Lecture 30: Shadow Maps

A Second Pass from the Lamp, the Bias, PCF

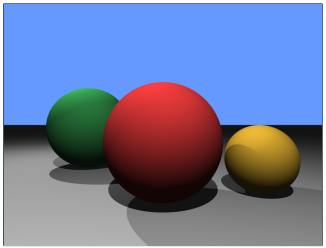
CS 453 — Computer Graphics

Fakhir Shaheen

Forman Christian University

Lecture 17 cast a ray; today we ask a picture

lecture 17, slide 14



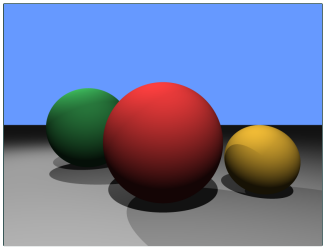
lecture 17



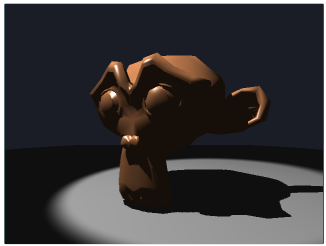
- Lecture 17 asked each point: is the lamp hidden from me?

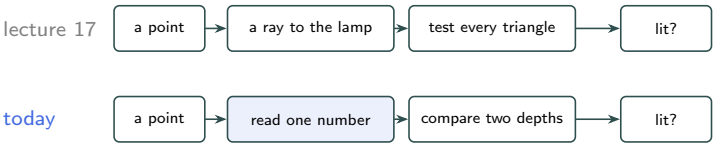
Lecture 17 cast a ray; today we ask a picture

lecture 17, slide 14



today

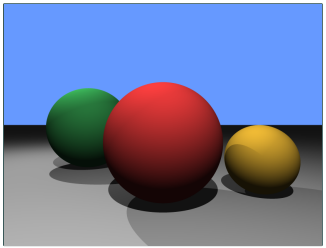




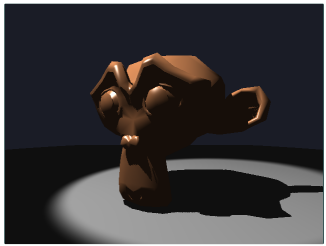
- Lecture 17 asked each point: is the lamp hidden from me?
- Today the lamp draws one picture of what it sees.
- Each point reads one number from it.

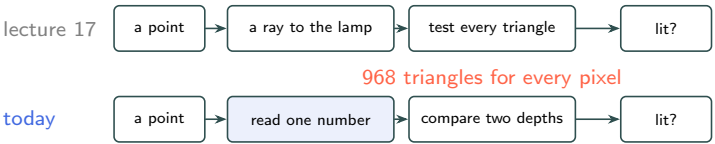
Lecture 17 cast a ray; today we ask a picture

lecture 17, slide 14



today

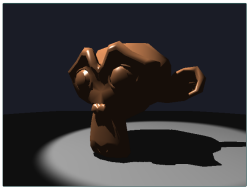




- Lecture 17 asked each point: is the lamp hidden from me?
- Today the lamp draws one picture of what it sees.
- Each point reads one number from it.
- A ray per pixel would test all 968 triangles.

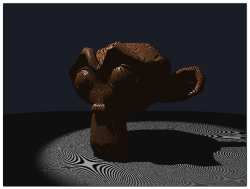
Today: three rounds of slides, then you try

1 **The spot's own depth buffer**



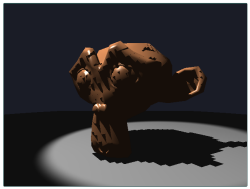
then You Try 1, 2

2 **Acne and the bias**



then You Try 3, 4

3 **Blocky edges and PCF**



then You Try 5

Open the Shadow Lab now. It is on the course calendar, in lecture 30's row.

It is lecture 29's Suzanne Lab, drawn twice a frame. Work in pairs: one screen for two.

A slide with a blue **In the Shadow Lab** box has its code in the lab too.

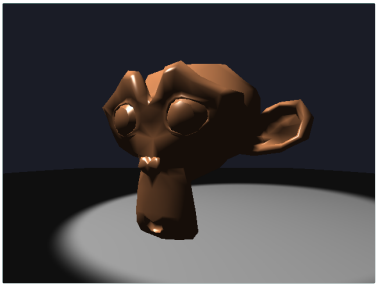
Round 1

The spot's own depth buffer

Draw what the spot sees, then let every point ask it

Lecture 29's question: where is her shadow?

lecture 29, slide 28: no_shadow.wgsl

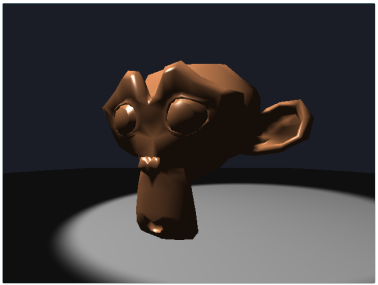


- Lecture 29 ended here. The spot's light goes straight through her.

▶ **In the Shadow Lab:**
Lecture 29's last picture: no shadow

Lecture 29's question: where is her shadow?

lecture 29, slide 28: no_shadow.wgsl



```
let c = base * (ambient + k * diffuse) + k * shine;
```

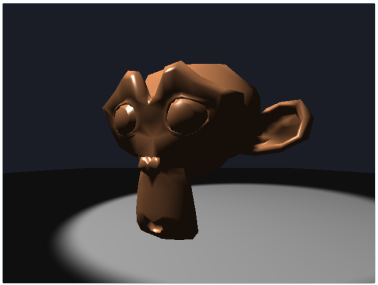
k (the cone, lecture 29, slide 23) and diffuse know only this point.

- Lecture 29 ended here. The spot's light goes straight through her.
- Each point asks only: am I in the cone, and do I face the spot?

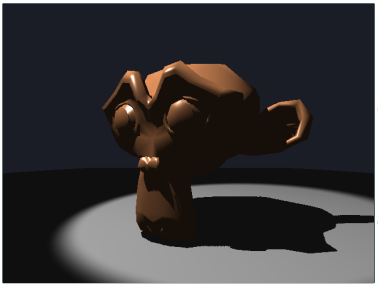
▶ **In the Shadow Lab:**
Lecture 29's last picture: no shadow

Lecture 29's question: where is her shadow?

lecture 29, slide 28: no_shadow.wgsl



today: shadow.wgsl



```
let c = base * (ambient + k * diffuse) + k * shine;
```

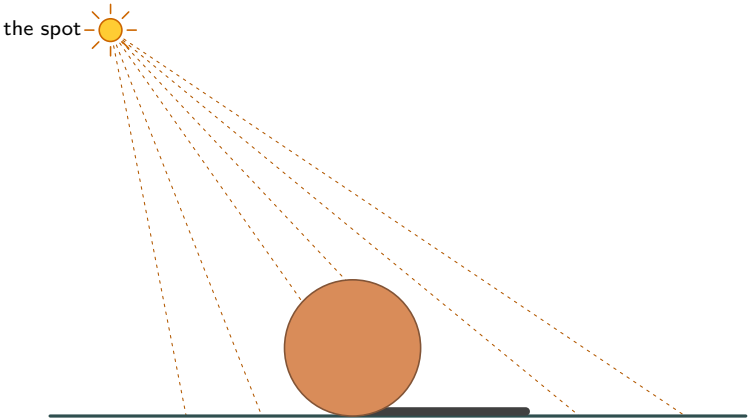
k (the cone, lecture 29, slide 23) and diffuse know only this point.

```
let c = base * (ambient + lit * k * diffuse) + lit * k * shine;
```

- Lecture 29 ended here. The spot's light goes straight through her.
- Each point asks only: am I in the cone, and do I face the spot?
- Today: one more number per point, `lit`: 1 if the spot reaches it, 0 if not.

▶ **In the Shadow Lab:**
Lecture 29's last picture: no shadow

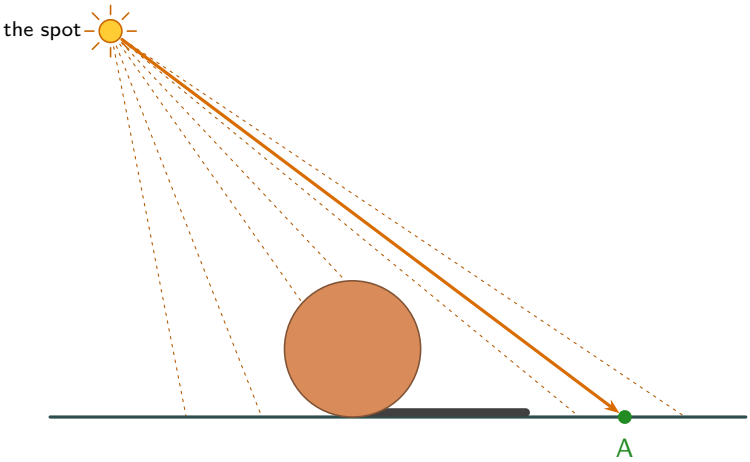
In shadow: the spot meets something else first



Suzanne from the side, drawn simply (lecture 29, slide 31)

- Lecture 17 (slide 7) sent a ray from the point to the lamp.

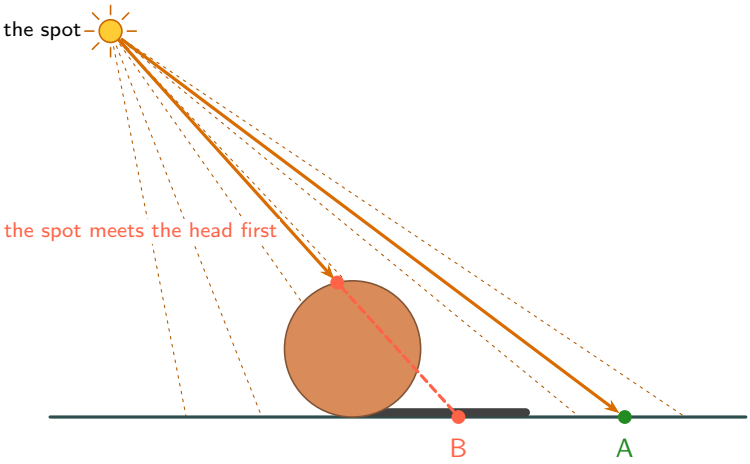
In shadow: the spot meets something else first



Suzanne from the side, drawn simply (lecture 29, slide 31)

- Lecture 17 (slide 7) sent a ray from the point to the lamp.
- A: the spot's light meets A itself first. A is lit.

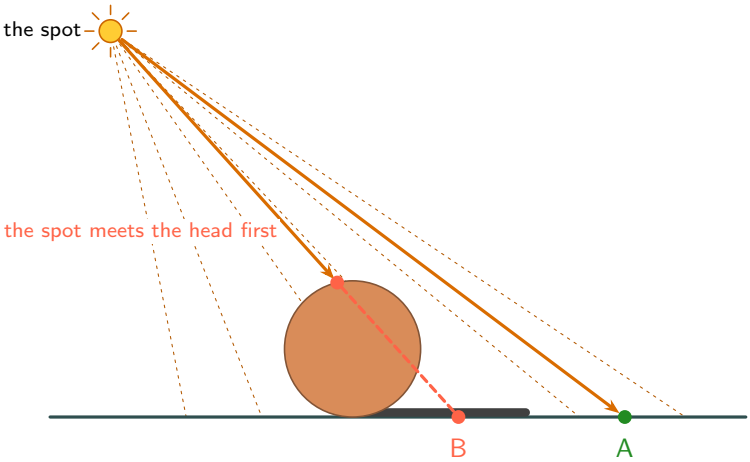
In shadow: the spot meets something else first



Suzanne from the side, drawn simply (lecture 29, slide 31)

- Lecture 17 (slide 7) sent a ray from the point to the lamp.
- A: the spot's light meets A itself first. A is lit.
- B: the light meets the head first. B is in shadow.

In shadow: the spot meets something else first

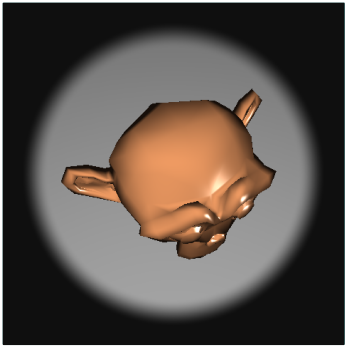


Suzanne from the side, drawn simply (lecture 29, slide 31)

- Lecture 17 (slide 7) sent a ray from the point to the lamp.
- A: the spot's light meets A itself first. A is lit.
- B: the light meets the head first. B is in shadow.
- Turn the question round: what does the spot see first?

Pass 1: the scene from the spot, depth only

the spot as the camera

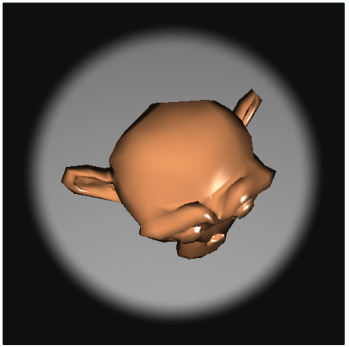


- Draw the scene once with the spot as the camera. The circle: its cone.

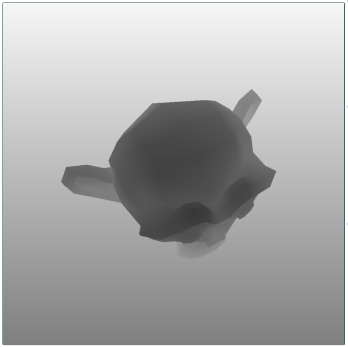
▶ **In the Shadow Lab:**
tick **The spot's view**, then **Show the shadow map**

Pass 1: the scene from the spot, depth only

the spot as the camera



what pass 1 keeps: depths



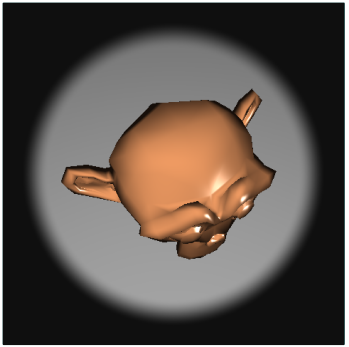
dark: near the spot light: far

- Draw the scene once with the spot as the camera. The circle: its cone.
- Keep only the depth buffer (lecture 28, slide 13). No colours.
- That depth picture is the **shadow map**. One square is a **texel**.

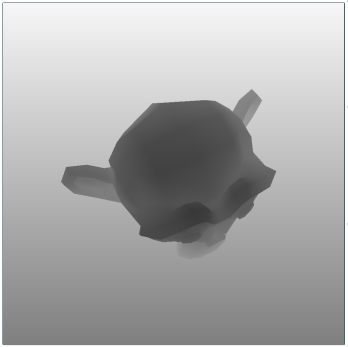
▶ **In the Shadow Lab:**
tick **The spot's view**, then **Show the shadow map**

Pass 1: the scene from the spot, depth only

the spot as the camera



what pass 1 keeps: depths



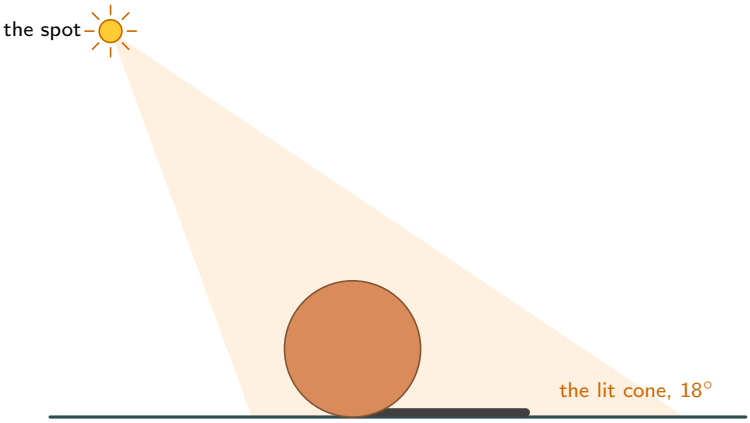
dark: near the spot light: far

- Draw the scene once with the spot as the camera. The circle: its cone.
- Keep only the depth buffer (lecture 28, slide 13). No colours.
- That depth picture is the **shadow map**. One square is a **texel**.

▶ **In the Shadow Lab:**
tick **The spot's view**, then **Show the shadow map**

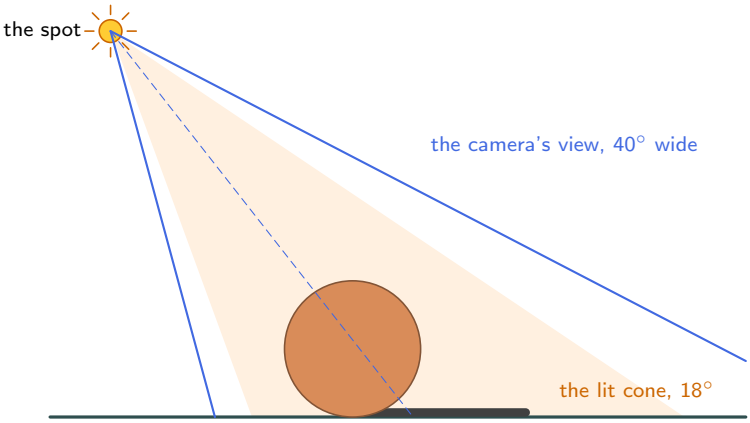
From the spot, every point you can see is lit.
A shadow is what the spot cannot see.

The spot as a camera: its cone is the view



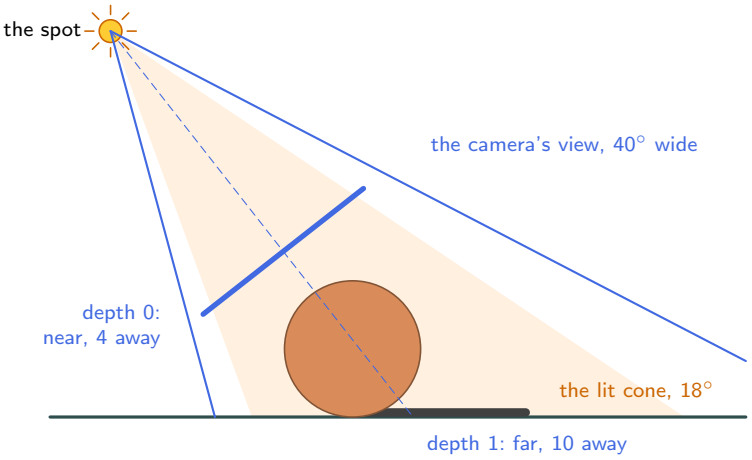
- Pass 1 needs a camera at the spot, looking along its aim.

The spot as a camera: its cone is the view



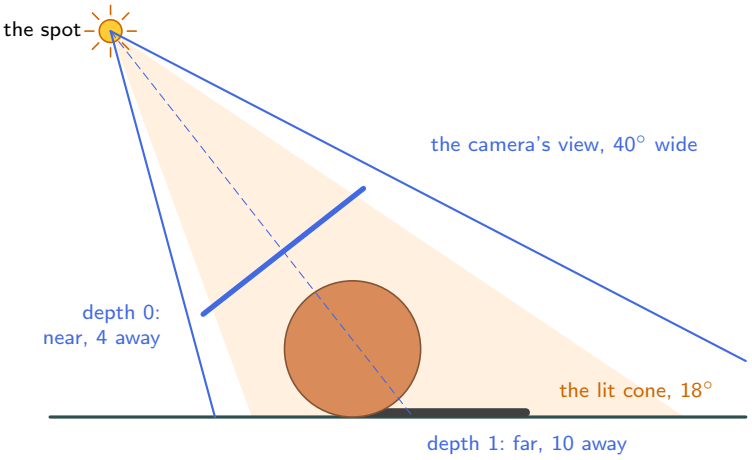
- Pass 1 needs a camera at the spot, looking along its aim.
- Lecture 28's camera. V: move the spot to the middle, turn it to look down $-z$.

The spot as a camera: its cone is the view



- Pass 1 needs a camera at the spot, looking along its aim.
- Lecture 28's camera. V: move the spot to the middle, turn it to look down $-z$.
- P: lecture 28's perspective, 40° wide, so the 18° cone fits.

The spot as a camera: its cone is the view

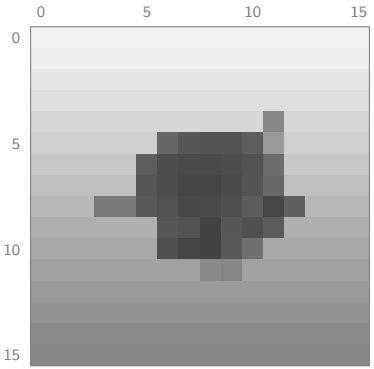


- Pass 1 needs a camera at the spot, looking along its aim.
- Lecture 28's camera. V: move the spot to the middle, turn it to look down $-z$.
- P: lecture 28's perspective, 40° wide, so the 18° cone fits.
- `u.lamp_mvp` = P V M, made by shadows.py.

```
return u.lamp_mvp * vec4f(xyz.x, xyz.y, xyz.z, 1.0);  // lamp_vs: all of pass 1's shader
```

The shadow map is a grid of numbers

a tiny 16×16 shadow map

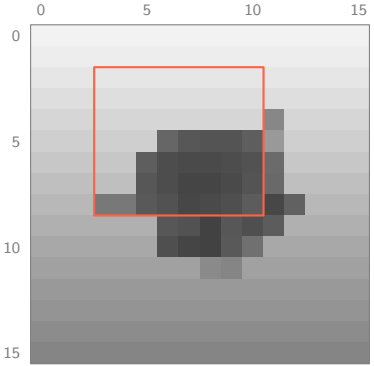


col $\rightarrow$, row $\downarrow$, counted from 0

- Each texel holds one depth: the nearest thing the spot saw there.

The shadow map is a grid of numbers

a tiny 16 × 16 shadow map



	3	4	5	6	7	8	9	10
2	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86
3	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83
4	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81
5	0.78	0.78	0.78	0.41	0.36	0.35	0.35	0.39
6	0.75	0.75	0.39	0.33	0.32	0.32	0.33	0.34
7	0.73	0.73	0.36	0.32	0.30	0.30	0.31	0.35
8	0.48	0.48	0.37	0.34	0.31	0.32	0.33	0.37

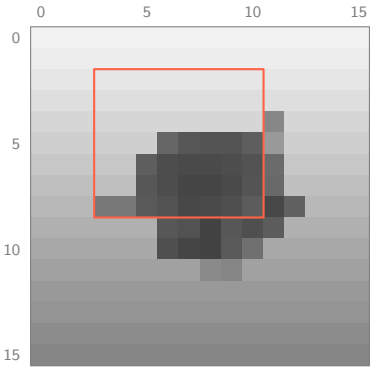
columns 3 to 10, rows 2 to 8

col →, row ↓, counted from 0

- Each texel holds one depth: the nearest thing the spot saw there.
- Suzanne is near the spot: about 0.3. The floor round her: 0.48 to 0.86.

The shadow map is a grid of numbers

a tiny 16 × 16 shadow map



col →, row ↓, counted from 0

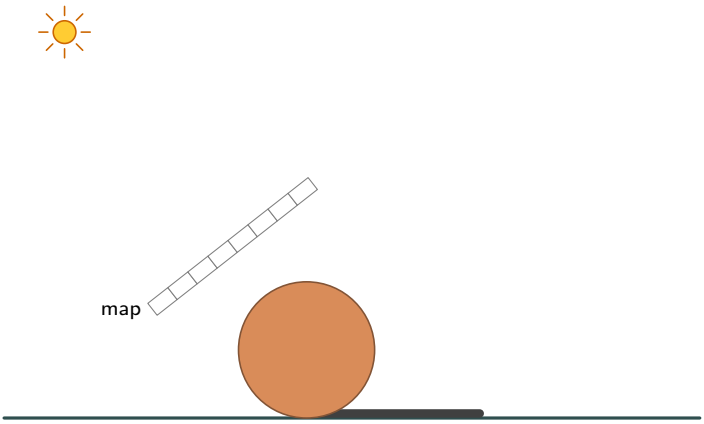
	3	4	5	6	7	8	9	10
2	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86
3	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83
4	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81
5	0.78	0.78	0.78	0.41	0.36	0.35	0.35	0.39
6	0.75	0.75	0.39	0.33	0.32	0.32	0.33	0.34
7	0.73	0.73	0.36	0.32	0.30	0.30	0.31	0.35
8	0.48	0.48	0.37	0.34	0.31	0.32	0.33	0.37

columns 3 to 10, rows 2 to 8

- Each texel holds one depth: the nearest thing the spot saw there.
- Suzanne is near the spot: about 0.3. The floor round her: 0.48 to 0.86.
- The floor's depth grows up the map: that part is farther from the spot.

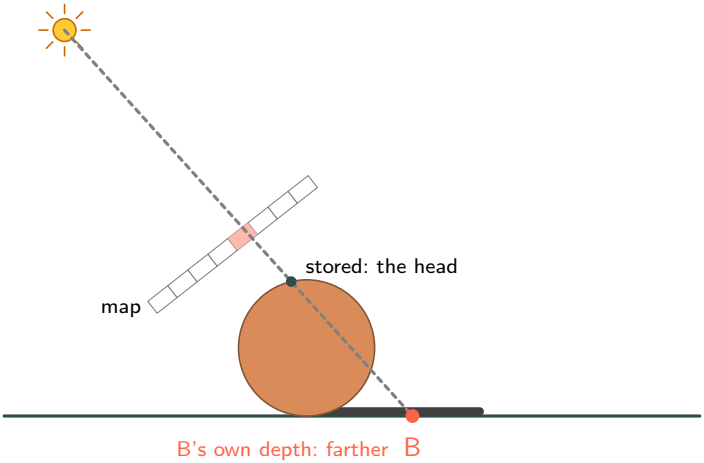
Real numbers from the GPU. The lab's map has 1024 × 1024.

Pass 2: each point looks itself up



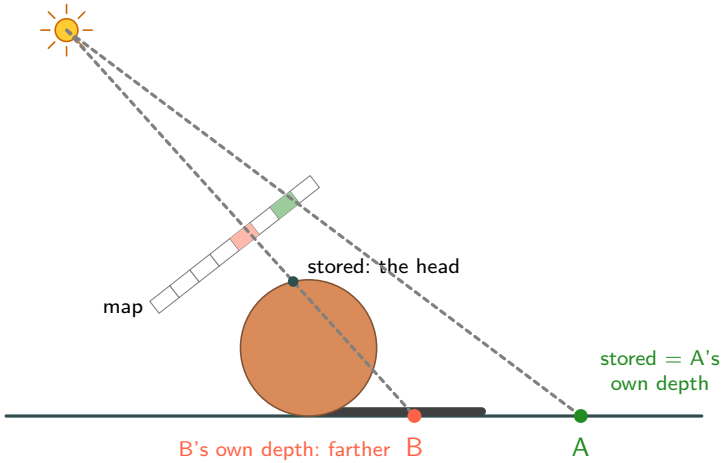
- Pass 2 draws from the eye. Each point finds its texel in the map.

Pass 2: each point looks itself up



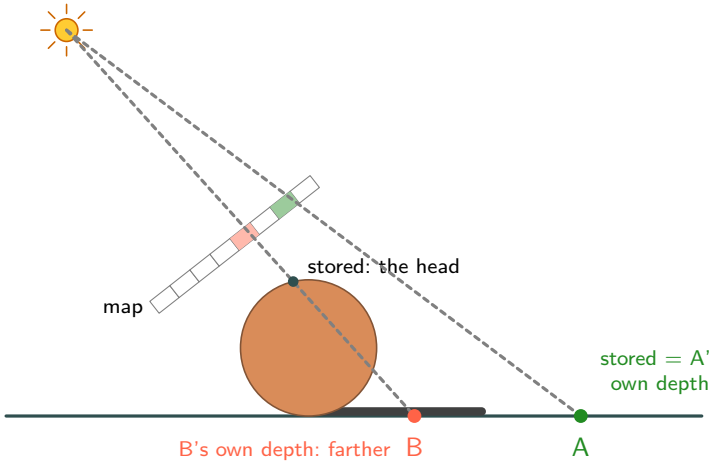
- Pass 2 draws from the eye. Each point finds its texel in the map.
- B: its own depth is bigger than the stored one. B is in shadow.

Pass 2: each point looks itself up



- Pass 2 draws from the eye. Each point finds its texel in the map.
- B: its own depth is bigger than the stored one. B is in shadow.
- A: the two depths are equal. A is lit.

Pass 2: each point looks itself up



- Pass 2 draws from the eye. Each point finds its texel in the map.
- B: its own depth is bigger than the stored one. B is in shadow.
- A: the two depths are equal. A is lit.
- vs_main also hands on each corner as the spot sees it: `lamp`.

```
out.lamp = u.lamp_mvp * vec4f(xyz.x, xyz.y, xyz.z, 1.0);  // new in vs_main
```

Worked: is P in shadow? Is Q?

Two floor points, the 16 × 16 map:

$P = (1, -1, -1)$: lamp $= (-0.381,\ 1.633,\ 5.595,\ 7.357)$

divided by w (lecture 28, slide 22): $(-0.052,\ 0.222,\ 0.761)$

$Q = (1.5, -1, -1.5)$: divided by w: $(-0.099,\ 0.423,\ 0.802)$

across $= (x + 1)/2$, down $= (1 - y)/2$, my_depth $= z$

col = across × 16, row = down × 16

(whole part only; lecture 25, slide 26: shift, halve, flip y)

Try first: find each texel, then compare.

	3	4	5	6	7	8	9	10
2	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86
3	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83
4	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81
5	0.78	0.78	0.78	0.41	0.36	0.35	0.35	0.39
6	0.75	0.75	0.39	0.33	0.32	0.32	0.33	0.34
7	0.73	0.73	0.36	0.32	0.30	0.30	0.31	0.35
8	0.48	0.48	0.37	0.34	0.31	0.32	0.33	0.37

Worked: is P in shadow? Is Q?

Two floor points, the 16 × 16 map:

$P = (1, -1, -1)$: lamp $= (-0.381,\ 1.633,\ 5.595,\ 7.357)$

divided by w (lecture 28, slide 22): $(-0.052,\ 0.222,\ 0.761)$

$Q = (1.5, -1, -1.5)$: divided by w: $(-0.099,\ 0.423,\ 0.802)$

across $= (x + 1)/2$, down $= (1 - y)/2$, my_depth $= z$

col $=$ across $\times$ 16, row $=$ down $\times$ 16

(whole part only; lecture 25, slide 26: shift, halve, flip y)

Try first: find each texel, then compare.

	3	4	5	6	7	8	9	10
2	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86
3	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83
4	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81
5	0.78	0.78	0.78	0.41	0.36	0.35	0.35	0.39
6	0.75	0.75	0.39	0.33	0.32	0.32	0.33	0.34
7	0.73	0.73	0.36	0.32	0.30	0.30	0.31	0.35
8	0.48	0.48	0.37	0.34	0.31	0.32	0.33	0.37

P: across 0.474, down 0.389 → col 7, row 6. Own 0.76 > stored 0.32: **shadow**

Worked: is P in shadow? Is Q?

Two floor points, the 16 × 16 map:

$P = (1, -1, -1)$: lamp $= (-0.381,\ 1.633,\ 5.595,\ 7.357)$

divided by w (lecture 28, slide 22): $(-0.052,\ 0.222,\ 0.761)$

$Q = (1.5, -1, -1.5)$: divided by w: $(-0.099,\ 0.423,\ 0.802)$

across $= (x + 1)/2$, down $= (1 - y)/2$, my_depth $= z$

col = across × 16, row = down × 16

(whole part only; lecture 25, slide 26: shift, halve, flip y)

Try first: find each texel, then compare.

	3	4	5	6	7	8	9	10
2	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86
3	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83
4	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81
5	0.78	0.78	0.78	0.41	0.36	0.35	0.35	0.39
6	0.75	0.75	0.39	0.33	0.32	0.32	0.33	0.34
7	0.73	0.73	0.36	0.32	0.30	0.30	0.31	0.35
8	0.48	0.48	0.37	0.34	0.31	0.32	0.33	0.37

P: across 0.474, down 0.389 → col 7, row 6. Own 0.76 > stored 0.32: **shadow**

Q: across 0.451, down 0.288 → col 7, row 4. Own 0.80 ≤ stored 0.81: **lit**

Worked: is P in shadow? Is Q?

Two floor points, the 16 × 16 map:

$P = (1, -1, -1)$: lamp = $(-0.381,\ 1.633,\ 5.595,\ 7.357)$

divided by w (lecture 28, slide 22): $(-0.052,\ 0.222,\ 0.761)$

$Q = (1.5, -1, -1.5)$: divided by w: $(-0.099,\ 0.423,\ 0.802)$

across $= (x + 1)/2$, down $= (1 - y)/2$, my_depth $= z$

col = across × 16, row = down × 16

(whole part only; lecture 25, slide 26: shift, halve, flip y)

Try first: find each texel, then compare.

	3	4	5	6	7	8	9	10
2	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86
3	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83
4	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81
5	0.78	0.78	0.78	0.41	0.36	0.35	0.35	0.39
6	0.75	0.75	0.39	0.33	0.32	0.32	0.33	0.34
7	0.73	0.73	0.36	0.32	0.30	0.30	0.31	0.35
8	0.48	0.48	0.37	0.34	0.31	0.32	0.33	0.37

P: across 0.474, down 0.389 → col 7, row 6. Own 0.76 > stored 0.32: **shadow**

Q: across 0.451, down 0.288 → col 7, row 4. Own 0.80 ≤ stored 0.81: **lit**

P's texel holds Suzanne (0.32): she is between P and the spot.

The shadow test, in WGSL

```
// fs_main(P, normal, lamp): lecture 29's base, k, diffuse, shine ...
// Step 1: where does the point land in the spot's picture?
let across = (lamp.x / lamp.w + 1.0) / 2.0;
let down = (1.0 - lamp.y / lamp.w) / 2.0;
let my_depth = lamp.z / lamp.w;
```

- The GPU divides by w only for the screen. Here we divide ourselves.

▶ **In the Shadow Lab:**
The shadow test: one texel

The shadow test, in WGSL

```
// fs_main(P, normal, lamp): lecture 29's base, k, diffuse, shine ...
// Step 1: where does the point land in the spot's picture?
let across = (lamp.x / lamp.w + 1.0) / 2.0;
let down = (1.0 - lamp.y / lamp.w) / 2.0;
let my_depth = lamp.z / lamp.w;
// Step 2: read the depth the spot kept in that texel.
let size = i32(textureDimensions(shadow_map).x);
let col = clamp(i32(across * f32(size)), 0, size - 1);
let row = clamp(i32(down * f32(size)), 0, size - 1);
let stored = textureLoad(shadow_map, vec2i(col, row), 0);
```

- The GPU divides by w only for the screen. Here we divide ourselves.
- `shadow_map`: pass 1's depths, binding 1 (given, like `u`).

textureLoad: built into WGSL reads one texel: its column and row

▶ **In the Shadow Lab:**
The shadow test: one texel

The shadow test, in WGSL

```
// fs_main(P, normal, lamp): lecture 29's base, k, diffuse, shine ...
// Step 1: where does the point land in the spot's picture?
let across = (lamp.x / lamp.w + 1.0) / 2.0;
let down = (1.0 - lamp.y / lamp.w) / 2.0;
let my_depth = lamp.z / lamp.w;
// Step 2: read the depth the spot kept in that texel.
let size = i32(textureDimensions(shadow_map).x);
let col = clamp(i32(across * f32(size)), 0, size - 1);
let row = clamp(i32(down * f32(size)), 0, size - 1);
let stored = textureLoad(shadow_map, vec2i(col, row), 0);
// Step 3: compare (slide 10's rule).
let bias = 0.01;
var lit = 1.0;
if (my_depth - bias > stored) { lit = 0.0; }
```

- The GPU divides by w only for the screen. Here we divide ourselves.
- `shadow_map`: pass 1's depths, binding 1 (given, like `u`).

textureLoad: built into WGSL reads one texel: its column and row

`bias` in grey: round 2. clamp keeps col and row inside the map.

▶ **In the Shadow Lab:**
The shadow test: one texel

The shadow test, in WGSL

```
// fs_main(P, normal, lamp): lecture 29's base, k, diffuse, shine ...
// Step 1: where does the point land in the spot's picture?
let across = (lamp.x / lamp.w + 1.0) / 2.0;
let down = (1.0 - lamp.y / lamp.w) / 2.0;
let my_depth = lamp.z / lamp.w;
// Step 2: read the depth the spot kept in that texel.
let size = i32(textureDimensions(shadow_map).x);
let col = clamp(i32(across * f32(size)), 0, size - 1);
let row = clamp(i32(down * f32(size)), 0, size - 1);
let stored = textureLoad(shadow_map, vec2i(col, row), 0);
// Step 3: compare (slide 10's rule).
let bias = 0.01;
var lit = 1.0;
if (my_depth - bias > stored) { lit = 0.0; }
let c = base * (ambient + lit * k * diffuse) + lit * k * shine;
```

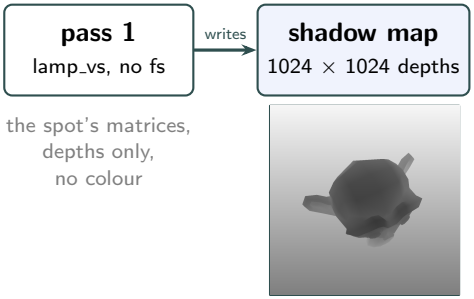
- The GPU divides by w only for the screen. Here we divide ourselves.
- `shadow_map`: pass 1's depths, binding 1 (given, like `u`).
- `lit` $= 0$ puts out the spot's light. The ambient 0.1 stays.

textureLoad: built into WGSL reads one texel: its column and row

bias in grey: round 2. clamp keeps col and row inside the map.

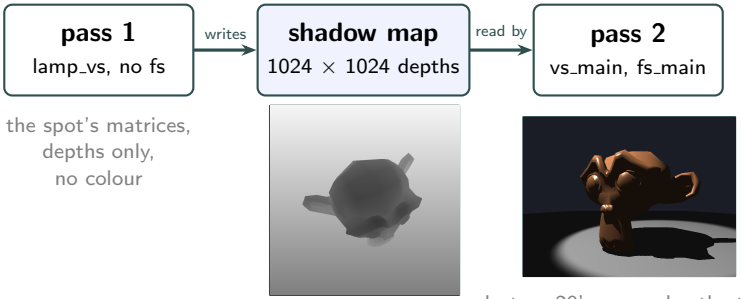
▶ **In the Shadow Lab:**
The shadow test: one texel

Every frame: two passes, one texture between them



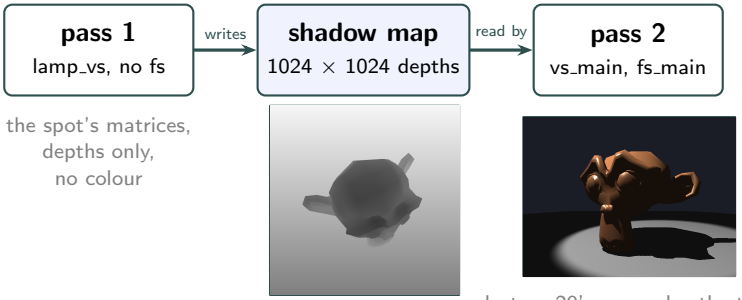
- Pass 1 draws from the spot into the shadow map.

Every frame: two passes, one texture between them



- Pass 1 draws from the spot into the shadow map.
- Pass 2 draws from the eye, reading the map.

Every frame: two passes, one texture between them



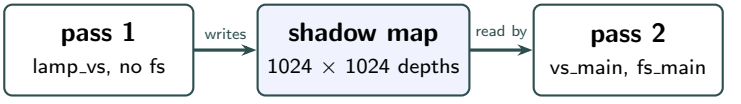
- Pass 1 draws from the spot into the shadow map.
- Pass 2 draws from the eye, reading the map.

In shadows.py: one depth texture, made once, used two ways

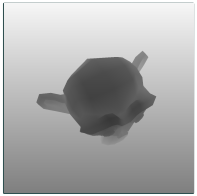
`usage = RENDER_ATTACHMENT | TEXTURE_BINDING`

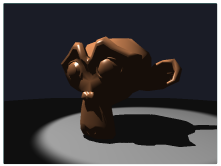
pass 1 draws into it; pass 2 binds it as `shadow_map`

Every frame: two passes, one texture between them



the spot's matrices, depths only, no colour





lecture 29's pass, plus the test

- Pass 1 draws from the spot into the shadow map.
- Pass 2 draws from the eye, reading the map.
- Both run every frame: the spot or Suzanne may move.
- The cost: one extra drawing of the scene, with no colour.

In shadows.py: one depth texture, made once, used two ways

`usage = RENDER_ATTACHMENT | TEXTURE_BINDING`

pass 1 draws into it; pass 2 binds it as `shadow_map`

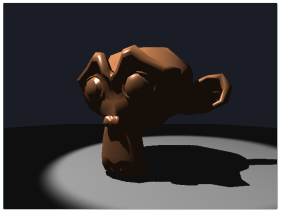
You Try 1: paint each point by my_depth

In the Shadow Lab, choose **You Try 1**. Guess first, then change the code.

```
let my_depth = lamp.z / lamp.w;
...
var c = base * (ambient + lit * k * diffuse) + lit * k * shine;
// your line here: c = ...
return vec4f(c.r, c.g, c.b, 1.0);
```

add a line here

now



- **Paint each point grey by its own depth, seen from the spot.**
- Guess first: her head or the floor, which is darker?

You Try 1: paint each point by my_depth

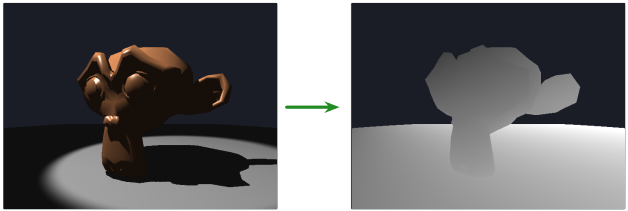
In the Shadow Lab, choose **You Try 1**. Guess first, then change the code.

```
let my_depth = lamp.z / lamp.w;
...
var c = base * (ambient + lit * k * diffuse) + lit * k * shine;
// your line here: c = ...
return vec4f(c.r, c.g, c.b, 1.0);
```

add a line here

answer: `c = vec3f(my_depth, my_depth, my_depth);`

now



- **Paint each point grey by its own depth, seen from the spot.**
- Guess first: her head or the floor, which is darker?
- Her head is nearest the spot: darkest. The floor lightens away from it.

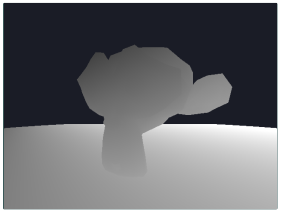
You Try 2: paint each point by stored

In the Shadow Lab, choose **You Try 2**. Guess first, then change the code.

```
let stored = textureLoad(shadow_map, vec2i(col, row), 0);
...
var c = base * (ambient + lit * k * diffuse) + lit * k * shine;
// your line here: c = ...
return vec4f(c.r, c.g, c.b, 1.0);
```

add a line here

You Try 1: my_depth



- **Now paint each point by stored: what the spot kept in its texel.**
- Guess first: where will it differ from You Try 1's picture?

You Try 2: paint each point by stored

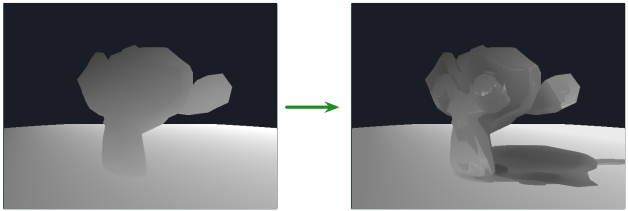
In the Shadow Lab, choose **You Try 2**. Guess first, then change the code.

```
let stored = textureLoad(shadow_map, vec2i(col, row), 0);
...
var c = base * (ambient + lit * k * diffuse) + lit * k * shine;
// your line here: c = ...
return vec4f(c.r, c.g, c.b, 1.0);
```

add a line here

answer: `c = vec3f(stored, stored, stored);`

You Try 1: my_depth



- **Now paint each point by stored: what the spot kept in its texel.**
- Guess first: where will it differ from You Try 1's picture?
- Only in her shadow: there the texel holds her head, nearer and darker.

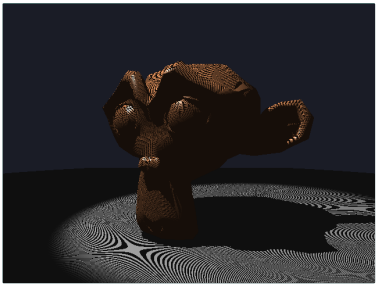
Round 2

Acne and the bias

A lit floor must not shadow itself

With no bias: stripes on every lit surface

slide 12's test, bias 0

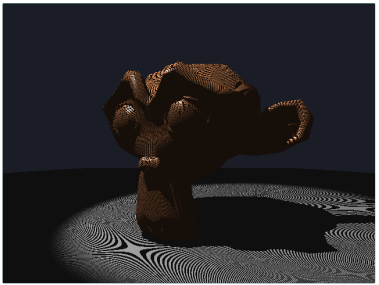


- Her shadow is right.
  Everything else is striped.

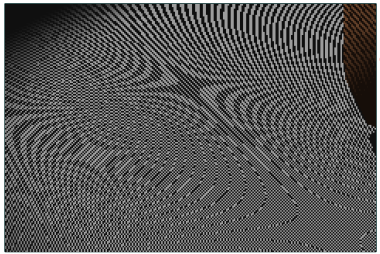
▶ **In the Shadow Lab:**
You Try 3, bias 0.0

With no bias: stripes on every lit surface

slide 12's test, bias 0



a bigger picture, close up

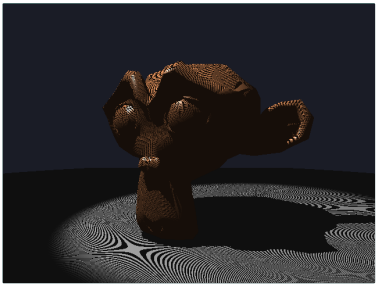


- Her shadow is right. Everything else is striped.
- Lit surfaces shadow themselves. This is called **shadow acne**.

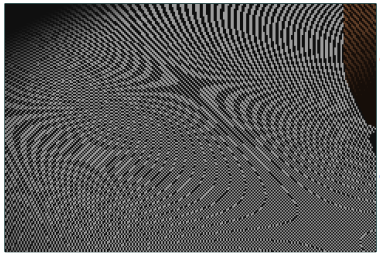
▶ **In the Shadow Lab:**
You Try 3, bias 0.0

With no bias: stripes on every lit surface

slide 12's test, bias 0



a bigger picture, close up



- Her shadow is right. Everything else is striped.
- Lit surfaces shadow themselves. This is called **shadow acne**.
- Lecture 17, slide 10, had the same trouble with rays.

Open floor and her own lit face: marked "in shadow" in stripes.

▶ **In the Shadow Lab:**
You Try 3, bias 0.0

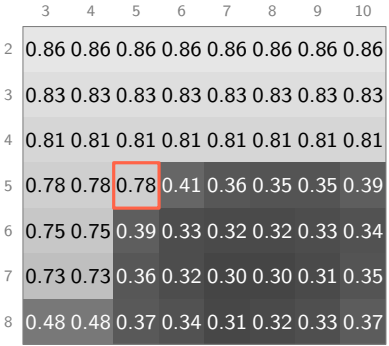
Worked: R is on open floor

$R = (1, -1, -1.75)$: open floor, nothing above it.

across 0.362, down 0.327 $\rightarrow$ col 5, row 5

its own depth: 0.79

Try first: is R lit, by slide 12's test?



Worked: R is on open floor

$R = (1, -1, -1.75)$: open floor, nothing above it.

across 0.362, down 0.327 $\rightarrow$ col 5, row 5

its own depth: 0.79

Try first: is R lit, by slide 12's test?

Own 0.79 $>$ stored 0.78: **shadow**. Wrong!

	3	4	5	6	7	8	9	10
2	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86
3	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83
4	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81
5	0.78	0.78	0.78	0.41	0.36	0.35	0.35	0.39
6	0.75	0.75	0.39	0.33	0.32	0.32	0.33	0.34
7	0.73	0.73	0.36	0.32	0.30	0.30	0.31	0.35
8	0.48	0.48	0.37	0.34	0.31	0.32	0.33	0.37

Worked: R is on open floor

$R = (1, -1, -1.75)$: open floor, nothing above it.

across 0.362, down 0.327 $\rightarrow$ col 5, row 5

its own depth: 0.79

Try first: is R lit, by slide 12's test?

Own 0.79 $>$ stored 0.78: **shadow**. Wrong!

The texel covers a patch of floor.
It stores one depth: its middle's, 0.78.
R lies in the patch's far half: 0.79.

	3	4	5	6	7	8	9	10
2	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86
3	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83
4	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81
5	0.78	0.78	0.78	0.41	0.36	0.35	0.35	0.39
6	0.75	0.75	0.39	0.33	0.32	0.32	0.33	0.34
7	0.73	0.73	0.36	0.32	0.30	0.30	0.31	0.35
8	0.48	0.48	0.37	0.34	0.31	0.32	0.33	0.37

Half of every texel's patch is
a little behind its stored depth.

Worked: R is on open floor

$R = (1, -1, -1.75)$: open floor, nothing above it.

across 0.362, down 0.327 $\rightarrow$ col 5, row 5

its own depth: 0.79

Try first: is R lit, by slide 12's test?

Own 0.79 $>$ stored 0.78: **shadow**. Wrong!

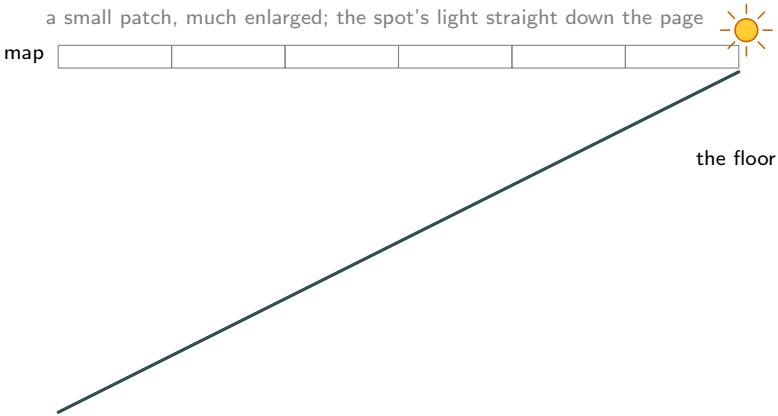
The texel covers a patch of floor.
It stores one depth: its middle's, 0.78.
R lies in the patch's far half: 0.79.

The fix, a **bias**: forgive a point that is only a little behind.
With a bias of 0.05: $0.79 - 0.05 = 0.74 \leq 0.78$: **lit**.

	3	4	5	6	7	8	9	10
2	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.86
3	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83
4	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81
5	0.78	0.78	0.78	0.41	0.36	0.35	0.35	0.39
6	0.75	0.75	0.39	0.33	0.32	0.32	0.33	0.34
7	0.73	0.73	0.36	0.32	0.30	0.30	0.31	0.35
8	0.48	0.48	0.37	0.34	0.31	0.32	0.33	0.37

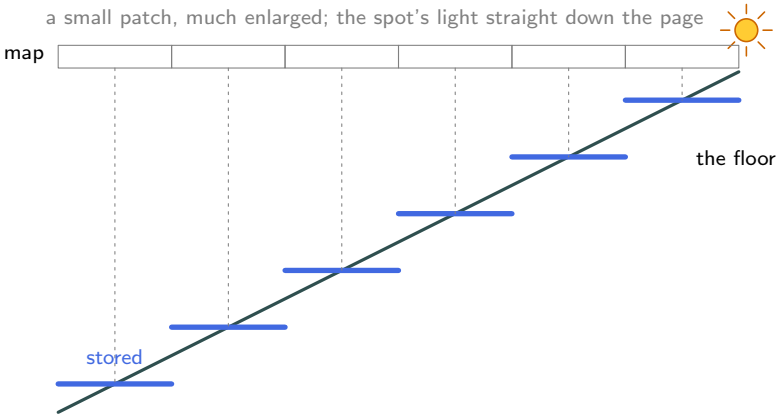
Half of every texel's patch is a little behind its stored depth.

Why: one depth per texel makes a staircase



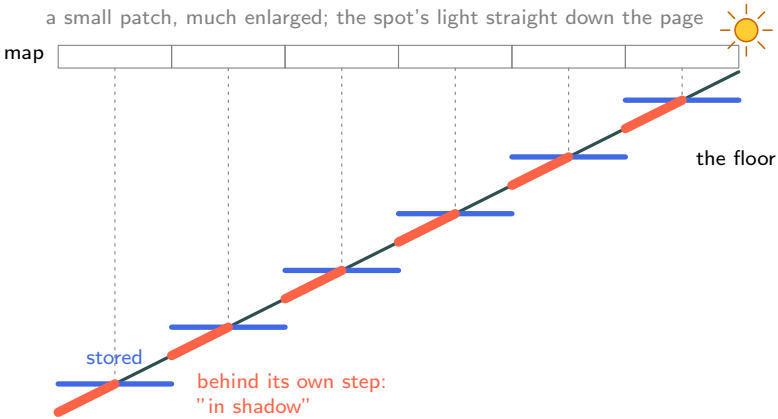
- Seen from the spot, the floor slants away.

Why: one depth per texel makes a staircase



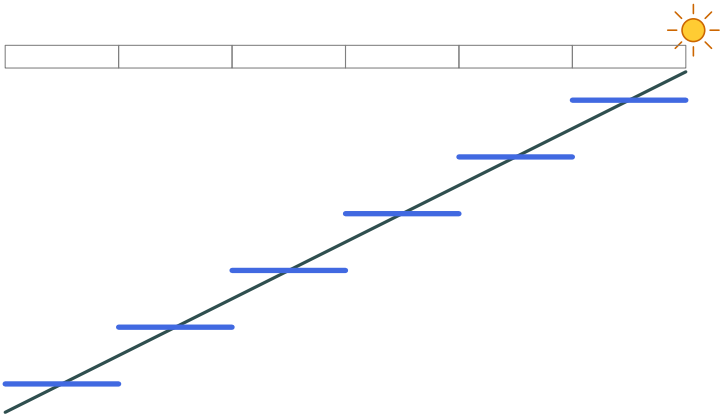
- Seen from the spot, the floor slants away.
- Each texel keeps one depth, its middle's. The map holds a staircase.

Why: one depth per texel makes a staircase



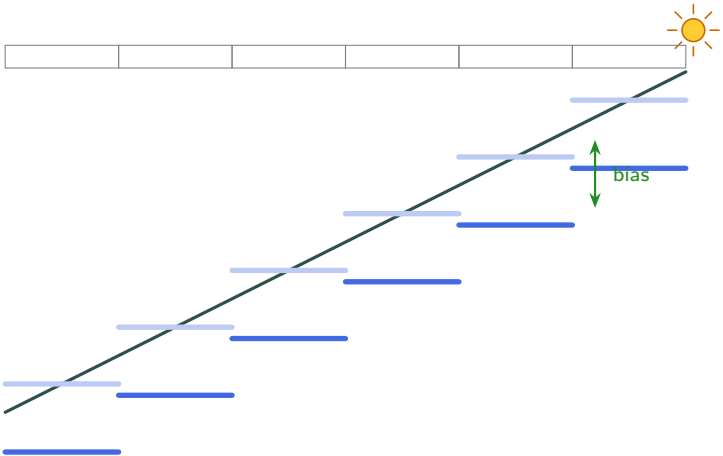
- Seen from the spot, the floor slants away.
- Each texel keeps one depth, its middle's. The map holds a staircase.
- Half of each step's floor is behind the step: stripes, one per texel.

The fix: a bias, lecture 17's ϵ



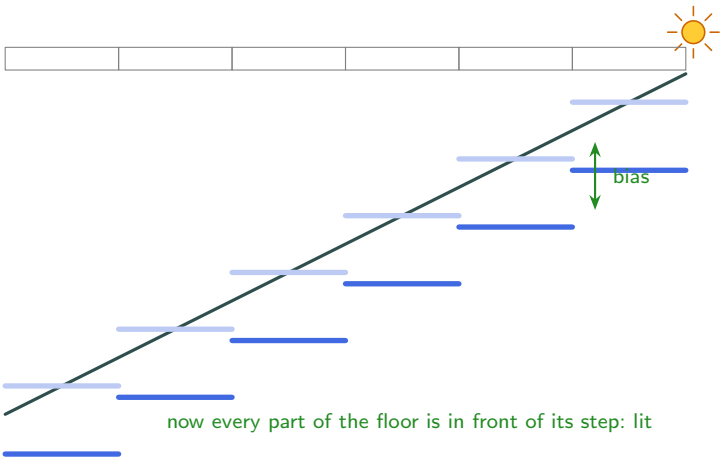
- Slide 19's staircase, before the fix.

The fix: a bias, lecture 17's ϵ



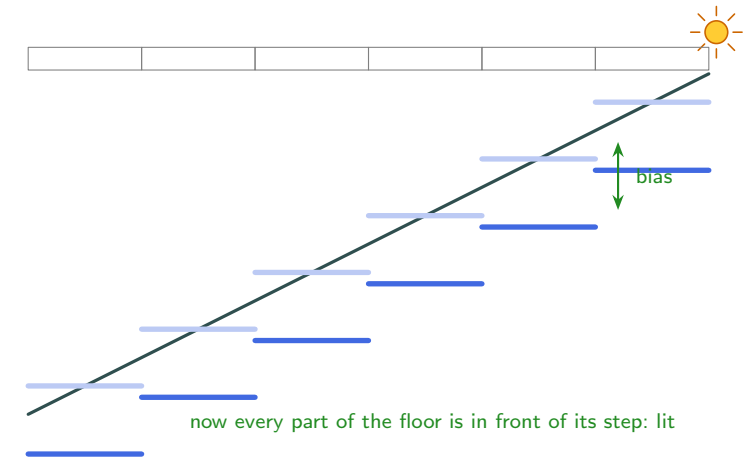
- Slide 19's staircase, before the fix.
- Subtract a small number, the **bias**, from the point's own depth.

The fix: a bias, lecture 17's ϵ



- Slide 19's staircase, before the fix.
- Subtract a small number, the **bias**, from the point's own depth.
- The steps move back, behind the whole floor.

The fix: a bias, lecture 17's ϵ

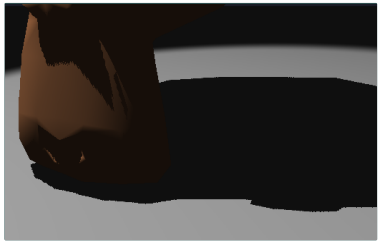



```
if (my_depth - bias > stored) { lit = 0.0; }
```

- Slide 19's staircase, before the fix.
- Subtract a small number, the **bias**, from the point's own depth.
- The steps move back, behind the whole floor.
- Lecture 17 started its ray ϵ along: the same idea.

Too much bias: Peter Panning

bias 0.01, under her chin

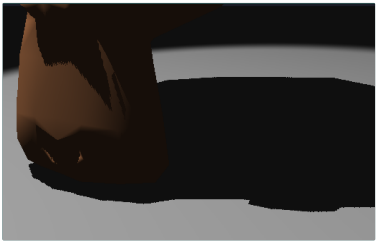


- Just big enough: the shadow reaches in under her chin.

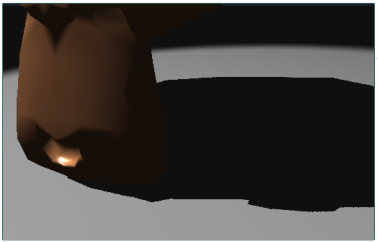
▶ **In the Shadow Lab:**
You Try 4, bias 0.1

Too much bias: Peter Panning

bias 0.01, under her chin



bias 0.1

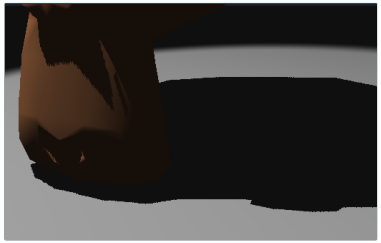


- Just big enough: the shadow reaches in under her chin.
- Too big: the floor under her chin is lit, and so is the chin itself.

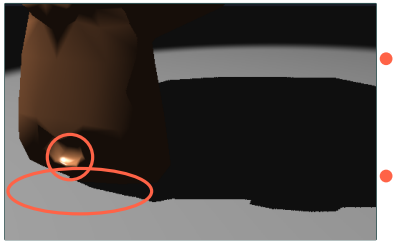
▶ **In the Shadow Lab:**
You Try 4, bias 0.1

Too much bias: Peter Panning

bias 0.01, under her chin



bias 0.1

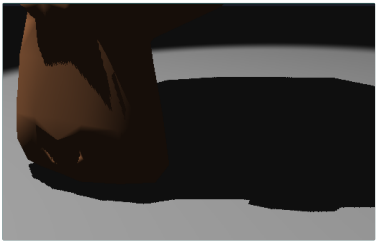


- Just big enough: the shadow reaches in under her chin.
- Too big: the floor under her chin is lit, and so is the chin itself.
- Called **Peter Panning**: the boy whose shadow came off.

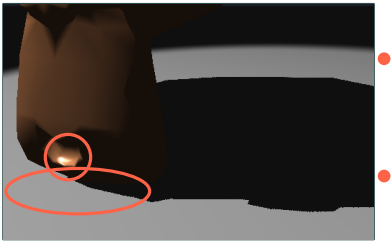
▶ **In the Shadow Lab:**
You Try 4, bias 0.1

Too much bias: Peter Panning

bias 0.01, under her chin



bias 0.1



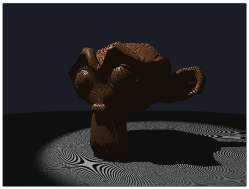
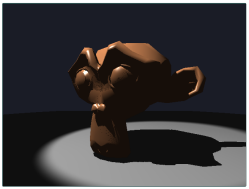
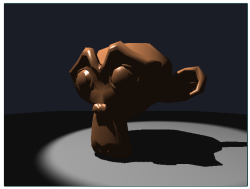
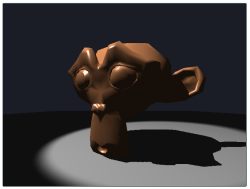
0.1 in depth is about half a unit there.

The floor under her chin is closer behind her than that: lit.

- Just big enough: the shadow reaches in under her chin.
- Too big: the floor under her chin is lit, and so is the chin itself.
- Called **Peter Panning**: the boy whose shadow came off.

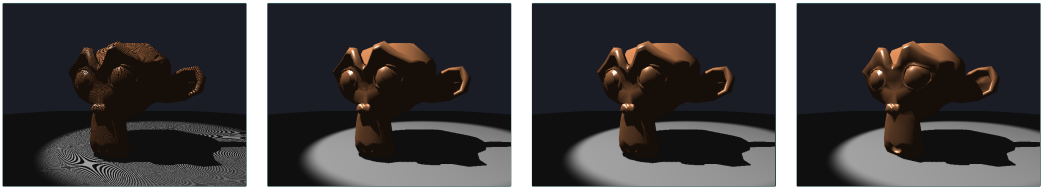
▶ **In the Shadow Lab:**
You Try 4, bias 0.1

Choose the bias between the two

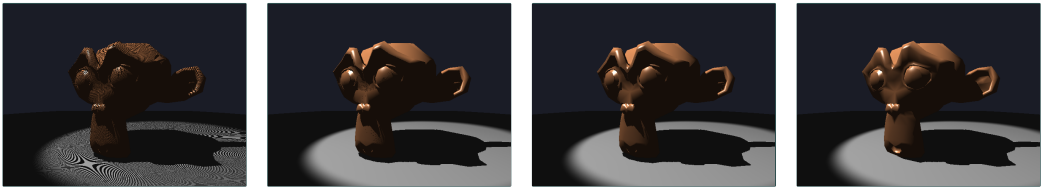


Choose the bias between the two



Too small: acne. Too big: Peter Panning. A coarse map needs a bigger bias (slide 18 used 0.05).

Choose the bias between the two



Too small: acne. Too big: Peter Panning. A coarse map needs a bigger bias (slide 18 used 0.05).

Engines also grow the bias where the light grazes a surface, since the staircase is steeper there (RTR 7.4).

You Try 3: no bias

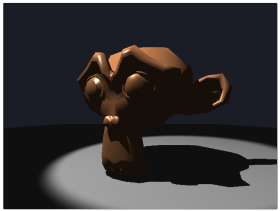
In the Shadow Lab, choose **You Try 3**. Guess first, then change the code.

```
let stored = textureLoad(shadow_map, vec2i(col, row), 0);
let bias = 0.01;
var lit = 1.0;
if (my_depth - bias > stored) { lit = 0.0; }
```

change this line

- **Set the bias to 0.0. Then find the smallest bias with no stripes.**
- Guess first: where do stripes appear?

now: bias 0.01



You Try 3: no bias

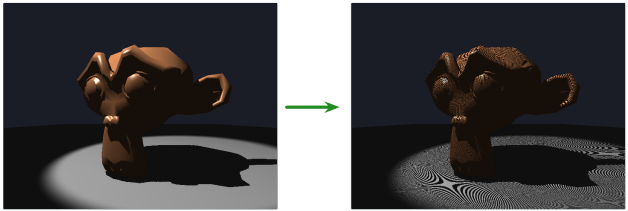
In the Shadow Lab, choose **You Try 3**. Guess first, then change the code.

```
let stored = textureLoad(shadow_map, vec2i(col, row), 0);
let bias = 0.01;
var lit = 1.0;
if (my_depth - bias > stored) { lit = 0.0; }
```

change this line

answer: `let bias = 0.0;`

now: bias 0.01



- **Set the bias to 0.0. Then find the smallest bias with no stripes.**
- Guess first: where do stripes appear?
- On the lit floor and her lit face. They go at about 0.004.

You Try 4: a big bias

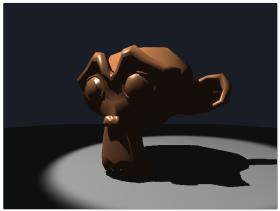
In the Shadow Lab, choose **You Try 4**. Guess first, then change the code.

```
let stored = textureLoad(shadow_map, vec2i(col, row), 0);
let bias = 0.01;
var lit = 1.0;
if (my_depth - bias > stored) { lit = 0.0; }
```

change this line

- **Set the bias to 0.1, ten times today's.**
- Guess first: which shadows go wrong?

now: bias 0.01



You Try 4: a big bias

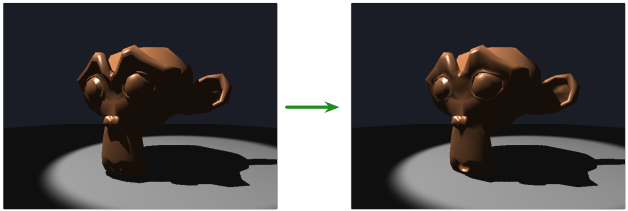
In the Shadow Lab, choose **You Try 4**. Guess first, then change the code.

```
let stored = textureLoad(shadow_map, vec2i(col, row), 0);
let bias = 0.01;
var lit = 1.0;
if (my_depth - bias > stored) { lit = 0.0; }
```

change this line

answer: `let bias = 0.1;`

now: bias 0.01



- **Set the bias to 0.1, ten times today's.**
- Guess first: which shadows go wrong?
- Under her chin the shadow goes, and her face loses its own shadows.

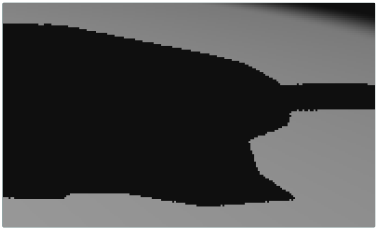
Round 3

Blocky edges and PCF

A texel is bigger than a pixel

A small map gives blocky edges

map 1024 × 1024, close up

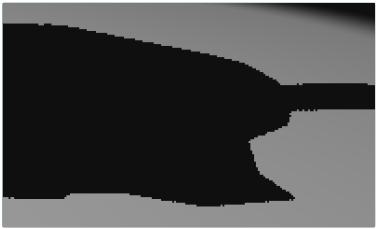


- A big map: the edge follows Suzanne's outline.

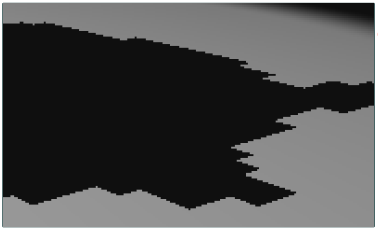
▶ **In the Shadow Lab:**
change line 2 to `// shadow map: 64`

A small map gives blocky edges

map 1024 × 1024, close up



map 64 × 64, close up

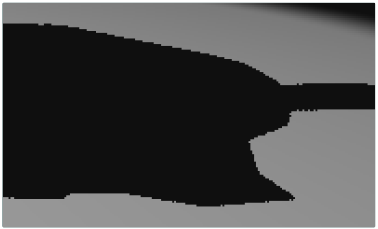


- A big map: the edge follows Suzanne's outline.
- A small map: the edge is made of steps.

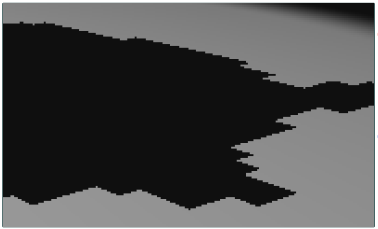
▶ **In the Shadow Lab:**
change line 2 to `// shadow map: 64`

A small map gives blocky edges

map 1024 × 1024, close up



map 64 × 64, close up



- A big map: the edge follows Suzanne's outline.
- A small map: the edge is made of steps.
- Each step is one texel, laid on the floor.

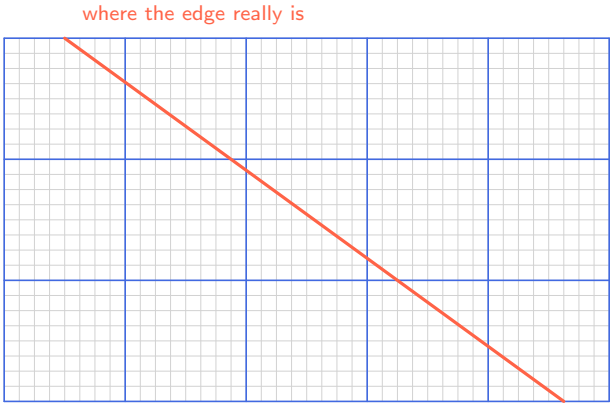
At the floor the spot's picture is about 5 units wide.

1024 texels: 0.005 units each. 64 texels: 0.08 each.

▶ **In the Shadow Lab:**
change line 2 to `// shadow map: 64`

Why: one texel covers many pixels

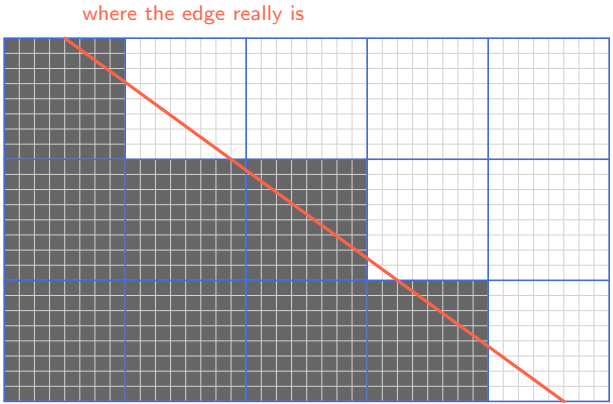
the floor seen from above



- Blue squares: texels, laid on the floor. Grey: the screen's pixels.

Why: one texel covers many pixels

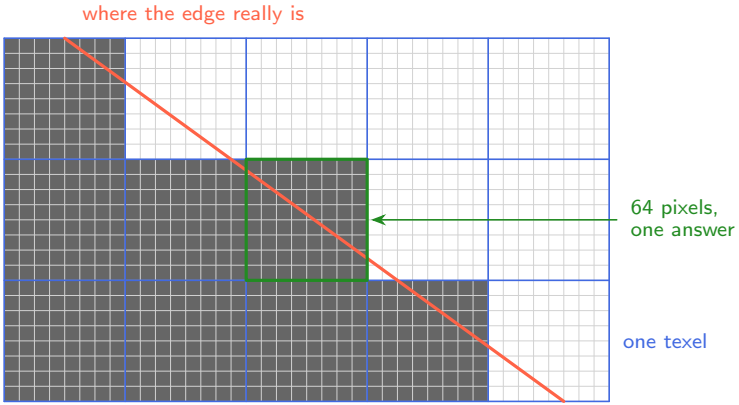
the floor seen from above



- Blue squares: texels, laid on the floor. Grey: the screen's pixels.
- Every pixel in a texel gets that texel's one answer.

Why: one texel covers many pixels

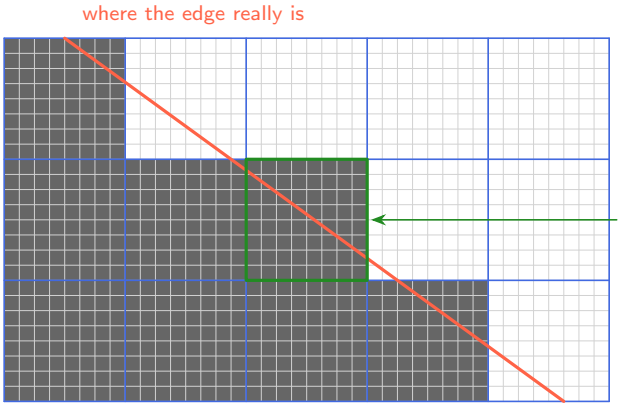
the floor seen from above



- Blue squares: texels, laid on the floor. Grey: the screen's pixels.
- Every pixel in a texel gets that texel's one answer.
- So the shadow's edge can turn only at texel borders: steps.

Why: one texel covers many pixels

the floor seen from above



- Blue squares: texels, laid on the floor. Grey: the screen's pixels.
- Every pixel in a texel gets that texel's one answer.
- So the shadow's edge can turn only at texel borders: steps.
- A bigger map costs memory and time. Can we blur the steps instead?

PCF: ask nine texels, average the answers

$S = (1.5, -1, -1)$ is on the floor. Its texel: col 8, row 5.
Own depth 0.79, bias 0.05: compare $0.79 - 0.05 = 0.74$.

Try first: how many of the nine say "lit"?

	col 7	col 8	col 9
row 4	0.81	0.81	0.81
row 5	0.36	0.35	0.35
row 6	0.32	0.32	0.33

- One texel gives 0 or 1. Nine texels give nine answers.

PCF: ask nine texels, average the answers

$S = (1.5, -1, -1)$ is on the floor. Its texel: col 8, row 5.
Own depth 0.79, bias 0.05: compare $0.79 - 0.05 = 0.74$.

Try first: how many of the nine say "lit"?

	col 7	col 8	col 9
row 4	0.81	0.81	0.81
row 5	0.36	0.35	0.35
row 6	0.32	0.32	0.33

one texel: $0.74 > 0.35$, shadow: 0

- One texel gives 0 or 1. Nine texels give nine answers.

PCF: ask nine texels, average the answers

$S = (1.5, -1, -1)$ is on the floor. Its texel: col 8, row 5.
Own depth 0.79, bias 0.05: compare $0.79 - 0.05 = 0.74$.

Try first: how many of the nine say "lit"?

	col 7	col 8	col 9
row 4	1 0.81	1 0.81	1 0.81
row 5	0 0.36	0 0.35	0 0.35
row 6	0 0.32	0 0.32	0 0.33

one texel: $0.74 > 0.35$, shadow: 0

- One texel gives 0 or 1. Nine texels give nine answers.
- Each texel: own depth minus bias $\leq$ stored? 1, else 0.

PCF: ask nine texels, average the answers

$S = (1.5, -1, -1)$ is on the floor. Its texel: col 8, row 5.
Own depth 0.79, bias 0.05: compare $0.79 - 0.05 = 0.74$.

Try first: how many of the nine say "lit"?

	col 7	col 8	col 9
row 4	1 0.81	1 0.81	1 0.81
row 5	0 0.36	0 0.35	0 0.35
row 6	0 0.32	0 0.32	0 0.33

one texel: $0.74 > 0.35$, shadow: 0

3 of 9 say lit: lit $= 3/9 = 0.33$
A third lit: a grey edge.

- One texel gives 0 or 1. Nine texels give nine answers.
- Each texel: own depth minus bias $\leq$ stored? 1, else 0.
- Their average is between 0 and 1: a soft edge.

PCF: ask nine texels, average the answers

$S = (1.5, -1, -1)$ is on the floor. Its texel: col 8, row 5.
Own depth 0.79, bias 0.05: compare $0.79 - 0.05 = 0.74$.

Try first: how many of the nine say "lit"?

	col 7	col 8	col 9
row 4	1 0.81	1 0.81	1 0.81
row 5	0 0.36	0 0.35	0 0.35
row 6	0 0.32	0 0.32	0 0.33

one texel: $0.74 > 0.35$, shadow: 0

3 of 9 say lit: lit $= 3/9 = 0.33$
A third lit: a grey edge.

Average the answers, not the depths: an average depth is still one compare.

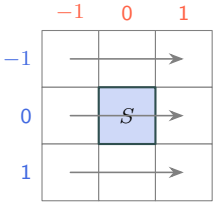
- One texel gives 0 or 1. Nine texels give nine answers.
- Each texel: own depth minus bias $\leq$ stored? 1, else 0.
- Their average is between 0 and 1: a soft edge.
- **PCF**: percentage-closer filtering.

PCF in WGSL: two loops round the texel

```
// steps 1 and 2: across, down, my_depth, size, col, row, as on slide 12
let reach = 1;           // 1: a 3 x 3 block; 2: 5 x 5
var lit_count = 0.0;    // how many texels said "lit"
var asked = 0.0;        // how many texels we asked
```

PCF in WGSL: two loops round the texel

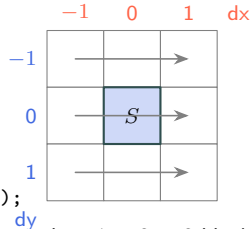
```
// steps 1 and 2: across, down, my_depth, size, col, row, as on slide 12
let reach = 1;           // 1: a 3 x 3 block; 2: 5 x 5
var lit_count = 0.0;    // how many texels said "lit"
var asked = 0.0;        // how many texels we asked
for (var dy = -reach; dy <= reach; dy++) {
    for (var dx = -reach; dx <= reach; dx++) {




    }
}
```



reach = 1: a 3 × 3 block

PCF in WGSL: two loops round the texel

```
// steps 1 and 2: across, down, my_depth, size, col, row, as on slide 12
let reach = 1;           // 1: a 3 x 3 block; 2: 5 x 5
var lit_count = 0.0;   // how many texels said "lit"
var asked = 0.0;       // how many texels we asked
for (var dy = -reach; dy <= reach; dy++) {
    for (var dx = -reach; dx <= reach; dx++) {
        let next_col = clamp(col + dx, 0, size - 1);
        let next_row = clamp(row + dy, 0, size - 1);
        let stored = textureLoad(shadow_map, vec2i(next_col, next_row), 0);
        if (my_depth - bias <= stored) { lit_count += 1.0; }
        asked += 1.0;
    }
}
```

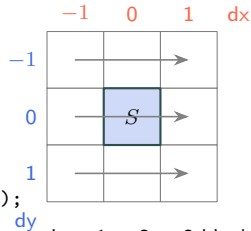


reach = 1: a 3 × 3 block

clamp (built into WGSL):
stay inside the map's edges

PCF in WGSL: two loops round the texel

```
// steps 1 and 2: across, down, my_depth, size, col, row, as on slide 12
let reach = 1;           // 1: a 3 x 3 block; 2: 5 x 5
var lit_count = 0.0;   // how many texels said "lit"
var asked = 0.0;       // how many texels we asked
for (var dy = -reach; dy <= reach; dy++) {
    for (var dx = -reach; dx <= reach; dx++) {
        let next_col = clamp(col + dx, 0, size - 1);
        let next_row = clamp(row + dy, 0, size - 1);
        let stored = textureLoad(shadow_map, vec2i(next_col, next_row), 0);
        if (my_depth - bias <= stored) { lit_count += 1.0; }
        asked += 1.0;
    }
}
let lit = lit_count / asked;  // 0.0 to 1.0
```



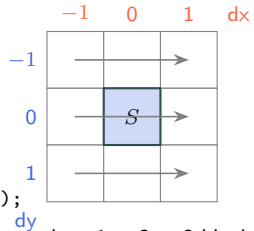
reach = 1: a 3 × 3 block

clamp (built into WGSL): stay inside the map's edges

lit: the point's share of the spot's light

PCF in WGSL: two loops round the texel

```
// steps 1 and 2: across, down, my_depth, size, col, row, as on slide 12
let reach = 1;           // 1: a 3 x 3 block; 2: 5 x 5
var lit_count = 0.0;   // how many texels said "lit"
var asked = 0.0;       // how many texels we asked
for (var dy = -reach; dy <= reach; dy++) {
    for (var dx = -reach; dx <= reach; dx++) {
        let next_col = clamp(col + dx, 0, size - 1);
        let next_row = clamp(row + dy, 0, size - 1);
        let stored = textureLoad(shadow_map, vec2i(next_col, next_row), 0);
        if (my_depth - bias <= stored) { lit_count += 1.0; }
        asked += 1.0;
    }
}
let lit = lit_count / asked;  // 0.0 to 1.0
```



reach = 1: a 3 × 3 block

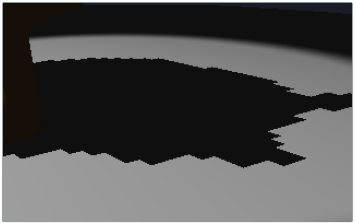
clamp (built into WGSL): stay inside the map's edges

lit: the point's share of the spot's light

GPUs can also do one 2 × 2 compare-and-average in hardware: `textureSampleCompare`. We write the loop so the compare stays in sight. The `for` loop and `+=`: lecture 29.

PCF softens the edge

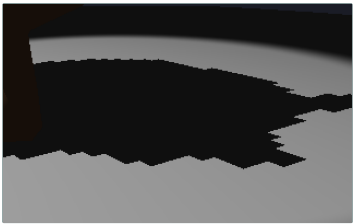
reach 0: one texel



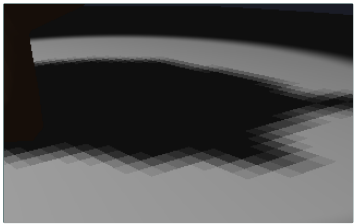
All three: the 64 $\times$ 64 map of slide 26, bias 0.02, the same close-up.

PCF softens the edge

reach 0: one texel



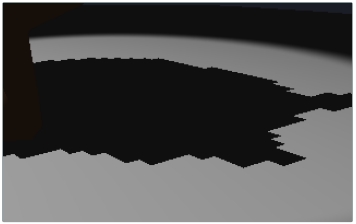
reach 1: 9 texels



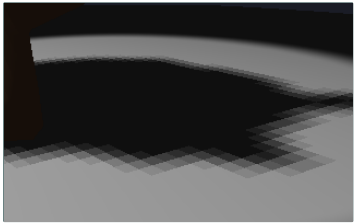
All three: the 64 $\times$ 64 map of slide 26, bias 0.02, the same close-up.

PCF softens the edge

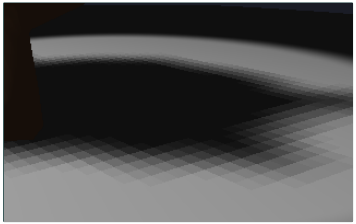
reach 0: one texel



reach 1: 9 texels



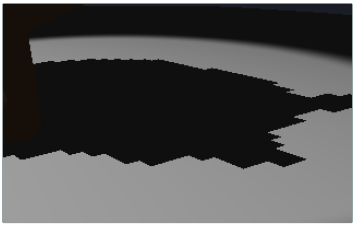
reach 2: 25 texels



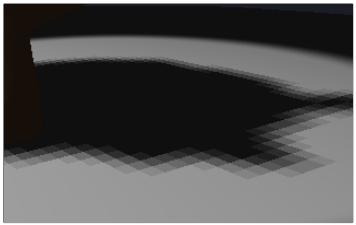
All three: the 64×64 map of slide 26, bias 0.02, the same close-up.

PCF softens the edge

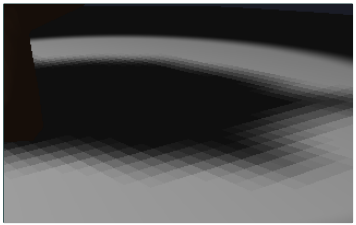
reach 0: one texel



reach 1: 9 texels



reach 2: 25 texels



All three: the 64×64 map of slide 26, bias 0.02, the same close-up.

A bigger block: a softer, wider edge, and 25 reads per pixel in place of 1.

The steps are blurred, not gone: they still show through.

A wider block also needs a bigger bias: its far texels sit further down the staircase.

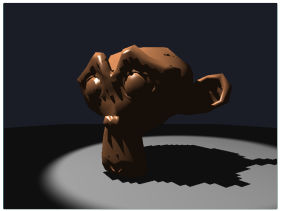
You Try 5: a small map, then PCF

In the Shadow Lab, choose **You Try 5**. Guess first, then change the code.

```
// shadow map: 64
...
let bias = 0.02;
let reach = 0;
```

change this line

now: map 64, reach 0



- **The map is only 64 × 64. Ask a 3 × 3 block.**
- Guess first: what happens to the shadow's jagged edge?

You Try 5: a small map, then PCF

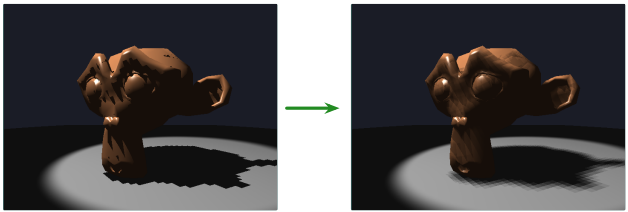
In the Shadow Lab, choose **You Try 5**. Guess first, then change the code.

```
// shadow map: 64
...
let bias = 0.02;
let reach = 0;
```

change this line

answer: `let reach = 1;`

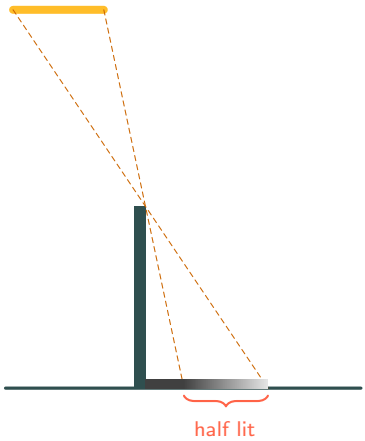
now: map 64, reach 0



- **The map is only 64 × 64. Ask a 3 × 3 block.**
- Guess first: what happens to the shadow's jagged edge?
- The steps blur into a soft grey band. Then try reach 2.

PCF's blur is not a real soft shadow

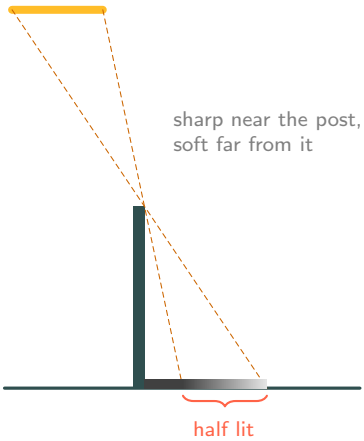
A7: a lamp with a size



- A7's lamp has a size: part of it can be hidden.

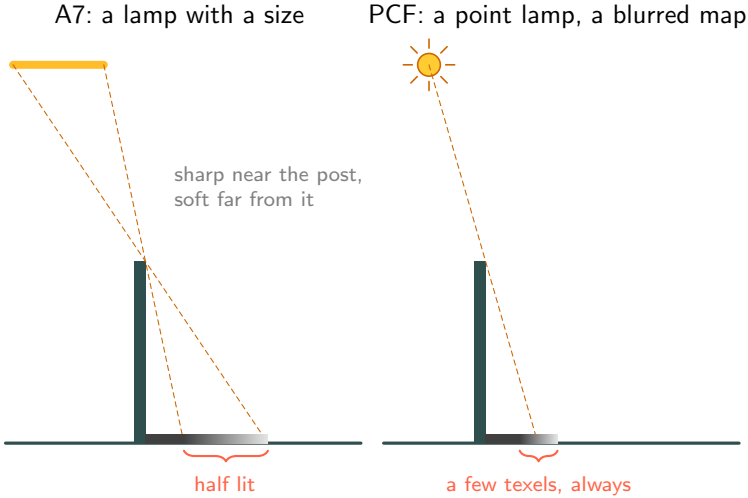
PCF's blur is not a real soft shadow

A7: a lamp with a size



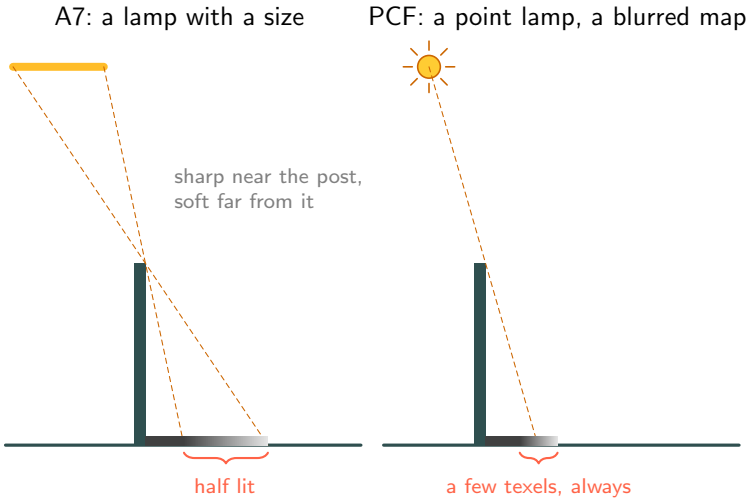
- A7's lamp has a size: part of it can be hidden.
- That half-lit band is thin near the post, wide far away.

PCF's blur is not a real soft shadow



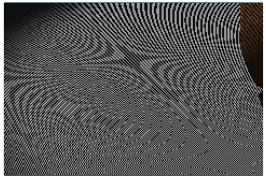
- A7's lamp has a size: part of it can be hidden.
- That half-lit band is thin near the post, wide far away.
- PCF's band is a few texels wide everywhere.

PCF's blur is not a real soft shadow



- A7's lamp has a size: part of it can be hidden.
- That half-lit band is thin near the post, wide far away.
- PCF's band is a few texels wide everywhere.
- A7's band cost many rays per pixel.

What the shadow map gets wrong



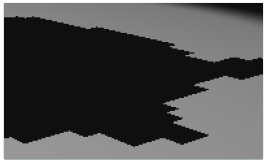
Acne

one depth per texel: a staircase



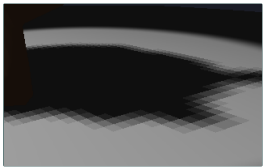
Peter Panning

a bias big enough to hide the staircase



Blocky edges

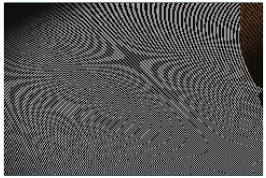
a texel covers many pixels



A fake soft edge

PCF's band is a few texels wide everywhere

What the shadow map gets wrong



Acne

one depth per texel: a staircase

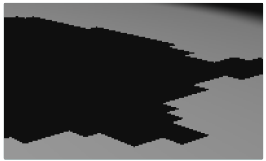
a ray starts at ϵ; no texels



Peter Panning

a bias big enough to hide the staircase

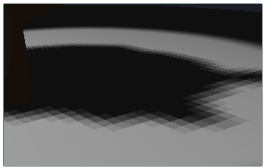
a small ϵ is enough for a ray



Blocky edges

a texel covers many pixels

a ray per pixel: exact edges

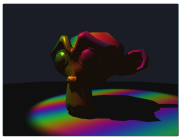


A fake soft edge

PCF's band is a few texels wide everywhere

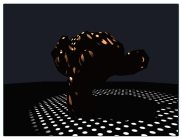
A7: many rays, a real half-lit band

Now make your own picture



A slide projector

`cos(18.85 * across + ...)`



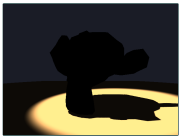
A mirror ball

`fract(across * 28.0)`



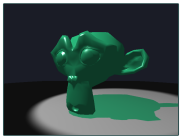
Film noir

`step(0.45, fract(...))`



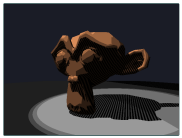
A shadow theatre

`screen = warm * lit * k`



A glass Suzanne

`mix(glass, white, lit)`



A comic book

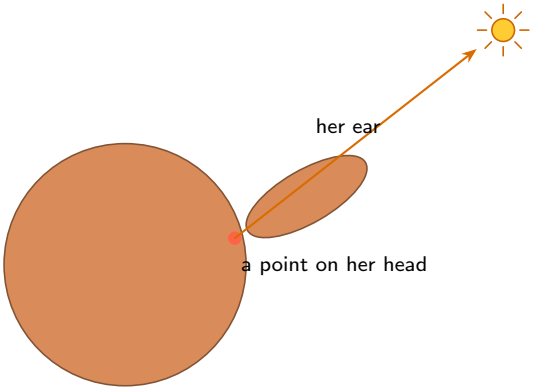
`floor(k * diffuse * 3.0)`

All six are in the Shadow Lab, under **Ideas**. Change one. Mix two. Show your neighbour.

Summary

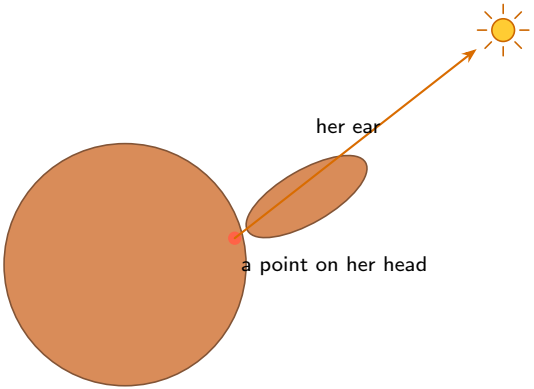
1	Pass 1: the shadow map	the scene from the spot, depth only lamp_vs and `u.lamp_mvp`: the spot as lecture 28's camera
2	Pass 2: the shadow test	own depth $>$ stored depth $\Rightarrow$ shadow `lamp` divided by w: across, down, my_depth; textureLoad
3	The bias	`my_depth - bias > stored` too small: acne; too big: Peter Panning
4	PCF	average the answers of a block of texels softens blocky edges; a blur, not a real half-lit band
5	The trade	one extra drawing in place of a ray per pixel fast, and four artefacts that lecture 17's ray never had

Next lecture: the shadow ray returns



- Lecture 31: a shadow ray in the fragment shader, against her own triangles.

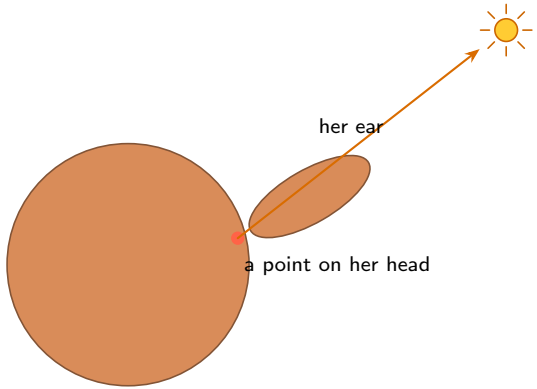
Next lecture: the shadow ray returns



968 triangles: is any of them on the ray?

- Lecture 31: a shadow ray in the fragment shader, against her own triangles.
- First a test against a ball round her: most rays miss it.

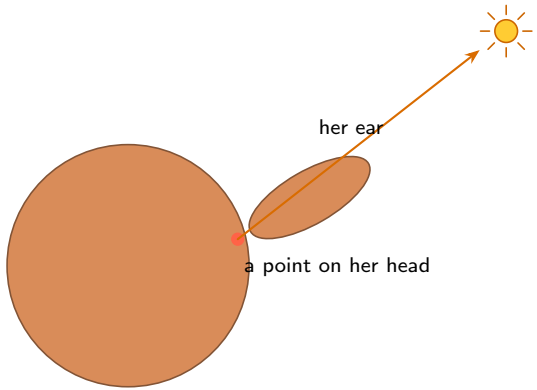
Next lecture: the shadow ray returns



968 triangles: is any of them on the ray?

- Lecture 31: a shadow ray in the fragment shader, against her own triangles.
- First a test against a ball round her: most rays miss it.
- Only points the spot lights send a ray.

Next lecture: the shadow ray returns



968 triangles: is any of them on the ray?

- Lecture 31: a shadow ray in the fragment shader, against her own triangles.
- First a test against a ball round her: most rays miss it.
- Only points the spot lights send a ray.
- Beside today's picture: no acne, no steps.

Exit Ticket

Today's question *(2 minutes)*

(a) A floor point lands at across $= 0.40$, down $= 0.35$ in a 16×16 shadow map. Its own depth is 0.78. Which texel (col, row)? That texel stores 0.41. With a bias of 0.05, is the point lit or in shadow?

(b) With PCF 3×3, four of the nine texels say "lit". How much of the spot's light does it get?

CS 453 — Exit Ticket #30

Name: _____

Roll #: _____ Date: _____

Answer:

Hand it to me on your way out.
No name, no attendance.