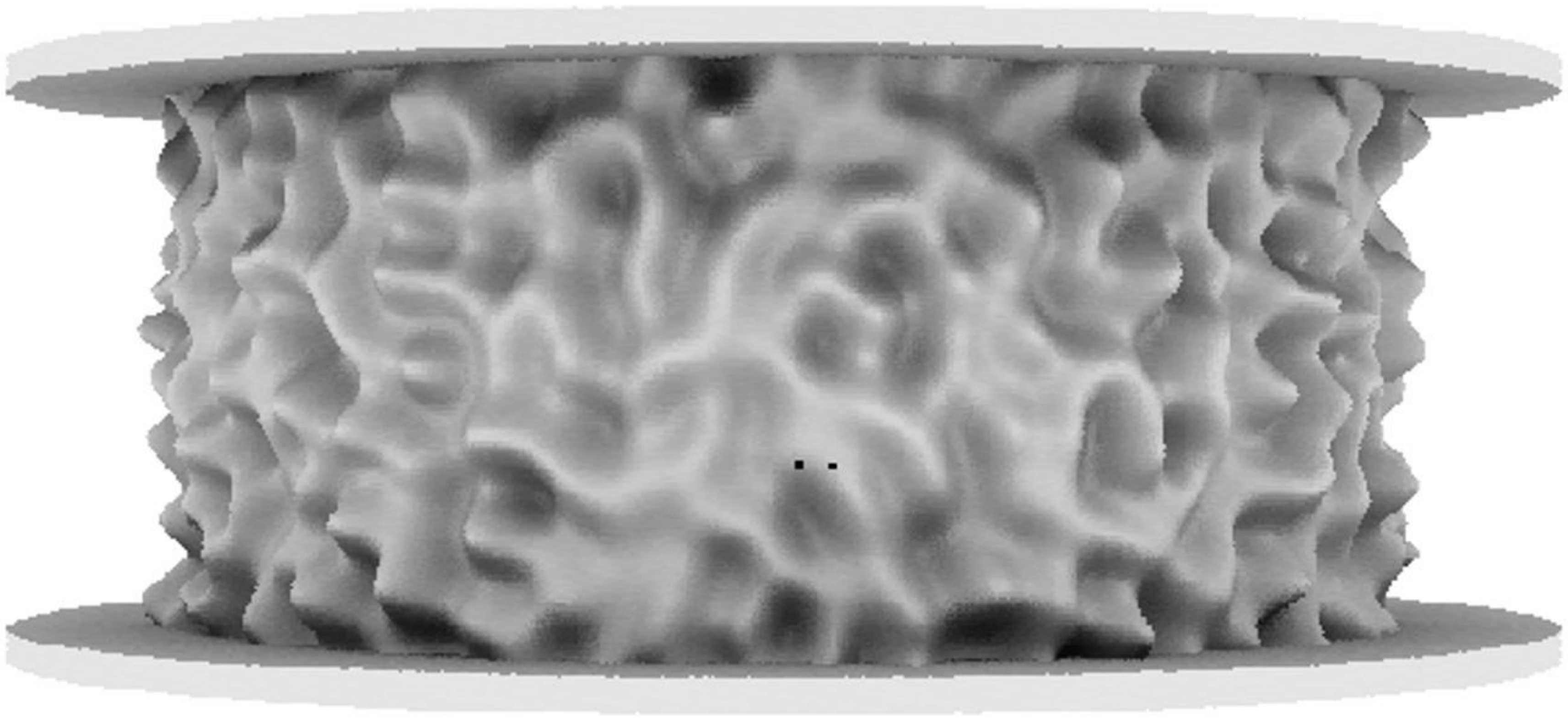
computed mesh



true mesh

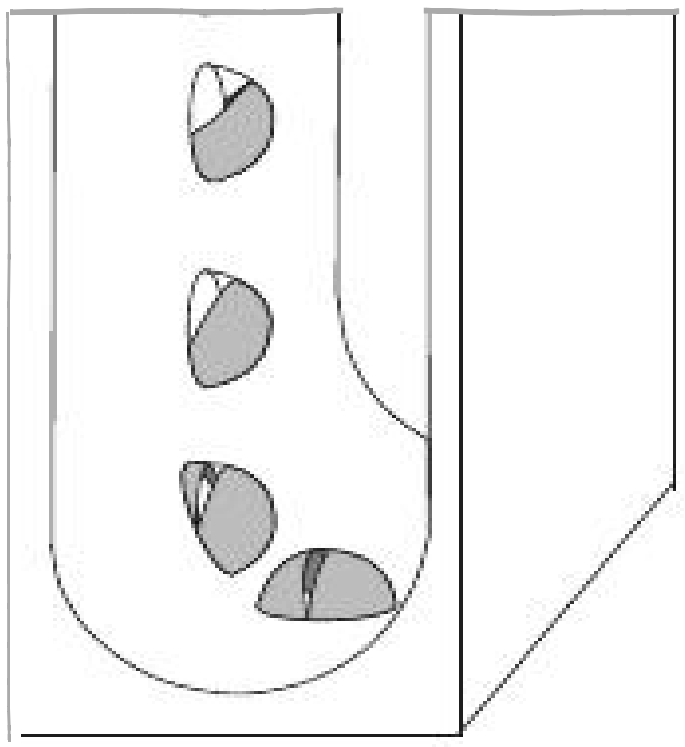
Post-doctoral Research [Langer & Buelthoff, 2000]

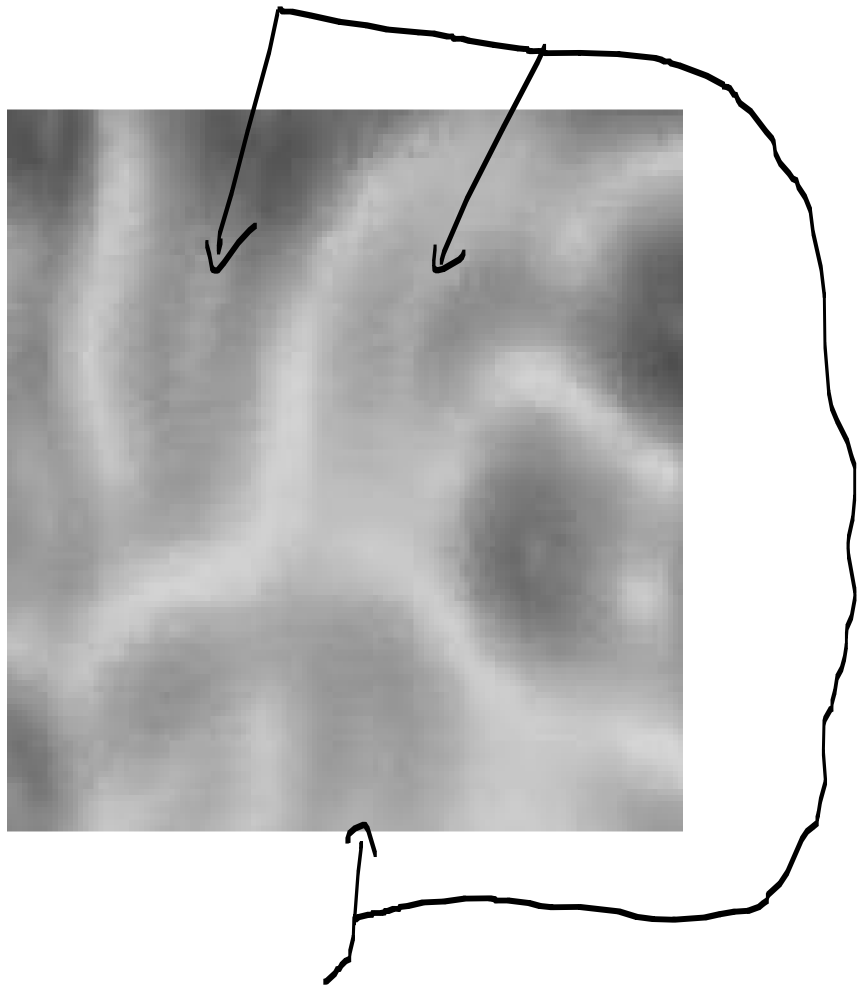
- I carried out the first shape perception experients that compared images rendered with vs. without ambient occlusion.



with ambient occlusion

I skipped this slide in the lecture to save time.





Local intensity maxima occur in valleys where surface normal turns to face the visible part of the light source.

One of our experiments looked at whether people (and computer vision algorithms) misinterpret these as local "hills".

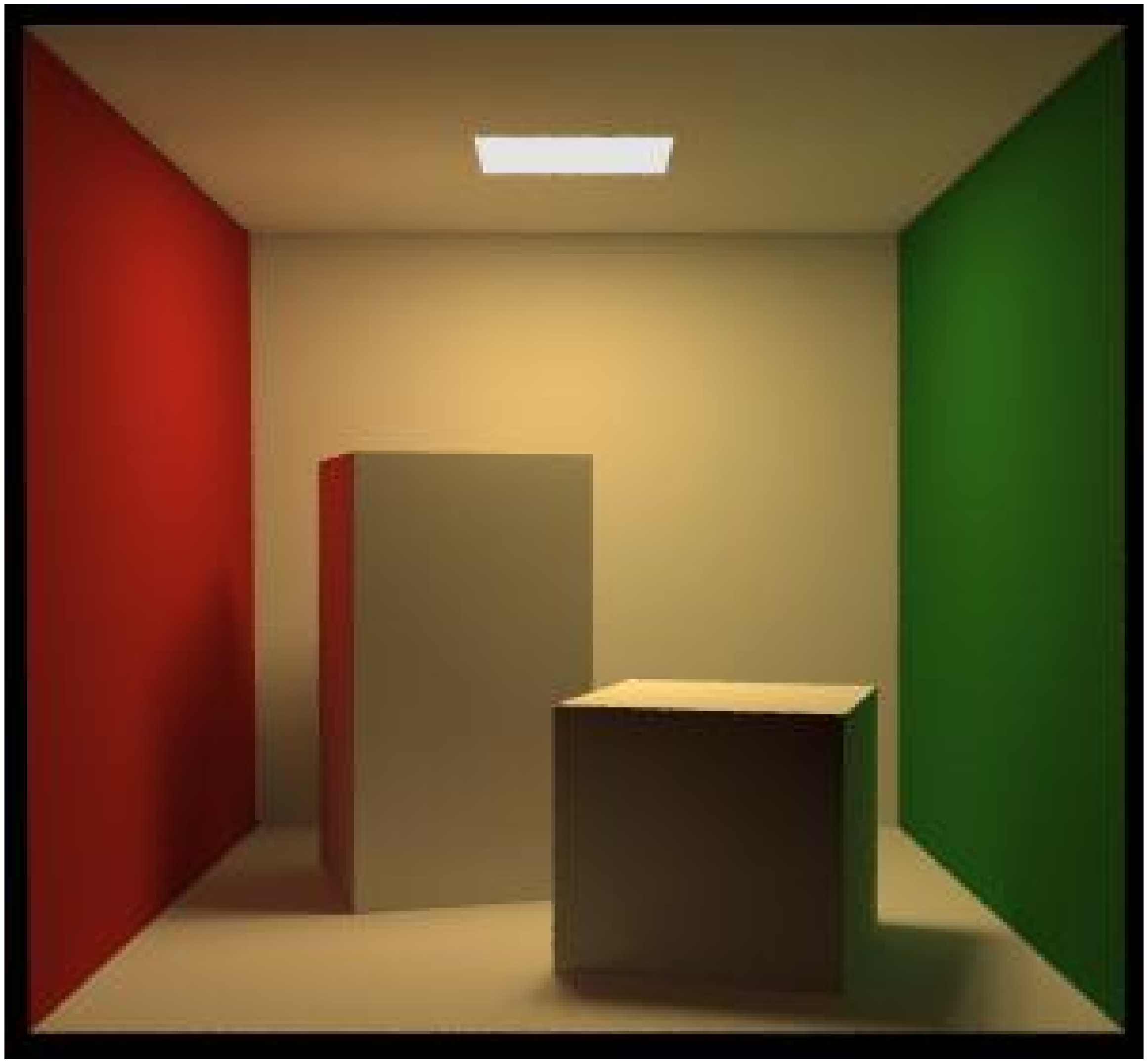
lecture 19

Shadows

- ray tracing
- shadow maps
- ambient occlusion

Interreflections

Cornell Box (1984) http://www.graphics.cornell.edu/online/box/



Surfaces are illuminated directly by light sources, but also indirectly by other surfaces. This leads to color bleeding. (Red and green walls bleed color onto cubes.)

Interreflections (sketch only)

Computing interreflections is a linear algebra problem:

$I(x) = I_{direct}(x) + I_{indirect}(x)$

But $I_{indirect}(x)$ is the sum of $I(y)$ for points y that are visible from x. Thus,

$$I(x) = I_{direct}(x) + \sum_{y} I(y) \, F(x,y)$$

where F(x,y) depends on:

- distance between x and y
- n(x) and n(y)
- whether x and y see each other

$$I(x) = I_{direct}(x) + \sum_{y} F(x,y)\; I(y)$$

$$\begin{bmatrix} \\ I \\ \\ \end{bmatrix} = \begin{bmatrix} I_{direct} \\ \\ \\ \end{bmatrix} + \begin{bmatrix} & & \\ & \text{"Form factor"} & \\ & \text{matrix F} & \\ & & \end{bmatrix} \begin{bmatrix} \\ I \\ \\ \end{bmatrix}$$

These are huge vectors and matrices. Researchers have developed many clever numerical methods for solving this.

These interreflection methods ("global illumination") can now produce photorealistic images of complex scenes.

Solutions are still expensive though...