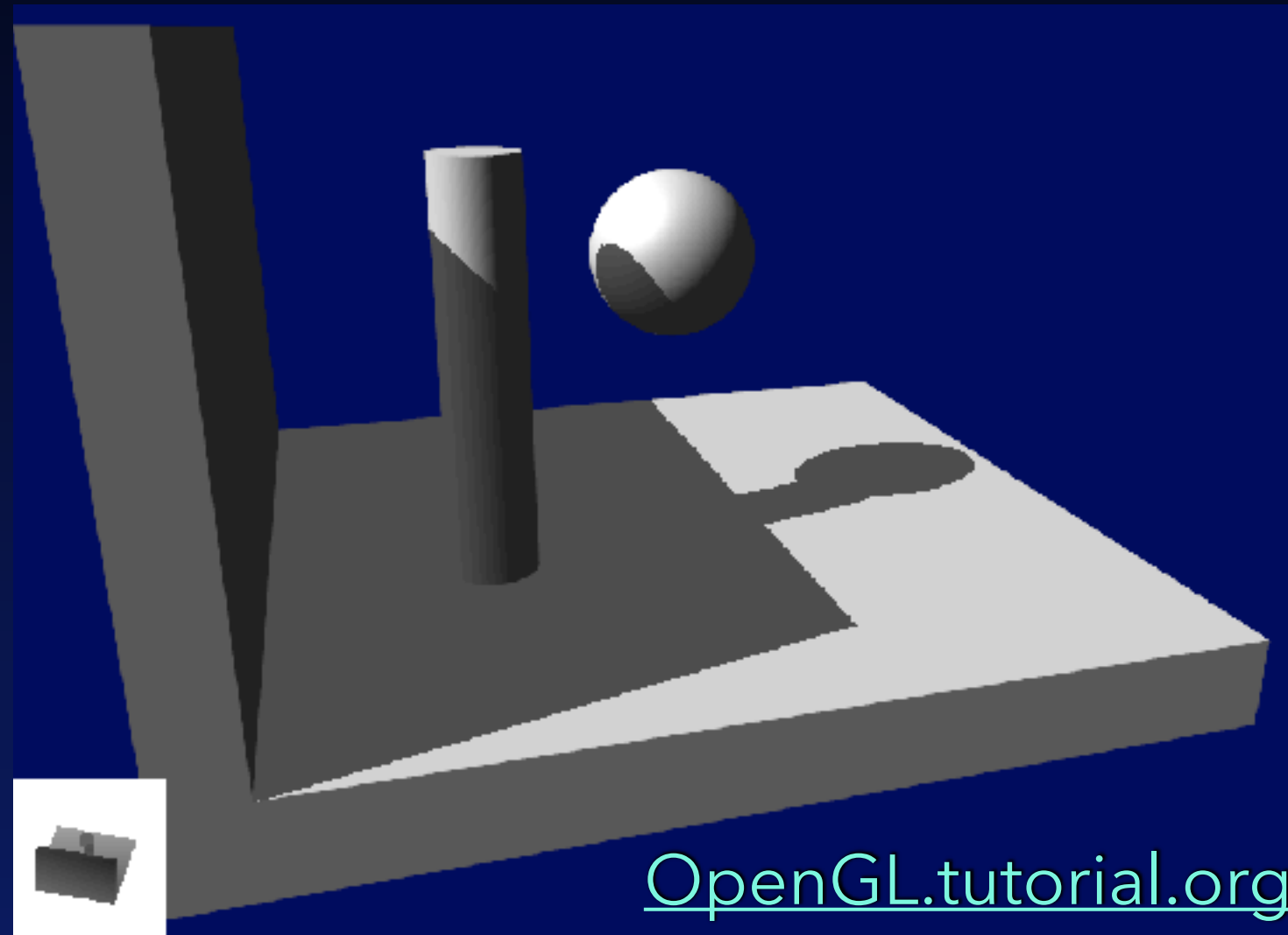


Computer Graphics

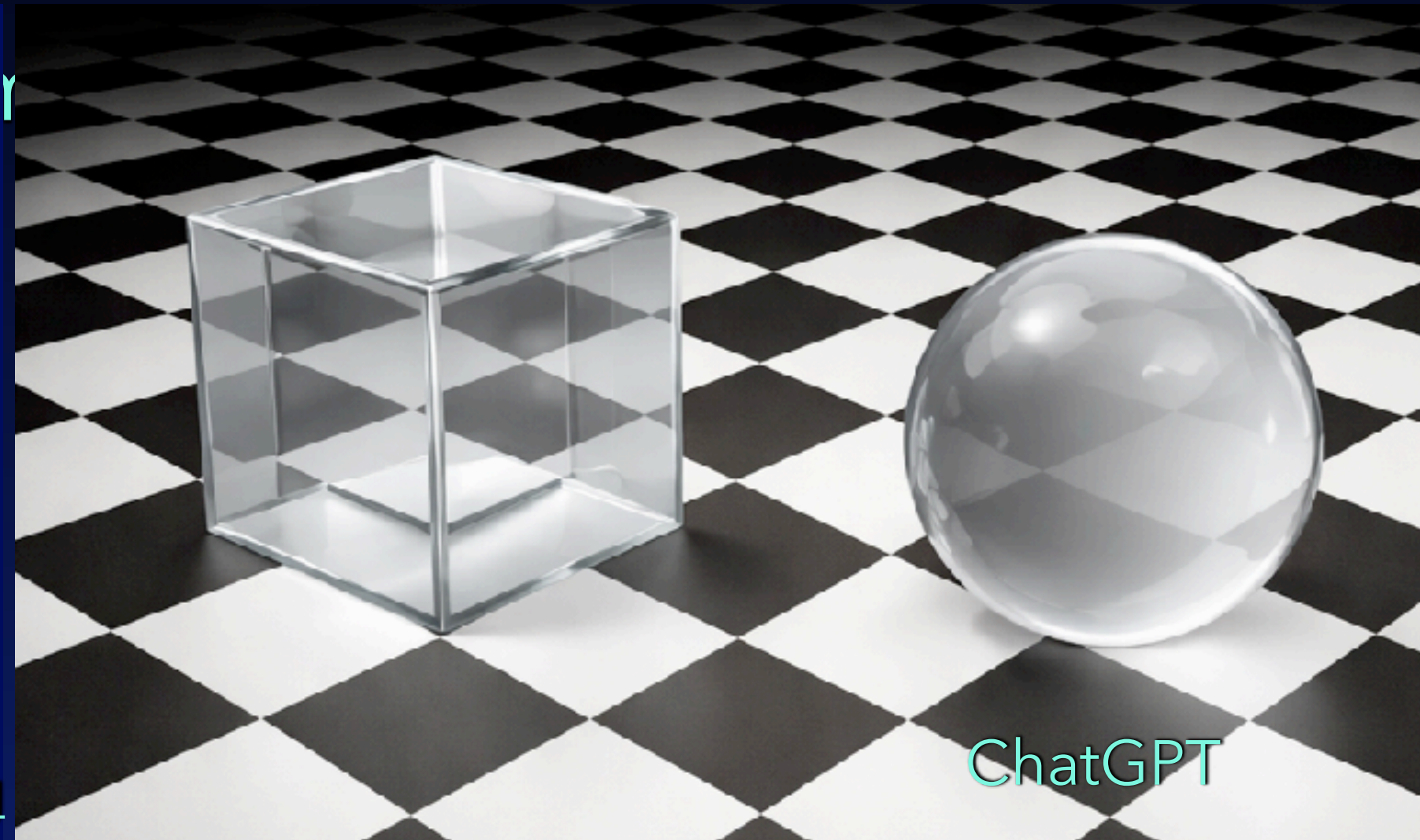
Shadows

Subodh Kumar, Dept of Computer Sc. & Engg., IIT Delhi

Effects



Shadow

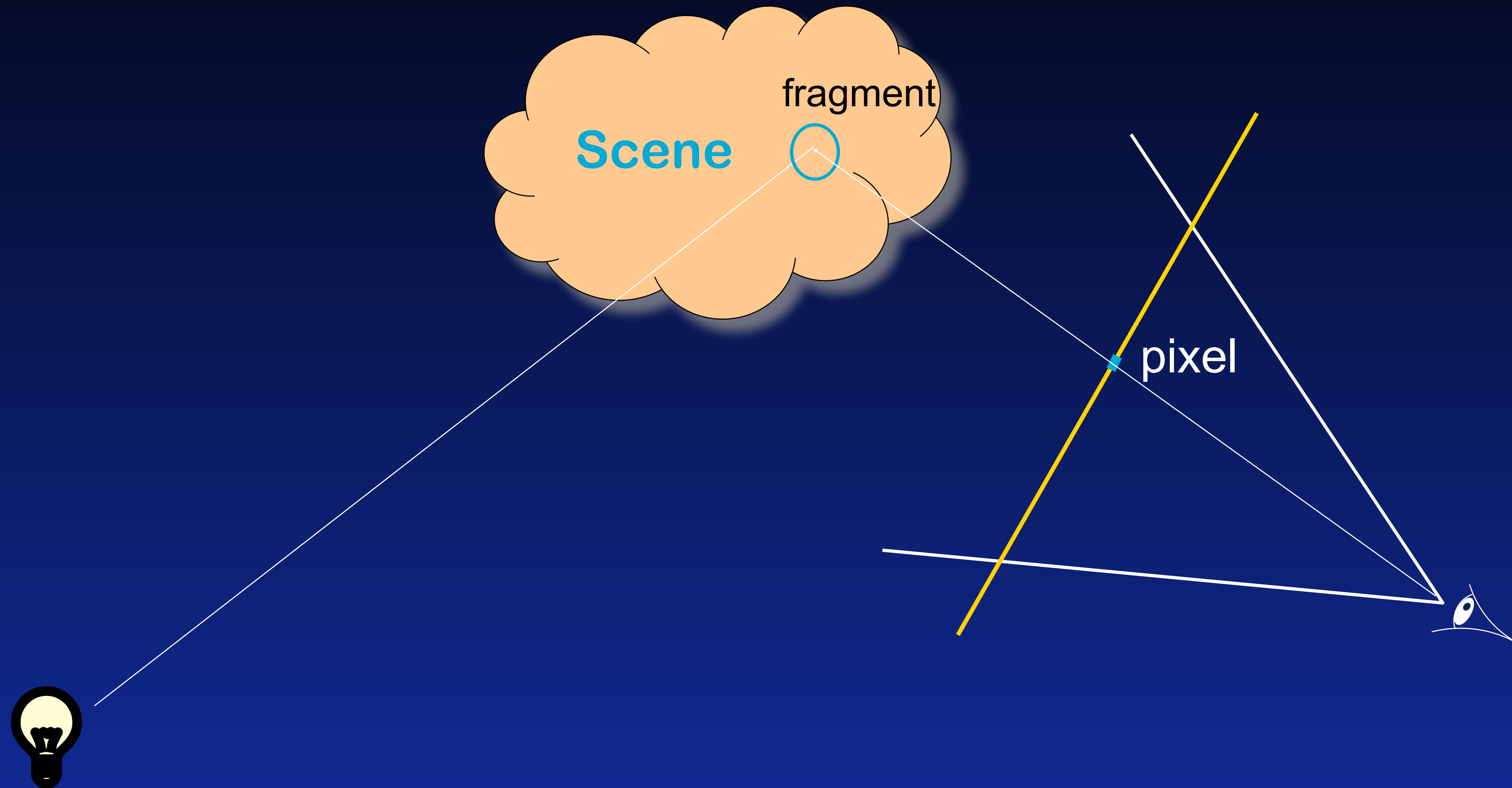


Transparency

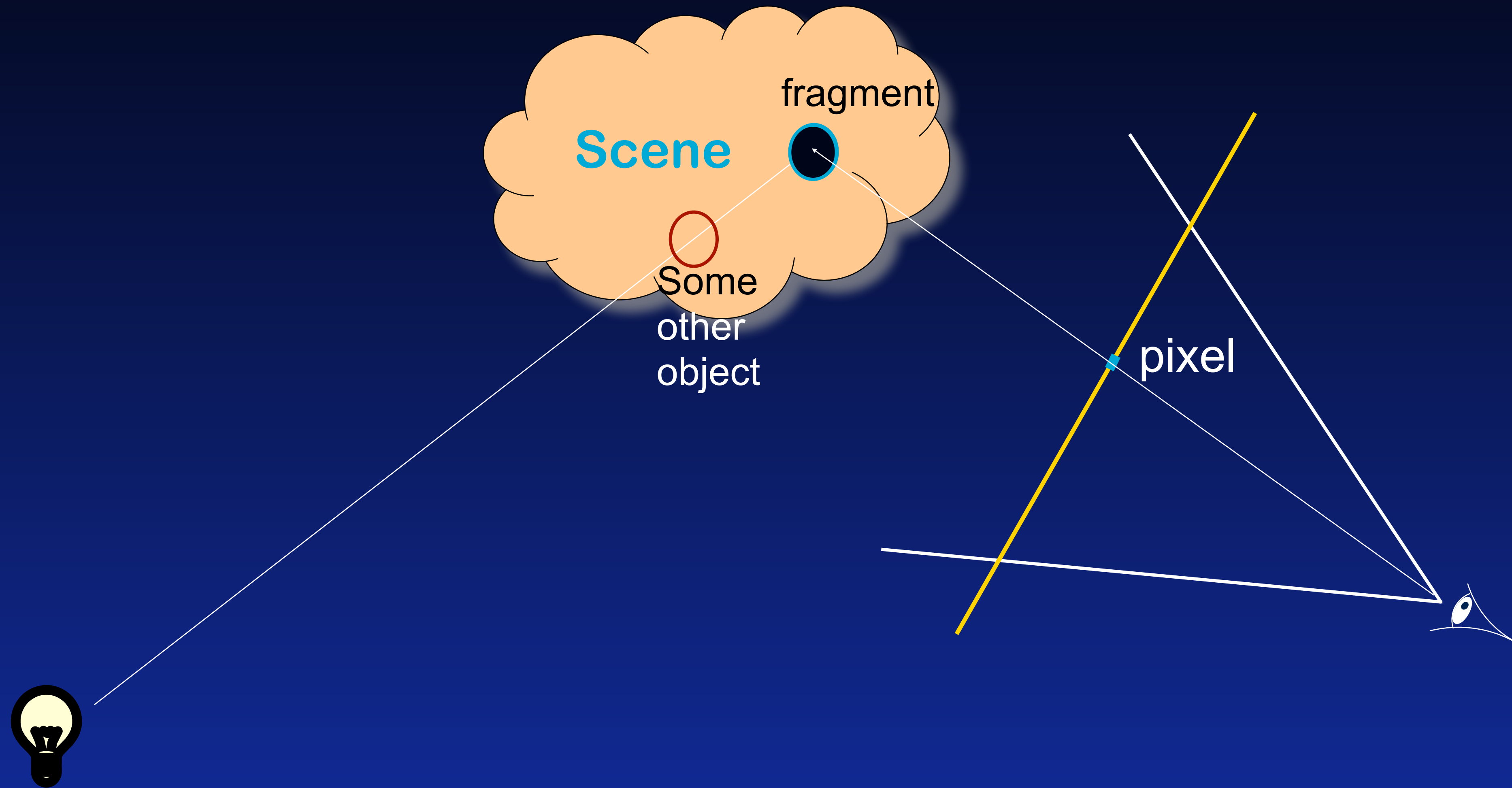


Ambient Occlusion
(Fraction of ambient light
occluded by other objects)

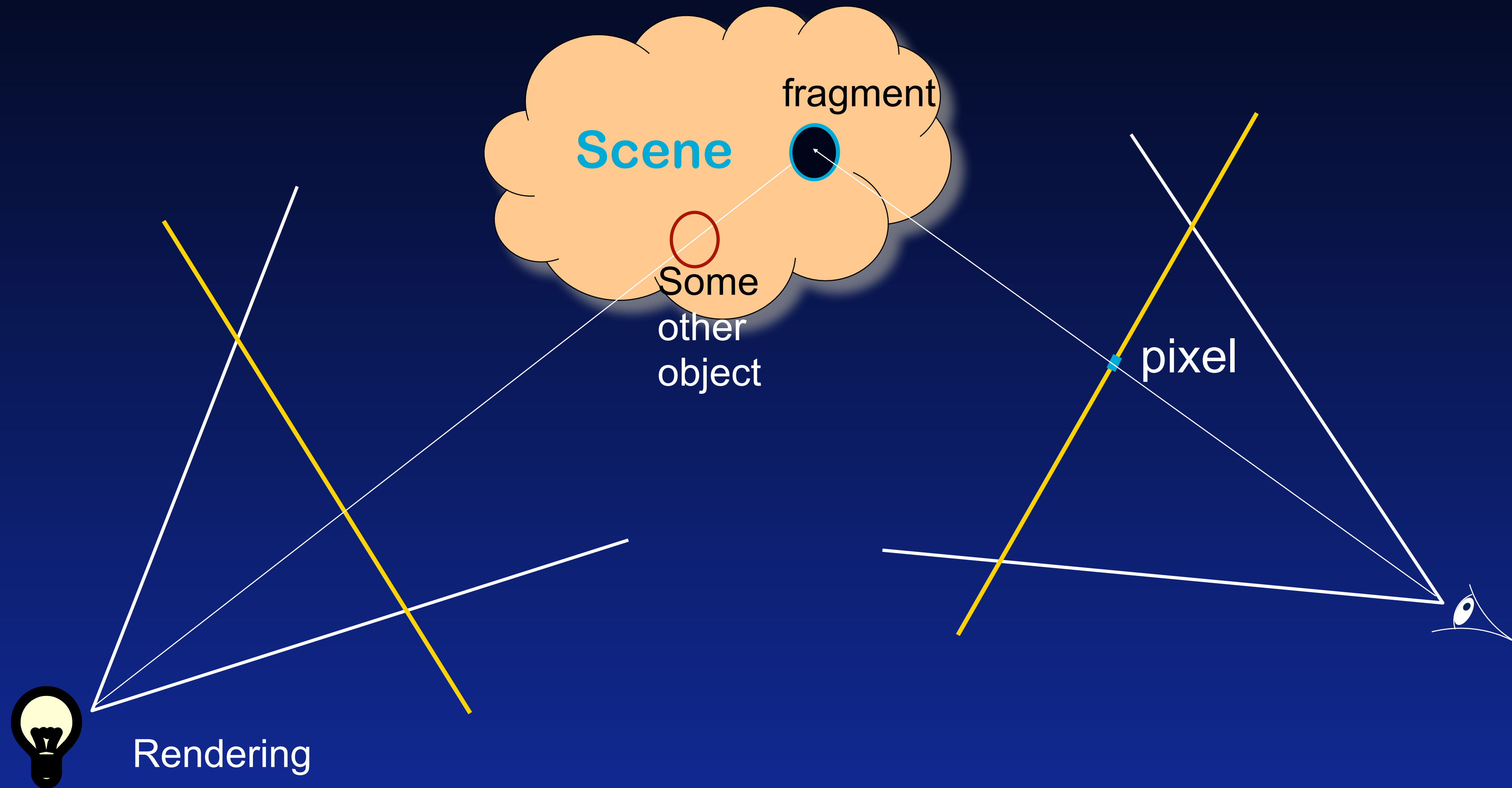
Shadow Map



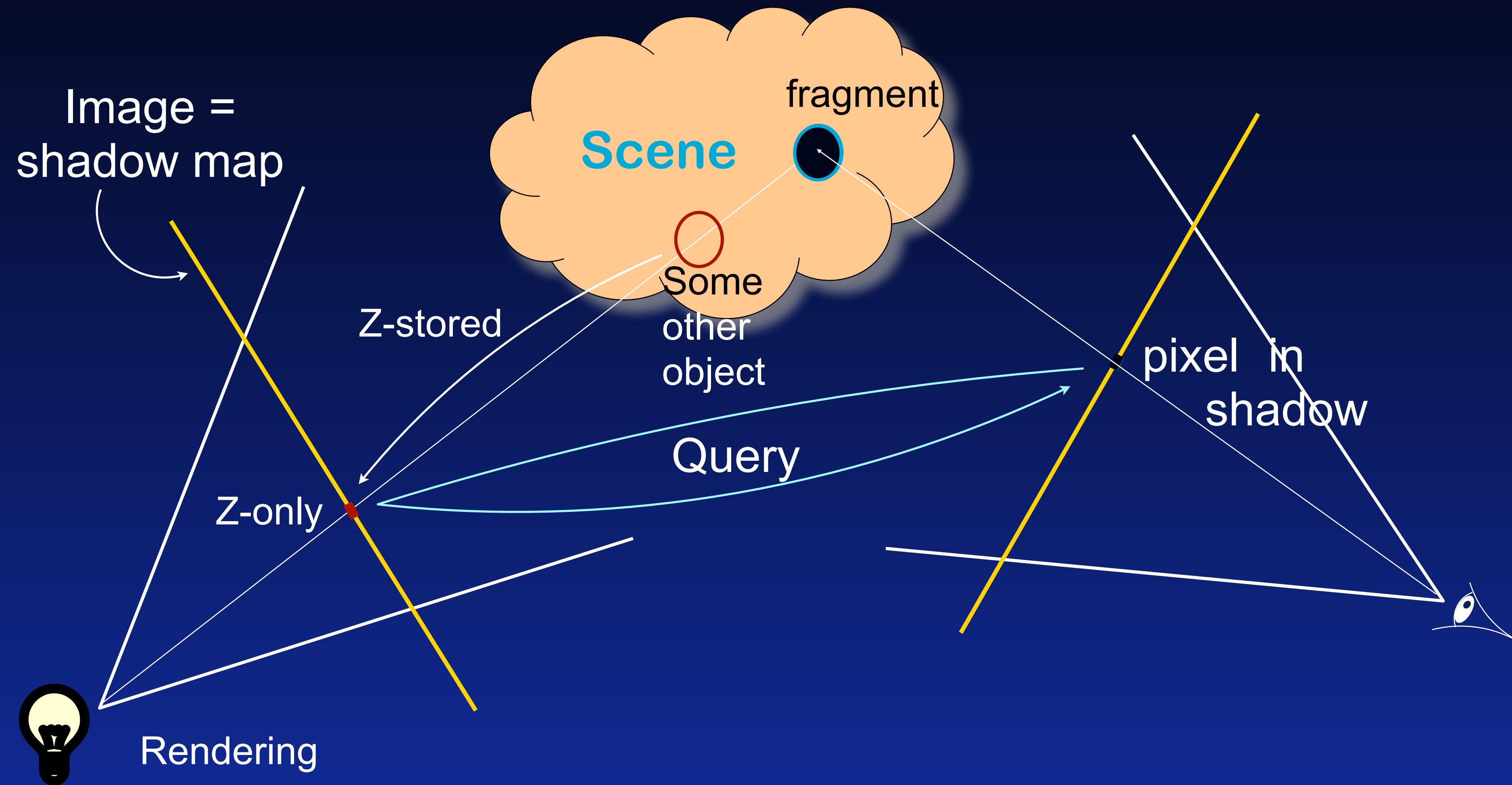
Shadow Map



Shadow Map



Shadow Map



Shadow Map Generation



- Render to texture
 - Depth only (see `glDrawBuffer()`)
- Light-camera position is that of light
- Direction:
 - Image should cover rays in all directions
 - Cube map: use six directions towards faces of a cube centered on the camera
 - This also suggests the field of view: 90°
- Near and far?
 - Clipping could be a problem, choose so occluders are not clipped
- View-port/Framebuffer size?
 - 'Large enough'

Shadow Map Generation



- Render to texture
 - Depth only (see `glDrawBuffer()`)
- Light-camera direction
- Direction:
 - Image should cover rays in all directions
 - Cube map: use six directions towards faces of a cube centered on the camera
 - This also suggests the field of view: 90°
- Near and far?
 - Clipping could be used
- View-port/Framebuffer
 - 'Large enough'

For render to texture, review
`glFramebufferTexture2D(GL_FRAMEBUFFER,
GL_DEPTH_ATTACHMENT, ..`

Also: geometry shader has a feature that can
render to all faces in the same pass:

See variable `gl_layer`

Shadow Map Look-up



- Find x, y to look up in shadow map
 - **Unproject** fragment (i, j) to Light space
 - Project into light's image space (say x, y, z)
- x and y may not be integers
 - But use standard texture interpolation
- Fetch Depth buffer/texture ($SM.z$)
 - Compare projected z with stored z
 - $SM.z < z \Rightarrow$ fragment in shadow

Shadow-Map in OpenGL

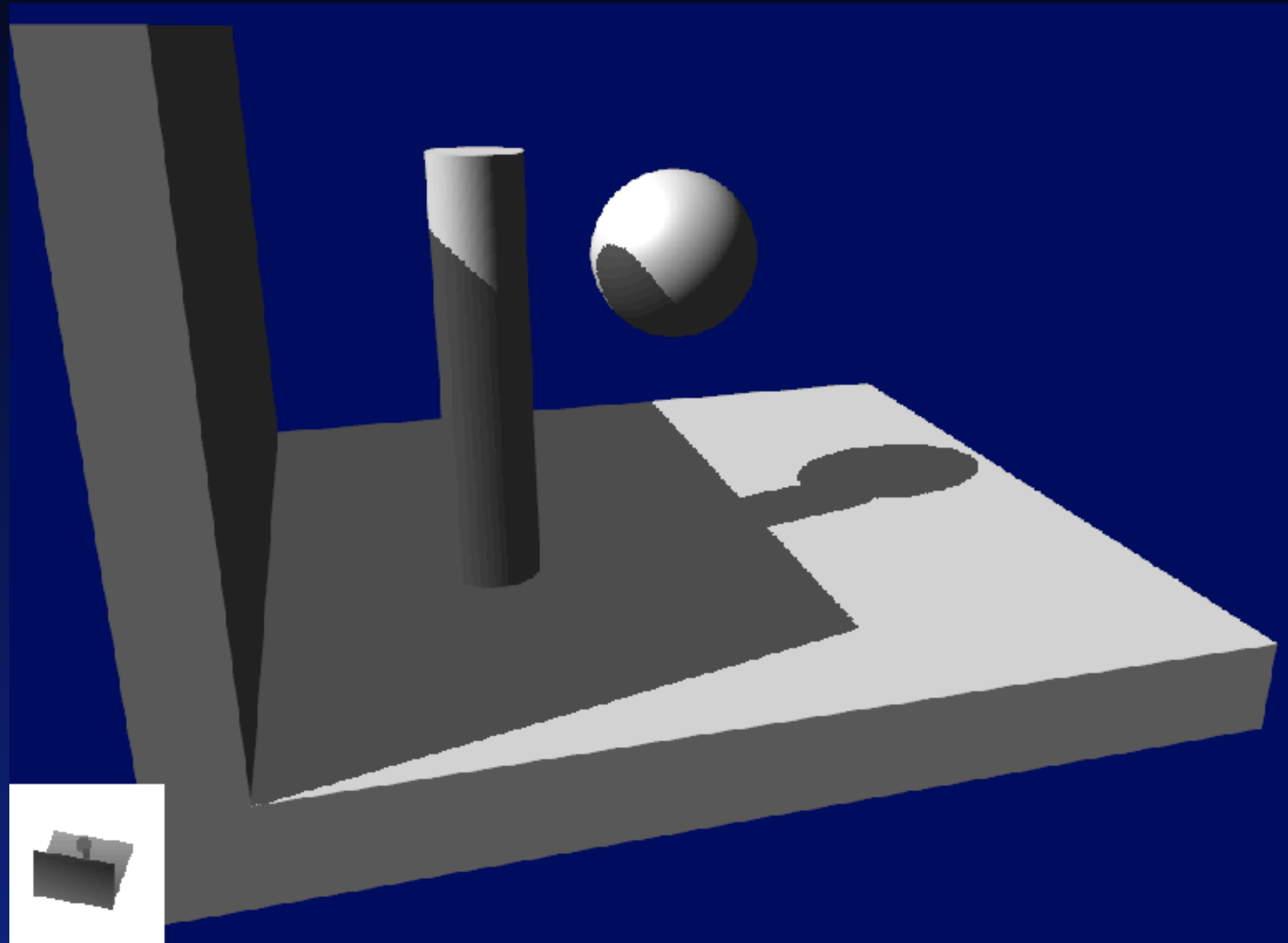


- Create FBO, say SB
- Create a 2D texture/cubemap for point lights: with internal format like GL_DEPTH_COMPONENT attach at SB's GL_DEPTH_ATTACHMENT.
- Use sampler2DShadow or samplerCubeShadow
 - Set GL_TEXTURE_COMPARE_MODE to GL_COMPARE_REF_TO_TEXTURE
 - Set GL_LINEAR filtering (actually uses PCF for Shadow Samplers)

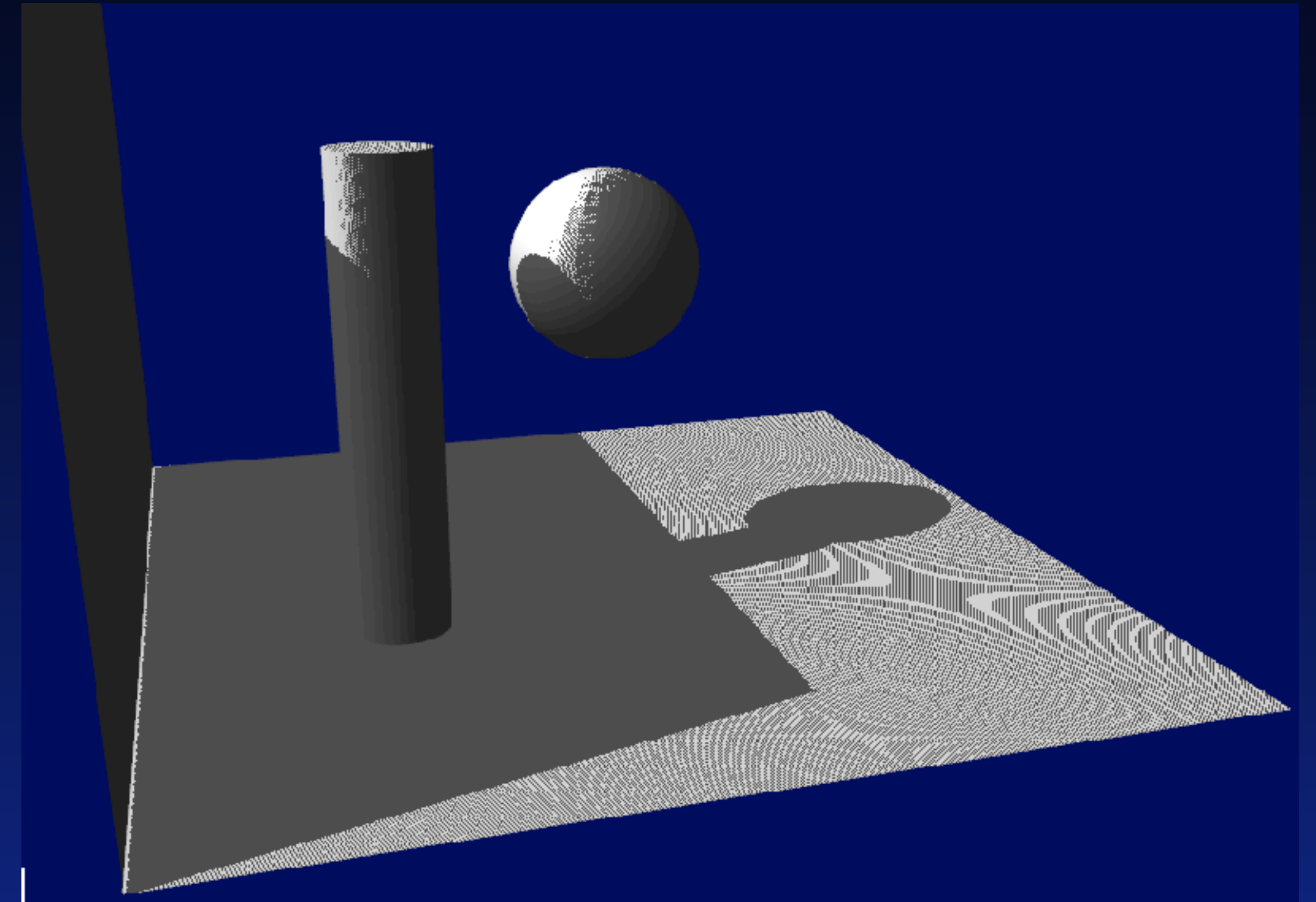
```
float shadow = texture(shadowMap, projCoords.xyz);
```

Samples Map, compares with z, and returns 0/1

Shadow Map Issues



Aliasing



Acne

Percent Closer Filtering



- Instead of prefiltering depth values, creating artificial geometry, ask “how much in shadow”



- Sample multiple texels in Depth Texture
 - Average 0/1 lookup
 - GL_LINEAR for shadow samplers implies this.

0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	1
0	0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1



Shadow Acne

`float bias = .01;?`

`float shadow = texture(shadowMap, vec3(projCoords.xy, projCoords.z - bias));`

- What is a good bias value?
 - Increase depth-map resolution by tight near-far range
 - Also see cascaded shadow maps



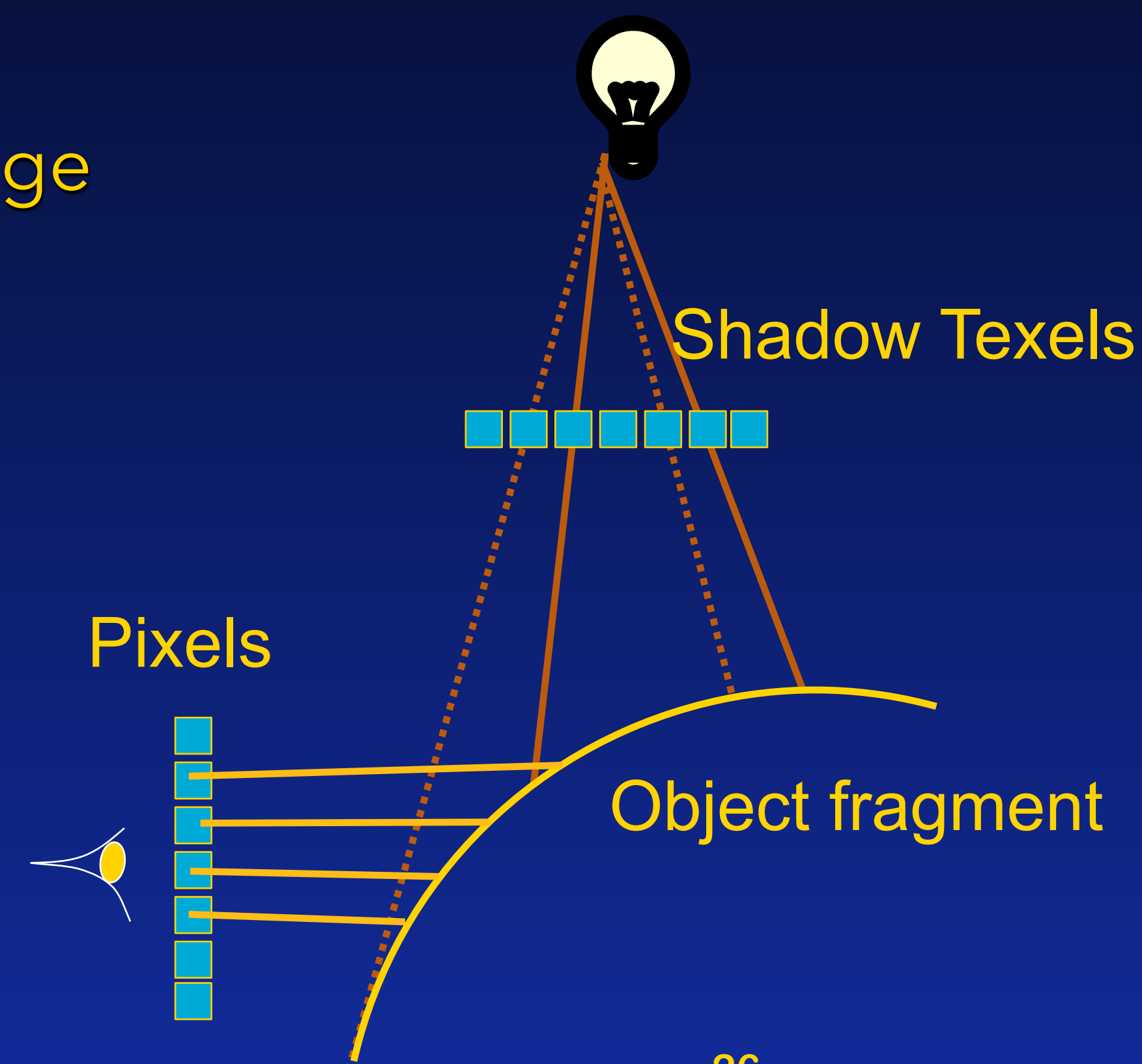
Shadow Acne

`float bias = .01;?`

`float bias = max(0.05 * (1.0 - dot(normal, lightDir)), 0.005); ?`

`float shadow = texture(shadowMap, vec3(projCoords.xy, projCoords.z - bias));`

- What is a good bias value?
 - Increase depth-map resolution by tight near-far range
 - Also see cascaded shadow maps





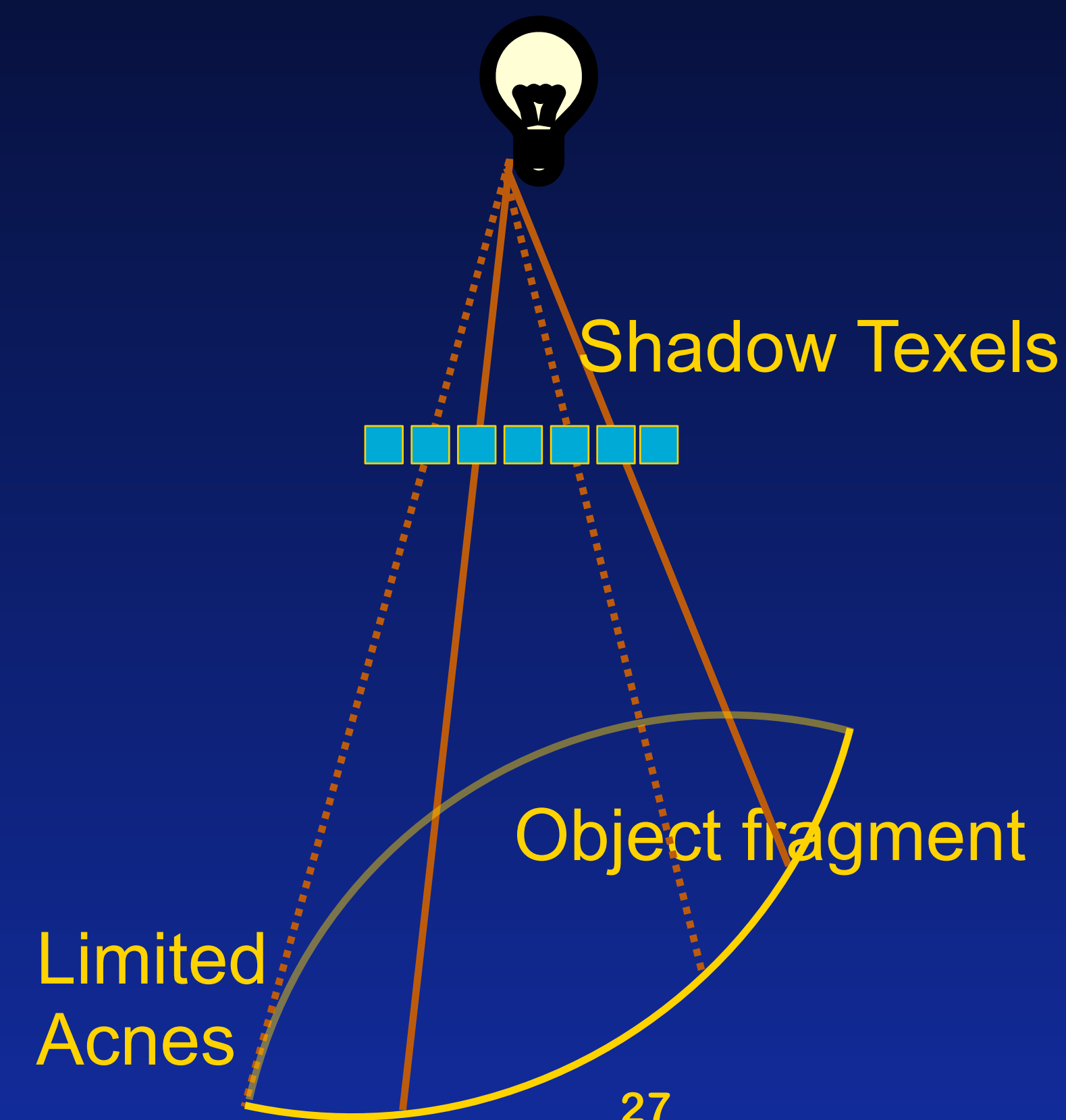
Shadow Acne

float bias = .01;?

float bias = max(0.05 * (1.0 - dot(normal, lightDir)), 0.005); ?

float shadow = texture(shadowMap, vec3(projCoords.xy, projCoords.z - bias));

- What is a good bias value?
 - Increase depth-map resolution by tight near-far range
 - Also see cascaded shadow maps
- Front-face culling can mitigate "self shadowing"
 - but only for closed objects





Shadow Acne

```
float bias = .01;?
```

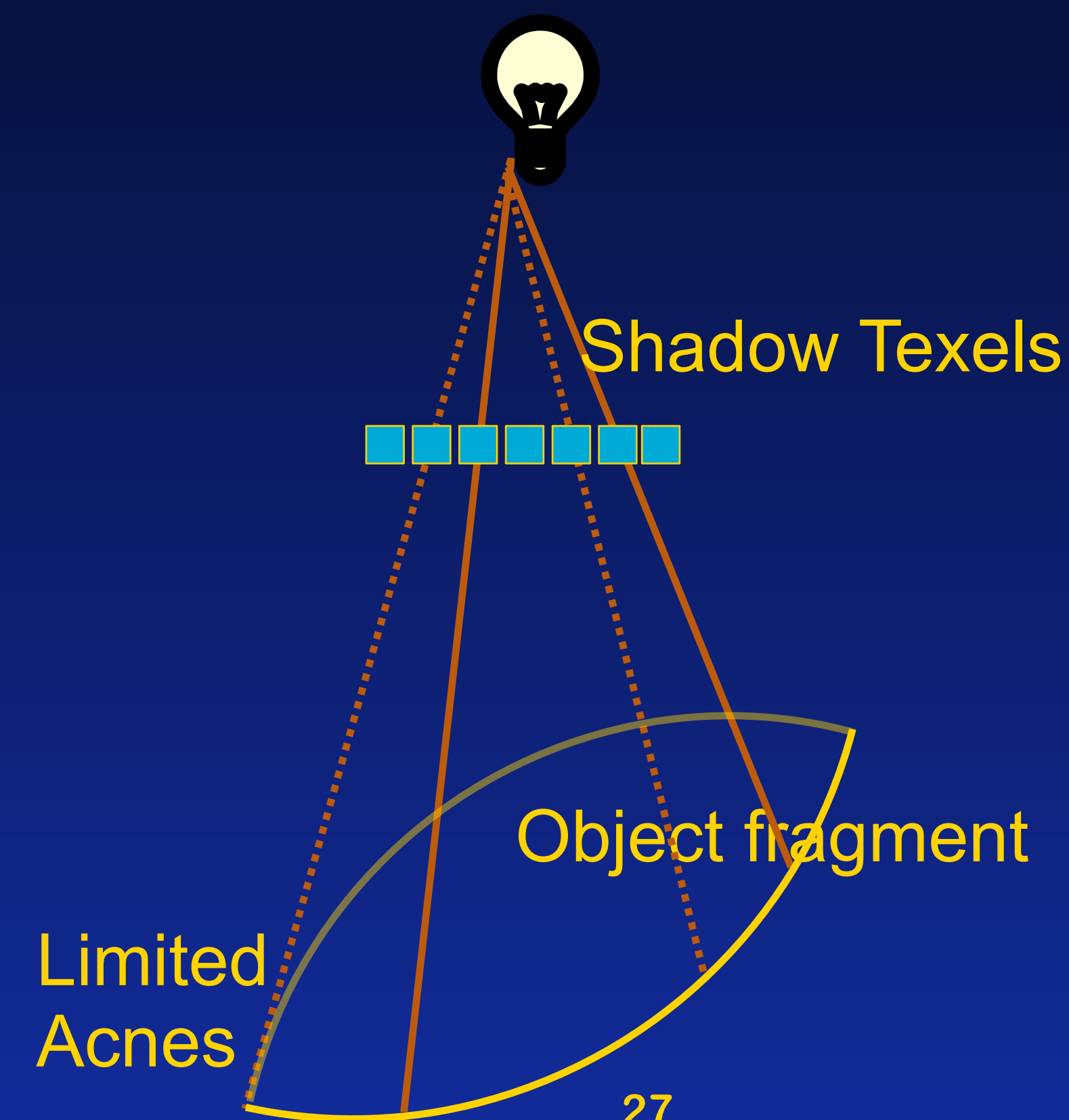
```
float bias = max(0.05 * (1.0 - dot(normal, lightDir)), 0.005); ?
```

```
float shadow = texture(shadowMap, vec3(projCoords.xy, projCoords.z - bias));
```

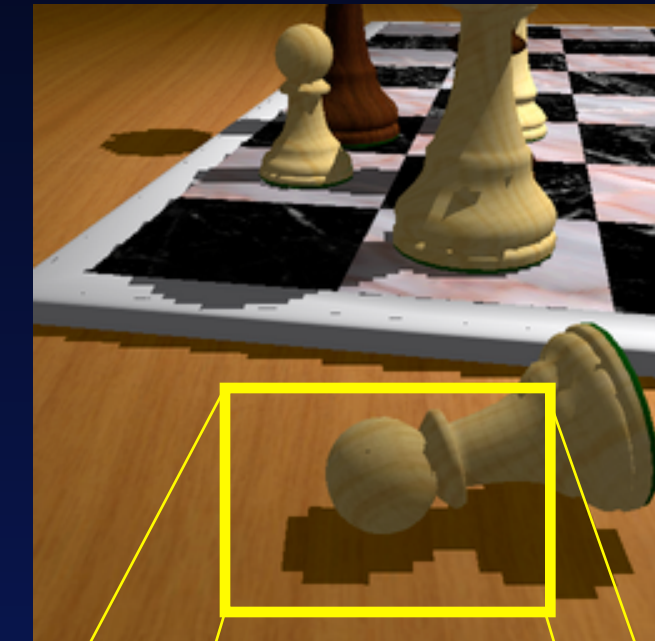
- What is a good bias value?
 - Increase depth-map resolution by tight near-far range
 - Also see cascaded shadow maps
- Front-face culling can mitigate "self shadowing"
 - but only for closed objects

Also see `glEnable(GL_POLYGON_OFFSET_FILL);`
`glPolygonOffset(2.0f, 4.0f);` // Factor and Units

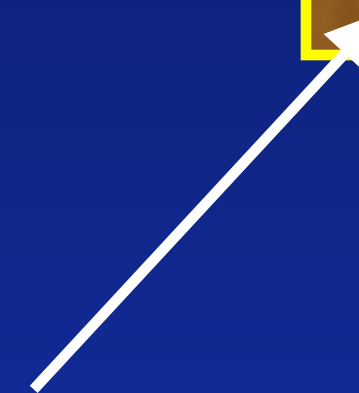
And shadow volumes



Shadow Map Aliasing



Filtering

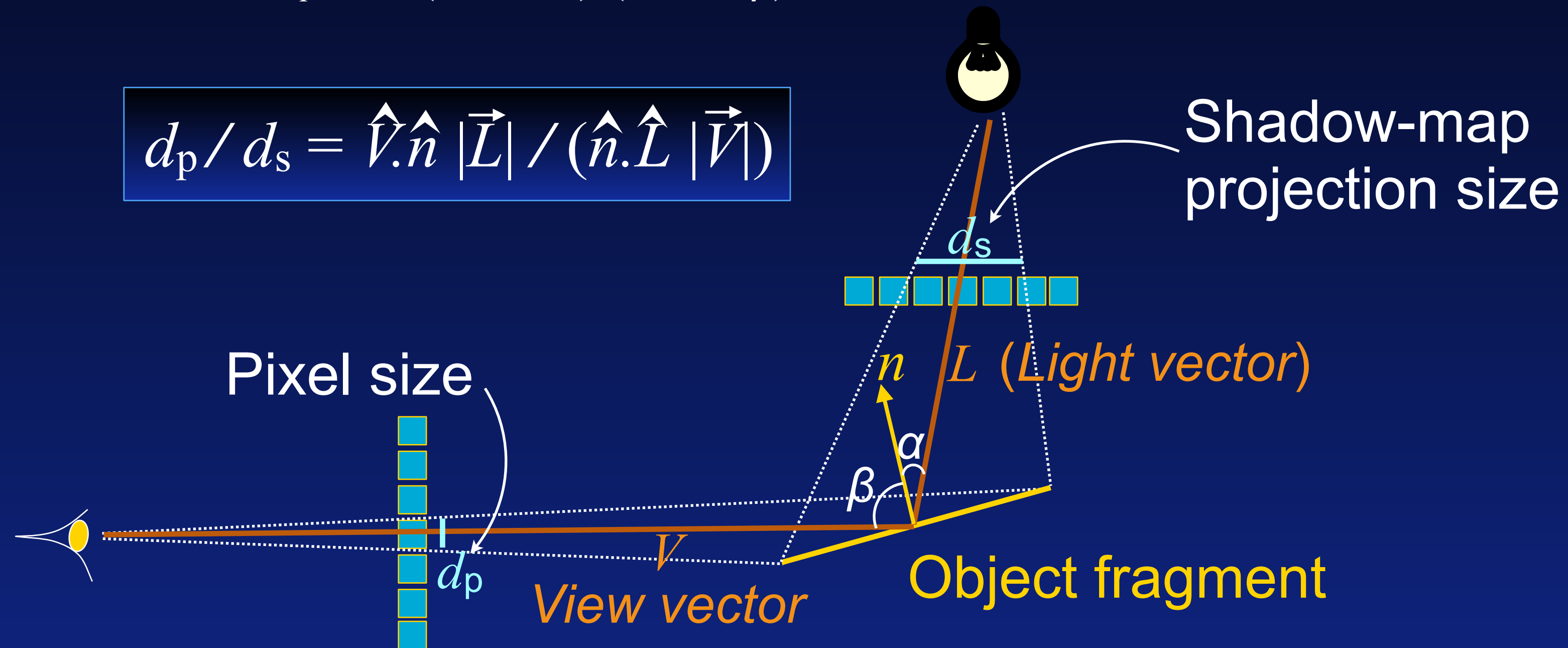


Perspective Scaling in Shadow Map



$$d_p \sim d_s (L/\cos \alpha)/(V/\cos \beta)$$

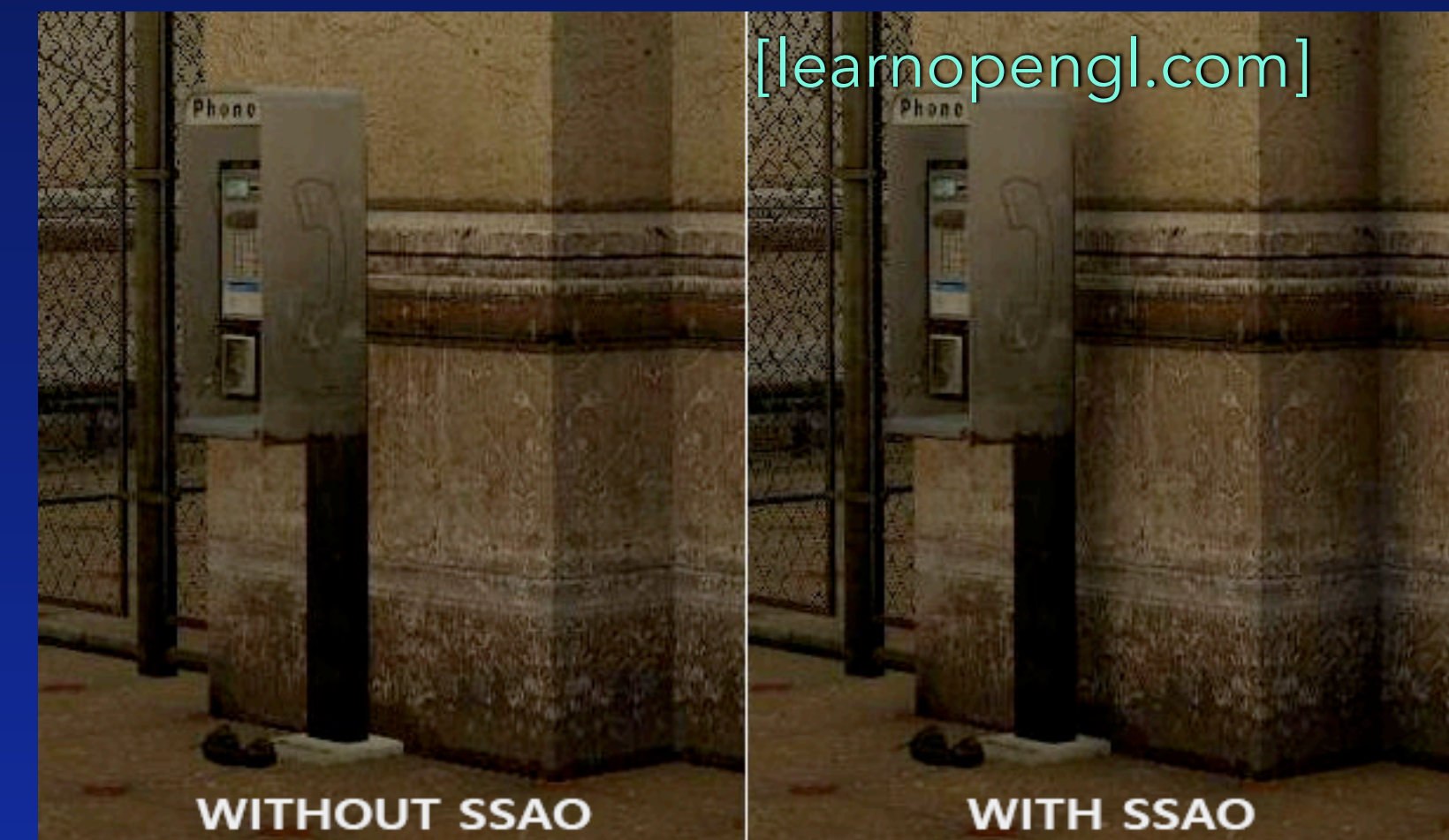
$$d_p / d_s = \hat{V} \cdot \hat{n} |\vec{L}| / (\hat{n} \cdot \hat{L} |\vec{V}|)$$



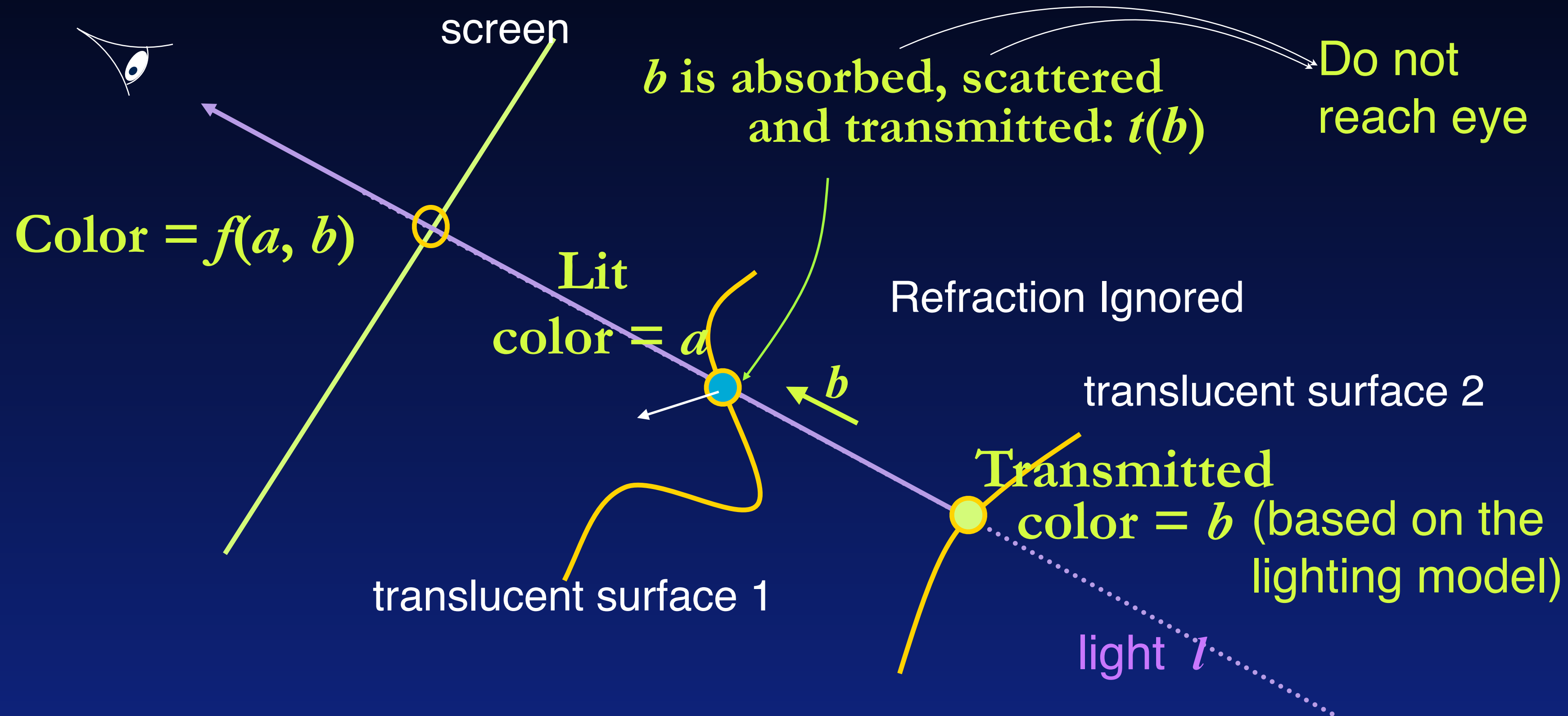
Ambient Occlusion



- Sample rays around normal
 - Rays from Crevices and corners will hit closely objects
- Can be approximated in screen-space
 - Sample depth of neighbors
 - with multi-pass rendering
- Neighbors at similar or deeper depth $\Rightarrow$ low occlusion

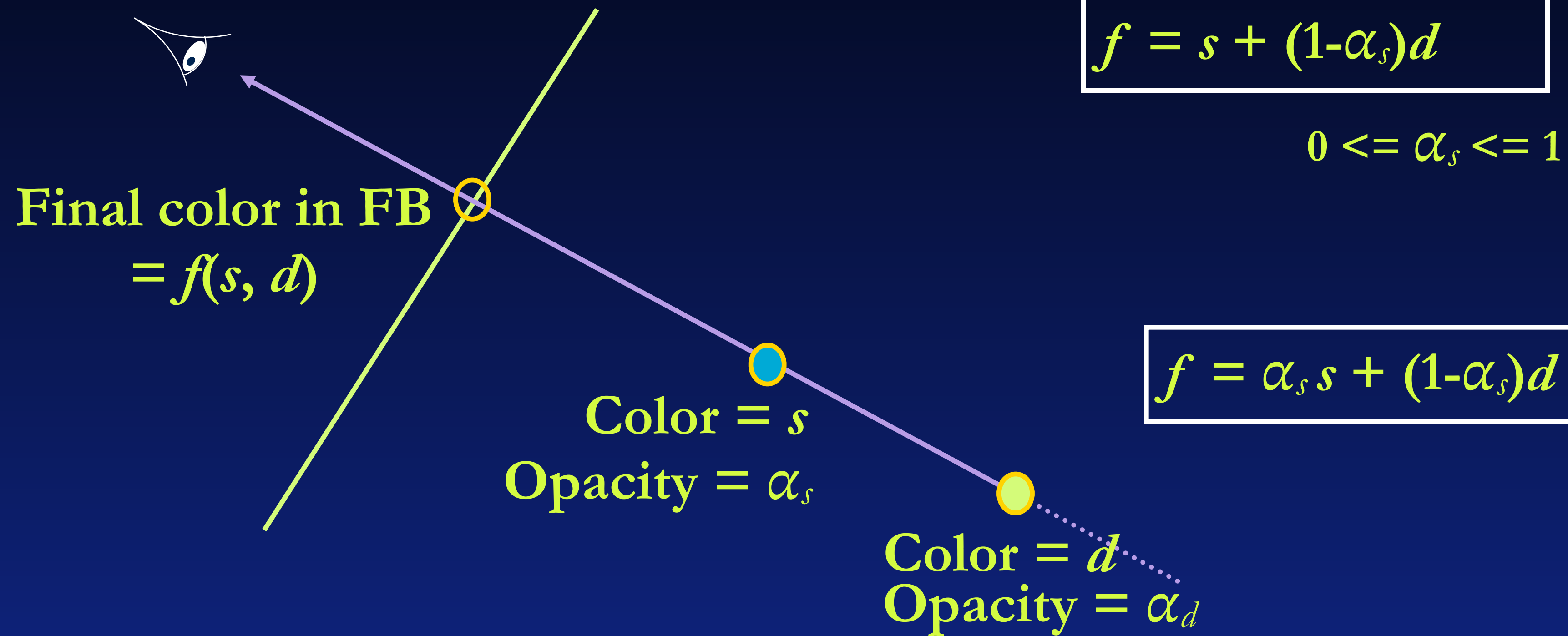


Transparency → Blending

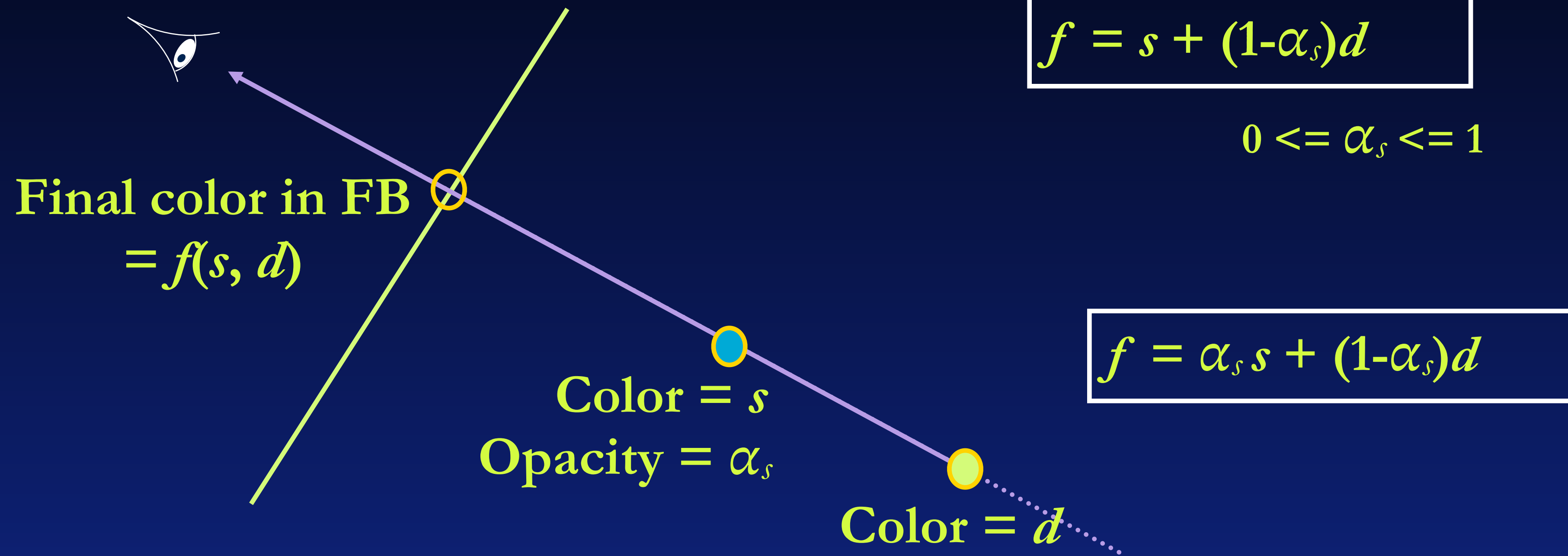


In rasterization pipeline, surfaces 1 and 2 are rendered in an arbitrary order. For opaque surfaces the nearest object (surface 1) overwrites the color b if surface 1 comes later. Or, surface 2 comes later and fails the Z-test, and discards the fragment. To support transparency we do $a + t(b)$?

Alpha Blending



Alpha Blending



Ignoring light attenuation as before



See Functions

- `glEnable(GL_BLEND)`
- `glBlendEquation(GL_ADD)`
- `glBlendFunc(sfactor, dfactor)`
 - `sfactor` and `dfactor` are vectors
 - `Rfactor`, `Gfactor`, `Bfactor`, `Afactor`

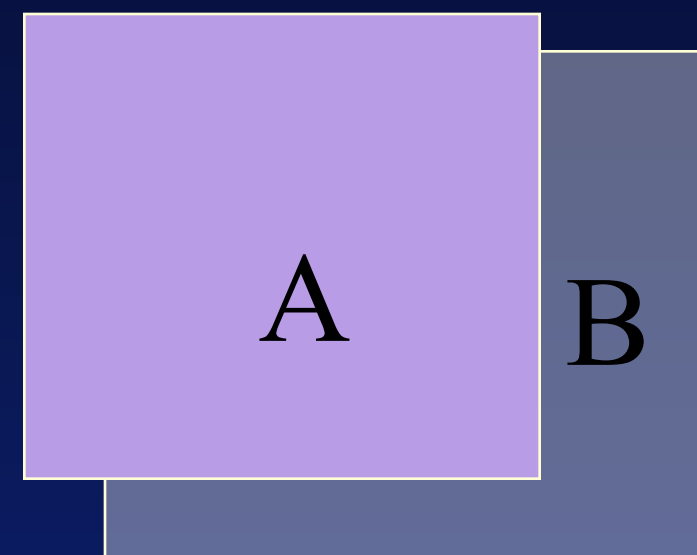
$$dcolor = sfactor * scolor + dfactor * dcolor$$

- Also see: `glBlendColor`



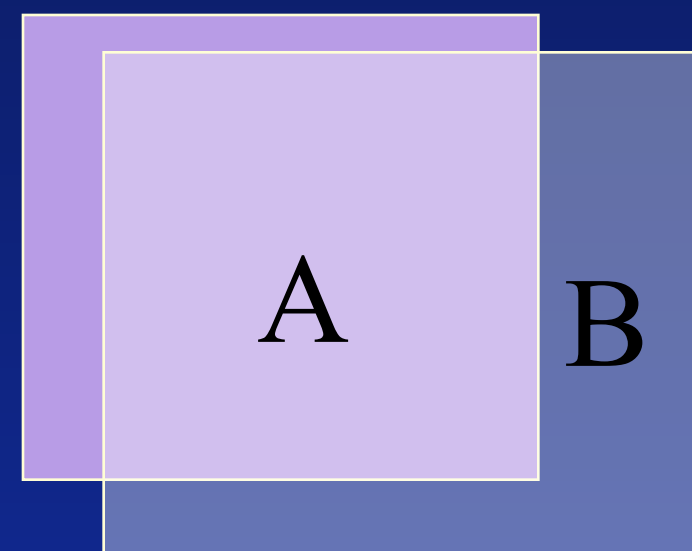
Order Dependent

- Suppose: $d' = \alpha_s s + (1-\alpha_s) d$
- Examples: $\alpha_A = 1$, $\alpha_B = 0.4$



A after B:

$$d' = 1 * C_A + (0) * C_B$$



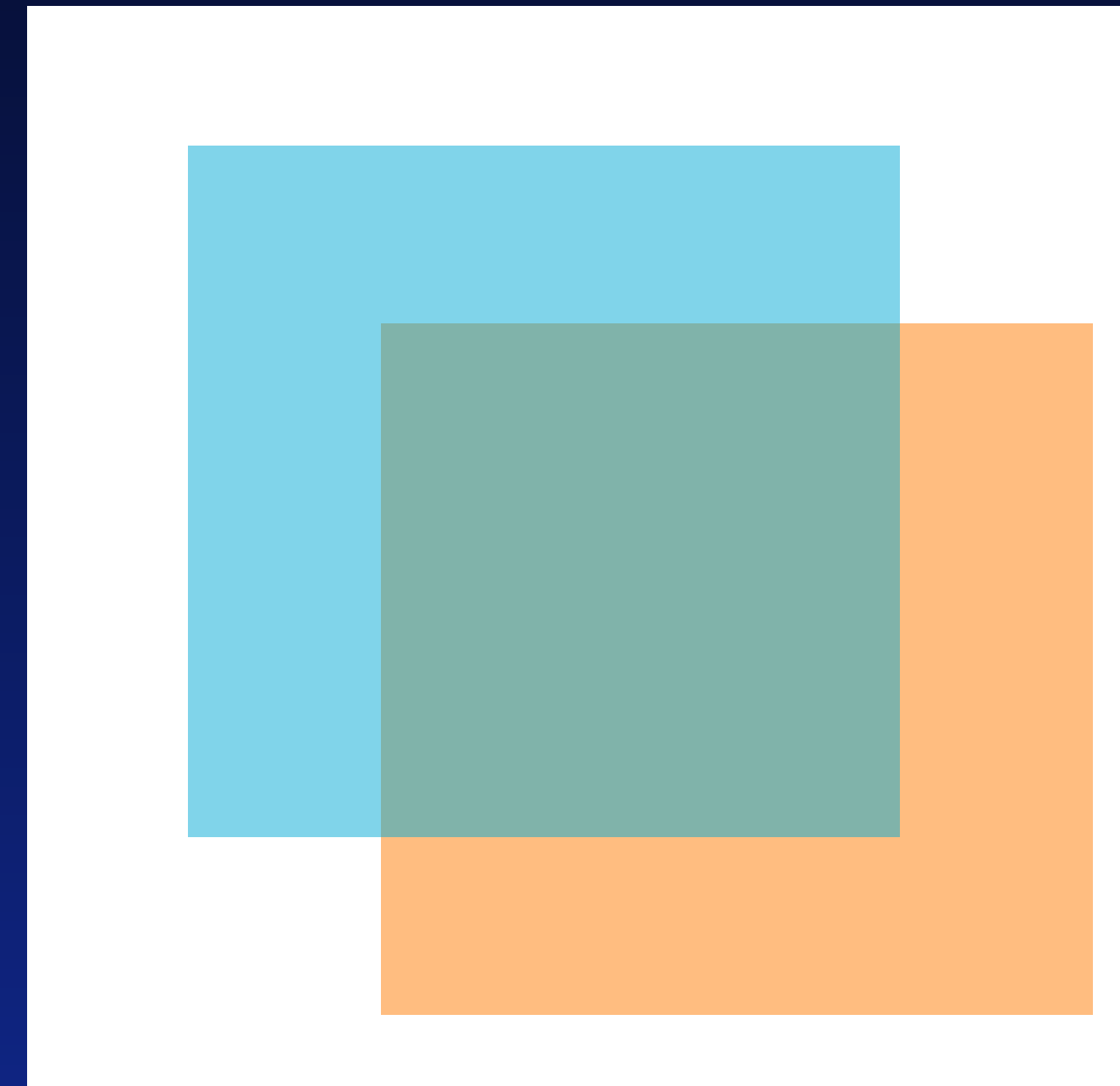
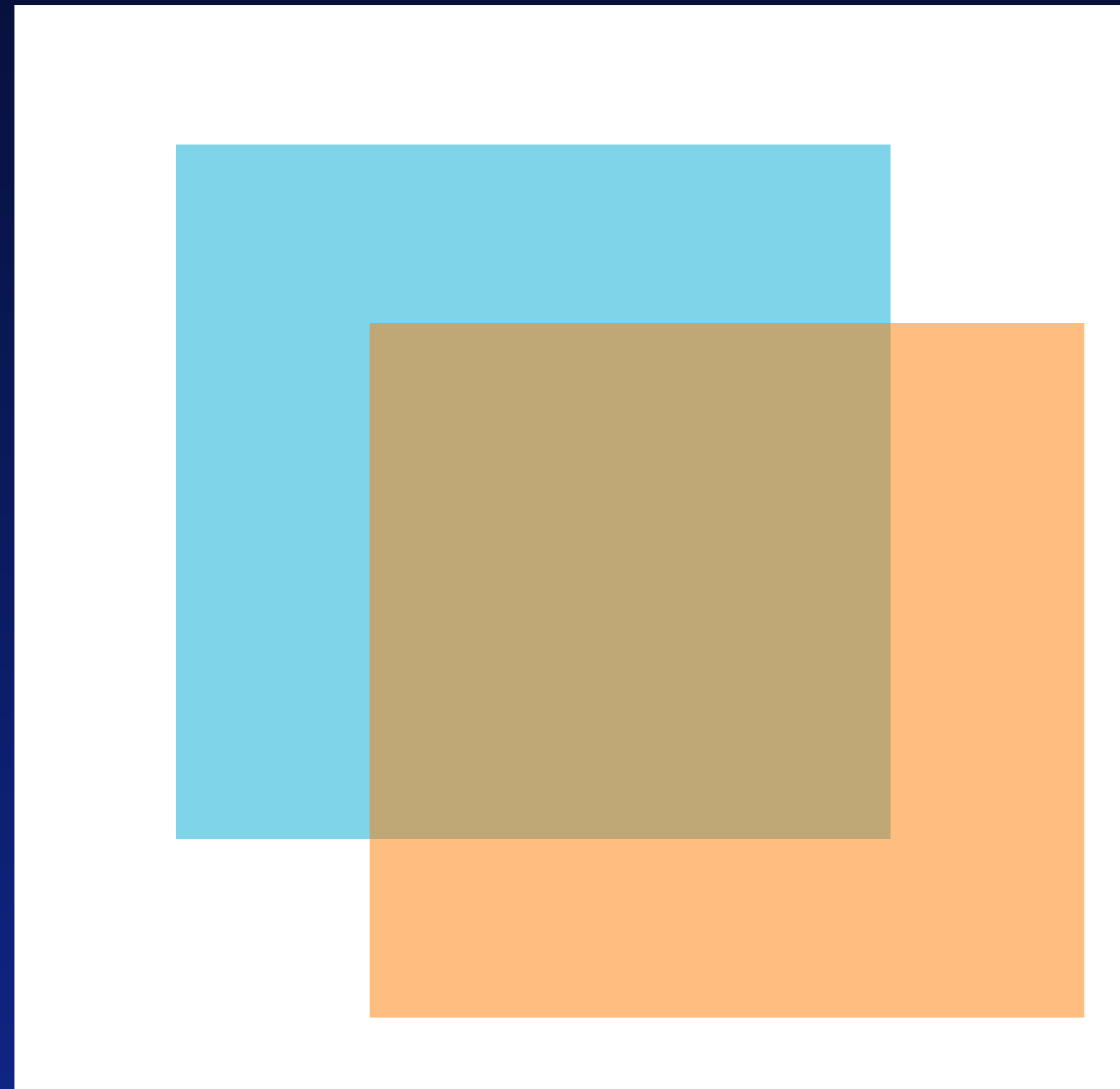
B after A:

$$d' = 0.4 * C_B + (0.6) * C_A$$



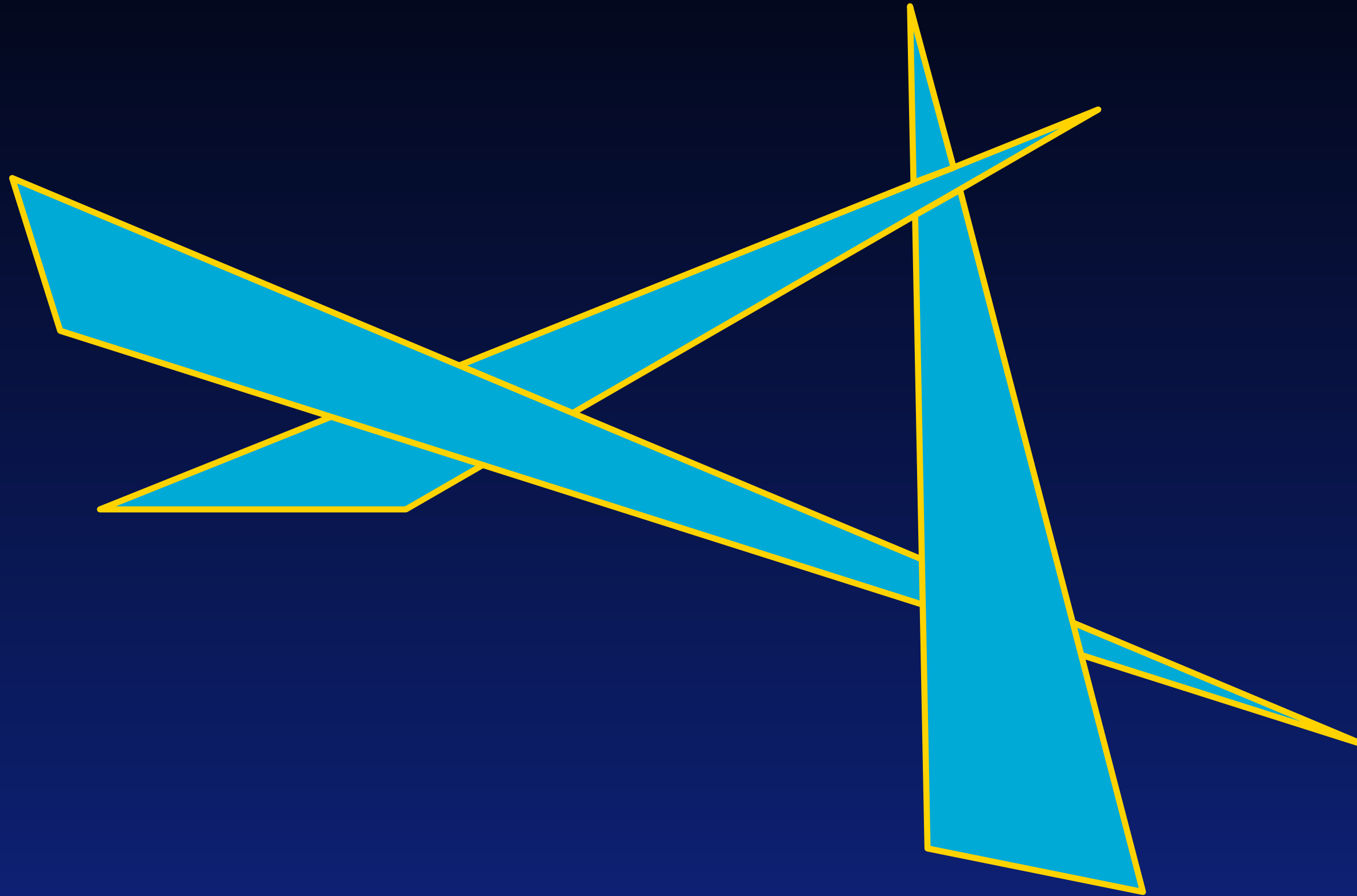
Order Dependent

Even if each quad has an opacity of 0.5

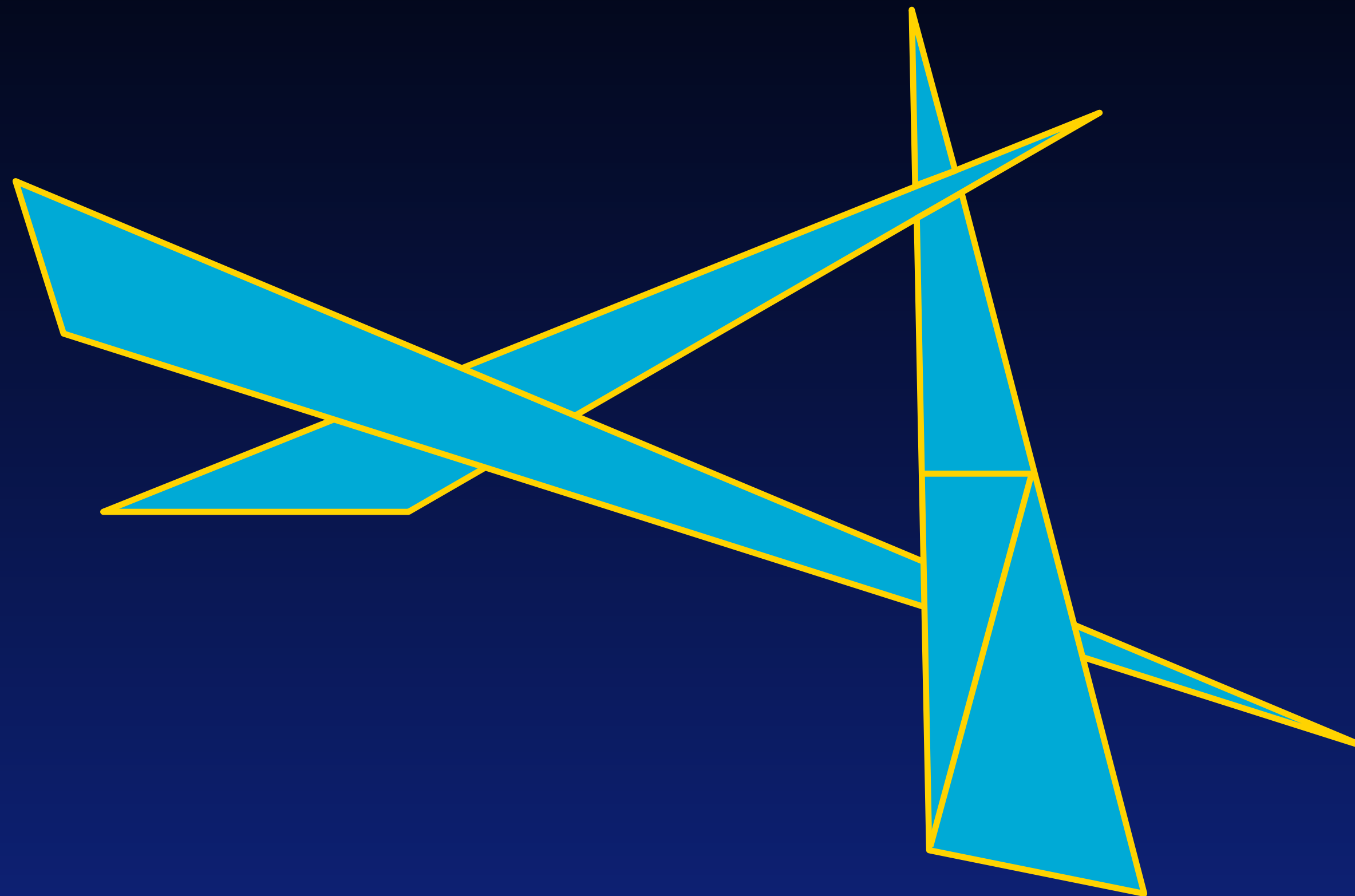


Here we blend the first quad with background

Ordering Triangles?



Ordering Triangles?



Alpha Textures



Multiple Depths Per Pixel



- Instead of over-writing Z, append fragment Z to a multi-depth Z buffer
 - A small number of fragments can be reasonably handled
 - At the end the Z values can be sorted for each pixel
 - and colors combined
- Multiple pass for multiple Z
 - Each pass draws the next nearest at any given pixel
 - Hardware has some support for this



Mammen's Algorithm

- Allocate Buffer 1 and Buffer 2
 - Color (C1, C2) and Depth (Z1, Z2)
- Render opaque primitives into C1,Z1
- Set Z2 depth test to
 - pass for greater Z (farther fragment)
- Repeat
 - Clear C2 and Z2 (to closest Z, usually 0)
 - Rasterize all translucent primitives into C2+Z2
 - Write if Fragment-Z < Z1 (read from texture)
 - Fragment-Z in front of Z1
 - This test is in addition to the regular depth test
 - Blend C1 color to fragment-Color, write to C2
 - Swap (C1, Z1) and (C2, Z2)
- Until no pixel is updated