Lecture 29: Suzanne

A Real Model, Coloured Lamps and a Spotlight

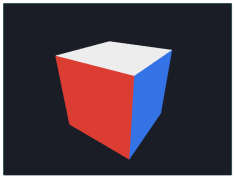
CS 453 — Computer Graphics

Fakhir Shaheen

Forman Christian University

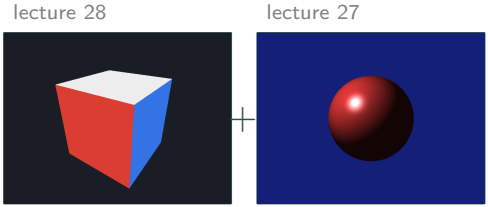
Lecture 28's corners, lecture 27's light

lecture 28



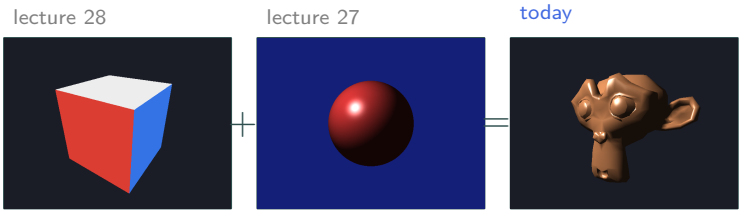
- Lecture 28: vs_main moved a cube's corners.

Lecture 28's corners, lecture 27's light



- Lecture 28: vs_main moved a cube's corners.
- Lecture 27: fs_main lit a fake ball by its $\mathbf{n}$.

Lecture 28's corners, lecture 27's light



`suzanne.py` lecture 28's cube.py, with Suzanne in the buffer

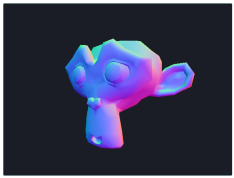
`vs_main` lecture 28's `u.mvp * ...`, and it turns the normal

`fs_main` lecture 27's light: $\mathbf{n}$, $\mathbf{l}$, $\mathbf{v}$, $\mathbf{h}$

- Lecture 28: vs_main moved a cube's corners.
- Lecture 27: fs_main lit a fake ball by its $\mathbf{n}$.
- Today: both at once, on a real model. Then more lamps.

Today: three rounds of slides, then you try

1 **A model is a file of triangles**



then You Try 1, 2

2 **Lecture 27's light on Suzanne**



then You Try 3, 4

3 **Coloured lamps and a spotlight**



then You Try 5, 6

Open the Suzanne Lab now. It is on the course calendar, in lecture 29's row.

It is lecture 28's Cube Lab with Suzanne in it. Work in pairs: one screen for two.

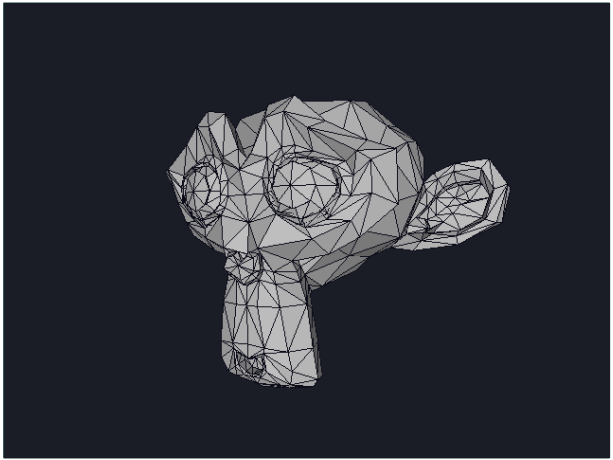
A slide with a blue **Run it in the Suzanne Lab** box has its code in the lab too.

Round 1

A model is a file of triangles

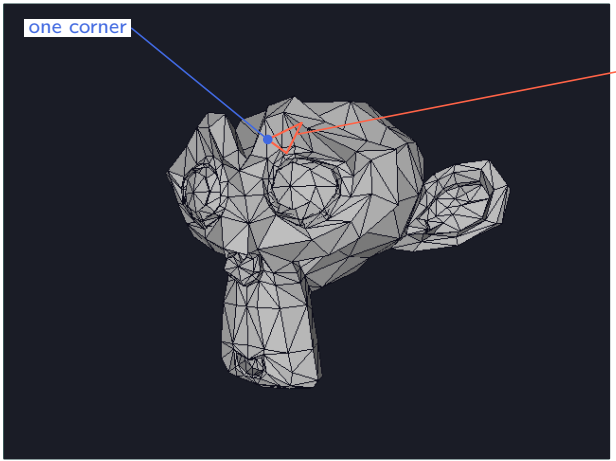
and every corner brings its own normal

Suzanne: 968 triangles where the cube had 12



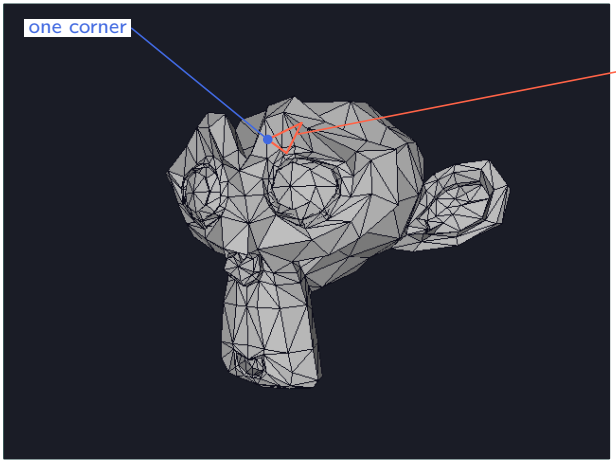
- Blender's test monkey, Suzanne: a head made of flat triangles.

Suzanne: 968 triangles where the cube had 12



- Blender's test monkey, Suzanne: a head made of flat triangles.
- Each triangle has three corners. Neighbours meet at their corners.

Suzanne: 968 triangles where the cube had 12



lecture 28's cube



- Blender's test monkey, Suzanne: a head made of flat triangles.
- Each triangle has three corners. Neighbours meet at their corners.
- Lecture 28's cube (slide 5): 12 triangles. Same idea, more of them.

An OBJ file: corners, normals, triangles

code/suzanne.obj (a text file)

```
v  0.437500 0.164062 0.765625
v  -0.437500 0.164062 0.765625
v  0.500000 0.093750 0.687500
```

... 507 lines of v: a point's x, y, z

- v: where each point is.



corner 1

An OBJ file: corners, normals, triangles

code/suzanne.obj (a text file)

```
v  0.437500 0.164062 0.765625
v  -0.437500 0.164062 0.765625
v  0.500000 0.093750 0.687500
```
... 507 lines of v: a point's x, y, z

```
vn 0.510775 -0.495297 0.702702
vn 0.510775 0.495297 0.702702
```
... 507 lines of vn: a point's normal

- v: where each point is.
- vn: the normal of the real, round surface at that point.



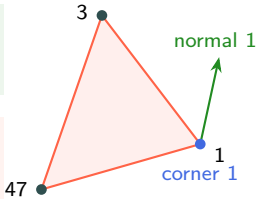
An OBJ file: corners, normals, triangles

code/suzanne.obj (a text file)

```
v  0.437500 0.164062 0.765625
v  -0.437500 0.164062 0.765625
v  0.500000 0.093750 0.687500
```
... 507 lines of v: a point's x, y, z

```
vn 0.510775 -0.495297 0.702702
vn 0.510775 0.495297 0.702702
```
... 507 lines of vn: a point's normal

```
f  47//47 1//1 3//3
f  47//47 3//3 45//45
```
... 968 lines of f: a triangle



- v: where each point is.
- vn: the normal of the real, round surface at that point.
- f: three point numbers make one triangle. Neighbours share points.

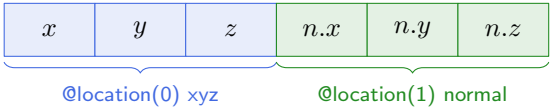
47//47: point 47, with normal 47. Counting starts at 1.

A corner: a normal where the cube had a colour

lecture 28's corner (slide 5):

x	y	z	red	green	blue

today's:



24 bytes:
corner_layout
unchanged

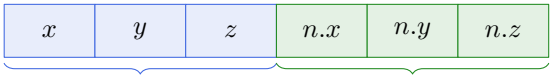
- The cube's colour slot now holds a normal.

A corner: a normal where the cube had a colour

lecture 28's corner (slide 5):

x	y	z	red	green	blue

today's:



24 bytes: corner_layout unchanged

```
for a, b, c in triangles:        each f line: three point numbers;
    for k in (a, b, c):          each corner brings its own normal
        rows.append(list(points[k]) + list(normals[k]))
```

- The cube's colour slot now holds a normal.
- Each triangle's corners are copied in, as cube.py copied each face's.

A corner: a normal where the cube had a colour

lecture 28's corner (slide 5):

x	y	z	red	green	blue

today's:

x	y	z	$n.x$	$n.y$	$n.z$

24 bytes: corner_layout unchanged

@location(0) xyz @location(1) normal

```
for a, b, c in triangles:
    for k in (a, b, c):
        rows.append(list(points[k]) + list(normals[k]))
```

each f line: three point numbers; each corner brings its own normal

python suzanne.py lit.wgsl prints:

```
507 points and 968 triangles in suzanne.obj

2904 corners x 24 bytes = 69696 bytes in the vertex buffer
```

- The cube's colour slot now holds a normal.
- Each triangle's corners are copied in, as cube.py copied each face's.
- draw(2904): vs_main runs 2904 times a frame, not 36.

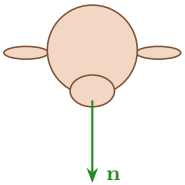
vs_main turns the normal too

```
out.pos = u.mvp * vec4f(xyz.x, xyz.y, xyz.z, 1.0);
```

- Lecture 28: u.mvp lands each corner on the screen.

Grey: lecture 28, slide 26.

from above: in the file

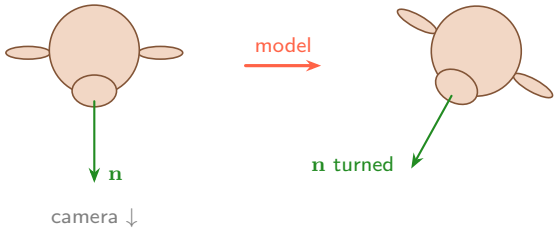


camera ↓

vs_main turns the normal too

```
out.pos = u.mvp * vec4f(xyz.x, xyz.y, xyz.z, 1.0);
let world = u.model * vec4f(xyz.x, xyz.y, xyz.z, 1.0);
let turned = u.model * vec4f(normal.x, normal.y, normal.z, 0.0);
```

Grey: lecture 28, slide 26.
u.model: M alone (lecture 28, slide 25), a new field of u.
from above: in the file turned 0.5 radians



- Lecture 28: u.mvp lands each corner on the screen.
- The light needs the world: M alone places P and turns $\mathbf{n}$.
- $w = 0$: a direction turns but never moves (lecture 8, slide 21).

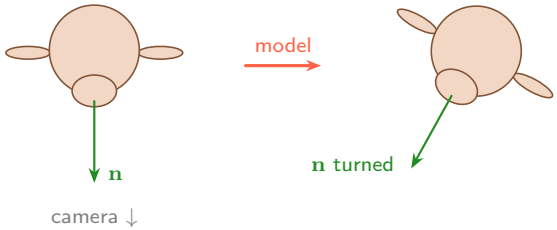
vs_main turns the normal too

```
out.pos = u.mvp * vec4f(xyz.x, xyz.y, xyz.z, 1.0);
let world = u.model * vec4f(xyz.x, xyz.y, xyz.z, 1.0);
let turned = u.model * vec4f(normal.x, normal.y, normal.z, 0.0);
out.P = vec3f(world.x, world.y, world.z);
out.normal = vec3f(turned.x, turned.y, turned.z);
```

Grey: lecture 28, slide 26.

u.model: M alone (lecture 28, slide 25), a new field of u.

from above: in the file turned 0.5 radians

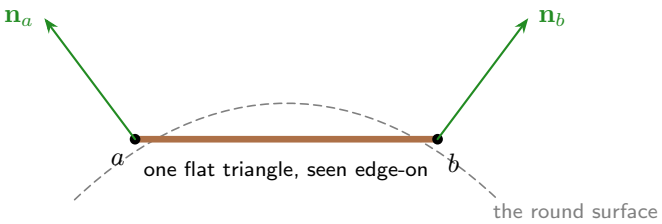


- Lecture 28: u.mvp lands each corner on the screen.
- The light needs the world: M alone places P and turns $\mathbf{n}$.
- $w = 0$: a direction turns but never moves (lecture 8, slide 21).
- Ours only turns. A stretched model needs another matrix.

Between the stages, the normals are blended

$\mathbf{n}_a = (-0.6,\ 0.8,\ 0)$, $\mathbf{n}_b = (0.6,\ 0.8,\ 0)$, both of length 1.

Try first: the normal halfway from a to b. How long is it?

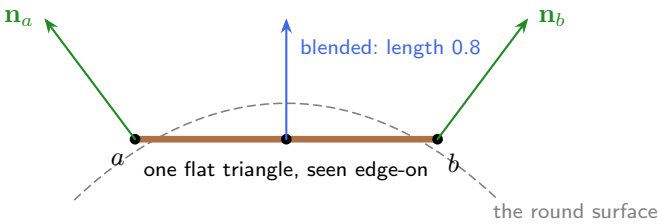


- Each pixel gets a blend of its triangle's three corner normals.
- Lecture 24 (slide 41): the weights blend whatever the corners carry.

Between the stages, the normals are blended

$\mathbf{n}_a = (-0.6,\ 0.8,\ 0)$, $\mathbf{n}_b = (0.6,\ 0.8,\ 0)$, both of length 1.

halfway: $\frac{1}{2}\mathbf{n}_a + \frac{1}{2}\mathbf{n}_b = (0,\ 0.8,\ 0)$, length 0.8



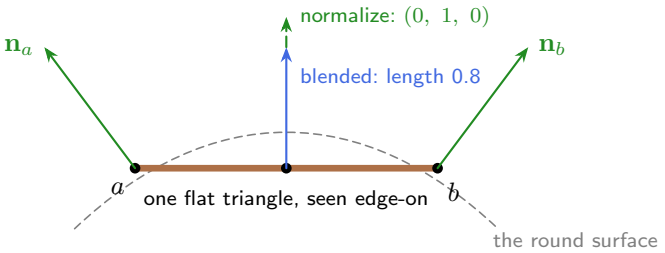
- Each pixel gets a blend of its triangle's three corner normals.
- Lecture 24 (slide 41): the weights blend whatever the corners carry.
- A blend of unit arrows is shorter than 1.

Between the stages, the normals are blended

$\mathbf{n}_a = (-0.6,\ 0.8,\ 0)$, $\mathbf{n}_b = (0.6,\ 0.8,\ 0)$, both of length 1.

halfway: $\frac{1}{2}\mathbf{n}_a + \frac{1}{2}\mathbf{n}_b = (0,\ 0.8,\ 0)$, length 0.8

```
let n = normalize(normal);    fs_main's first line
```

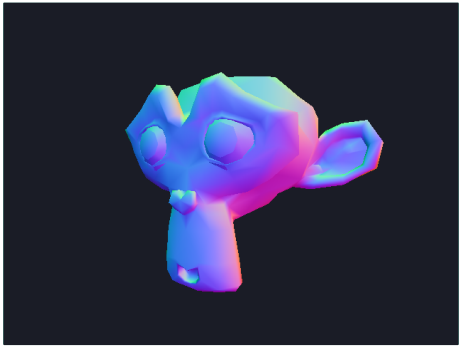


- Each pixel gets a blend of its triangle's three corner normals.
- Lecture 24 (slide 41): the weights blend whatever the corners carry.
- A blend of unit arrows is shorter than 1.
- So the fragment shader makes it length 1 again.

Colour by the normal: see $\mathbf{n}$

```
let n = normalize(normal);
let c = 0.5 * n + 0.5;
```

Lecture 27, slide 8: red $n.x$, green $n.y$, blue $n.z$.



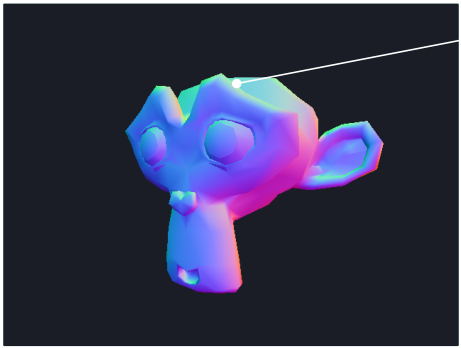
- Lecture 27's check, on a real model.

▶ **In the Suzanne Lab:**
Her normals as colours

Colour by the normal: see $\mathbf{n}$

```
let n = normalize(normal);
let c = 0.5 * n + 0.5;
```

Lecture 27, slide 8: red $n.x$, green $n.y$, blue $n.z$.



top of her head: faces up green

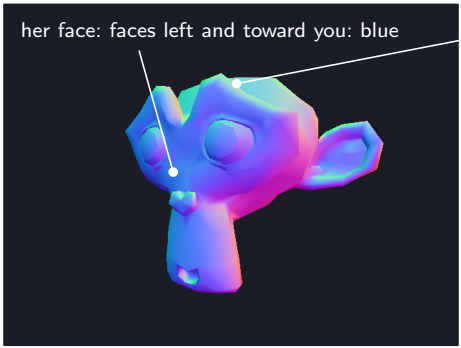
- Lecture 27's check, on a real model.

▶ **In the Suzanne Lab:**
Her normals as colours

Colour by the normal: see $\mathbf{n}$

```
let n = normalize(normal);
let c = 0.5 * n + 0.5;
```

Lecture 27, slide 8: red $n.x$, green $n.y$, blue $n.z$.



top of her head: faces up green

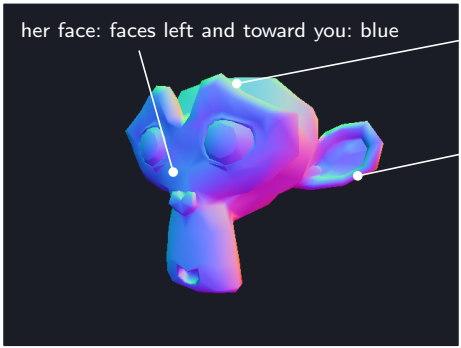
- Lecture 27's check, on a real model.
- She is turned 0.5 radians: her face is blue, her ear pink.

▶ **In the Suzanne Lab:**
Her normals as colours

Colour by the normal: see $\mathbf{n}$

```
let n = normalize(normal);
let c = 0.5 * n + 0.5;
```

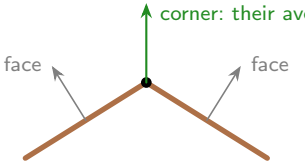
Lecture 27, slide 8: red $n.x$, green $n.y$, blue $n.z$.



- Lecture 27's check, on a real model.
- She is turned 0.5 radians: her face is blue, her ear pink.
- No seams, no steps: the blended normals are smooth.

▶ **In the Suzanne Lab:**
Her normals as colours

Smooth or flat: where the corner normals come from



A corner's normal: the average of the faces around it.
The file's vn lines hold it.

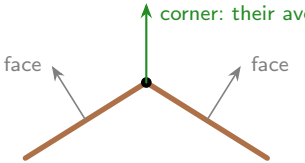
- Flat: you count the triangles.

flat: each triangle its face's normal



▶ **In the Suzanne Lab:**
Lecture 27's light on Suzanne, then tick Flat triangles

Smooth or flat: where the corner normals come from

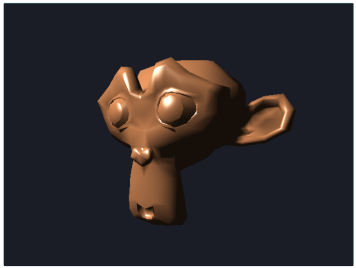


A corner's normal: the average of the faces around it.
The file's vn lines hold it.

flat: each triangle its face's normal



smooth: the corners' normals, blended



- Flat: you count the triangles.
- Smooth: the same 968 triangles look round.
- Not one corner moved. Only the normals differ. Her outline still has corners.

▶ **In the Suzanne Lab:**
Lecture 27's light on Suzanne, then tick Flat triangles

You Try 1: light from straight above

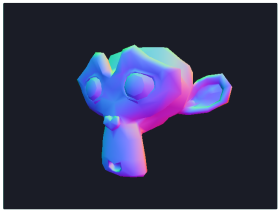
In the Suzanne Lab, choose **You Try 1**. Guess first, then change the code.

```
let n = normalize(normal);
let c = 0.5 * n + 0.5;
return vec4f(c.r, c.g, c.b, 1.0);
```

change this line

Then drag the slider M: turn. Does the white stay on top while she turns?

now



make this



- **Paint her grey by $n.y$: how much each point faces up.**
- Guess first: which parts of her will be white?

You Try 1: light from straight above

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

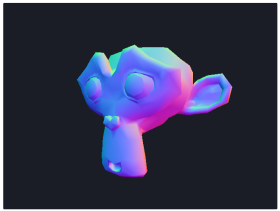
```
let n = normalize(normal);
let c = 0.5 * n + 0.5;
return vec4f(c.r, c.g, c.b, 1.0);
```

change this line

Then drag the slider M: turn. Does the white stay on top while she turns?

answer: `let c = vec3f(n.y, n.y, n.y);`

now



make this



- **Paint her grey by $n.y$: how much each point faces up.**
- Guess first: which parts of her will be white?
- The top of her head and brow: $n.y$ near 1. Under her chin $n.y < 0$: black.

You Try 2: colour her by height

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

```
let bottom = vec3f(0.1, 0.2, 0.8);
let top = vec3f(1.0, 0.85, 0.2);
let c = bottom;
```

change this line

P.y: about -1 at her chin, 1 on top. mix: lecture 26, slide 11.

now



make this



- **Blue at her chin, yellow on top, blended.**
- Guess first: what turns P.y's -1 to 1 into 0 to 1?

You Try 2: colour her by height

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

```
let bottom = vec3f(0.1, 0.2, 0.8);
let top = vec3f(1.0, 0.85, 0.2);
let c = bottom;
```

change this line

P.y: about -1 at her chin, 1 on top. mix: lecture 26, slide 11.

answer: `let c = mix(bottom, top, 0.5 + 0.5 * P.y);`

now



make this



- **Blue at her chin, yellow on top, blended.**
- Guess first: what turns P.y's -1 to 1 into 0 to 1?
- $0.5 + 0.5\times$ P.y, as lecture 27 did for the normal's colours.

Round 2

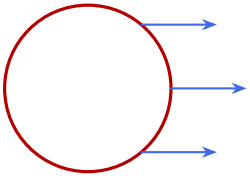
Lecture 27's light on Suzanne

one new line: a real eye

One new line: $\mathbf{v}$ points to the real eye

lecture 27: parallel

$$\mathbf{v} = (0,\ 0,\ 1)$$



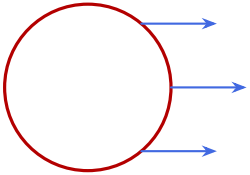
every point: the same $\mathbf{v}$

- Lecture 27 looked straight in from far away.

One new line: $\mathbf{v}$ points to the real eye

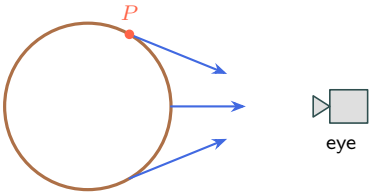
lecture 27: parallel

$\mathbf{v} = (0,\ 0,\ 1)$



every point: the same $\mathbf{v}$

today: one eye



each point: its own $\mathbf{v}$

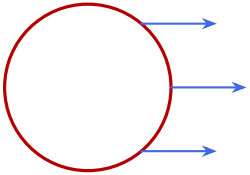
- Lecture 27 looked straight in from far away.
- Today lecture 28's camera is a point: $\mathbf{v}$ runs from P to it.

One new line: $\mathbf{v}$ points to the real eye

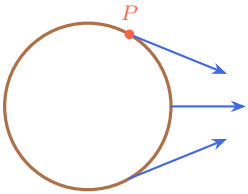
lecture 27: parallel

today: one eye

$\mathbf{v} = (0,\ 0,\ 1)$



every point: the same $\mathbf{v}$



each point: its own $\mathbf{v}$



- Lecture 27 looked straight in from far away.
- Today lecture 28's camera is a point: $\mathbf{v}$ runs from P to it.
- Lecture 17 did the same: $\mathbf{v}$ toward the eye.
- Everything else in lecture 27's light stays.

```
let v = normalize(u.eye - P);
```

u.eye $= (0, 0, 3)$: lecture 28's camera (slide 20), at $z = 3$

Lecture 27's lines, on Suzanne

```
let n = normalize(normal);          new
let base = vec3f(0.85, 0.55, 0.35);
let lamp = vec3f(-3.0, 3.0, 3.0);
let l = normalize(lamp - P);
let diffuse = max(dot(n, l), 0.0);
let ambient = 0.1;
let v = normalize(u.eye - P);       new
let h = normalize(l + v);
let shine = pow(max(dot(n, h), 0.0), 50.0);
let c = base * (ambient + diffuse) + shine;
```

grey: lecture 27, slides 14 and 19, unchanged

base: clay. lamp: up, left, in front:
lecture 27's lamp, three times as far.

- Two new lines. Eight are lecture 27's.

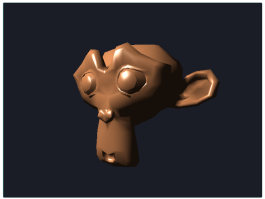
▶ **In the Suzanne Lab:**
Lecture 27's light on Suzanne

Lecture 27's lines, on Suzanne

```
let n = normalize(normal);          new
let base = vec3f(0.85, 0.55, 0.35);
let lamp = vec3f(-3.0, 3.0, 3.0);
let l = normalize(lamp - P);
let diffuse = max(dot(n, l), 0.0);
let ambient = 0.1;
let v = normalize(u.eye - P);       new
let h = normalize(l + v);
let shine = pow(max(dot(n, h), 0.0), 50.0);
let c = base * (ambient + diffuse) + shine;
```

grey: lecture 27, slides 14 and 19, unchanged

base: clay. lamp: up, left, in front: lecture 27's lamp, three times as far.

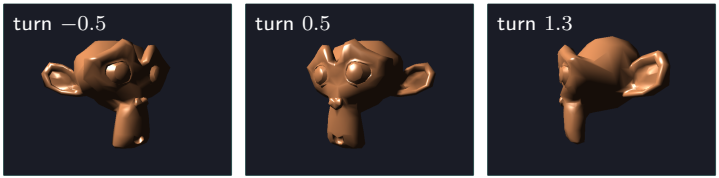


- Two new lines. Eight are lecture 27's.
- Lamp, ambient and highlight, on 968 triangles.
- P and normal come from vs_main, blended.

▶ **In the Suzanne Lab:**
Lecture 27's light on Suzanne

The lamp stays put while she turns

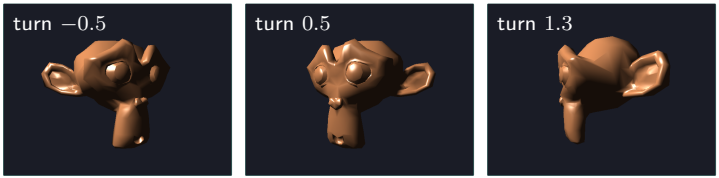
the normal turned with her ($w = 0$ line): right



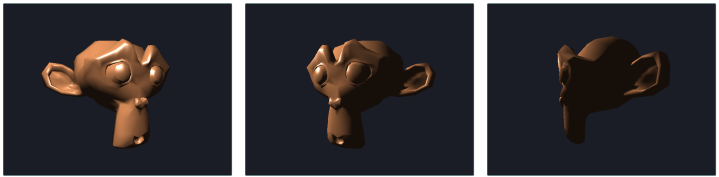
- Right: the lamp is up and left. The light stays there as she turns.

The lamp stays put while she turns

the normal turned with her ($w = 0$ line): right



the normal NOT turned: wrong

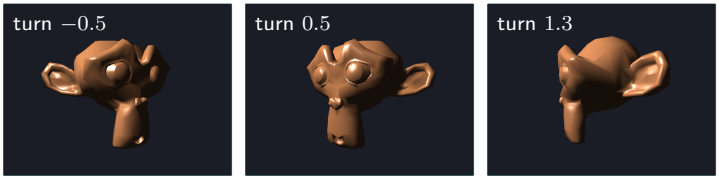


```
let turned = vec4f(normal.x, normal.y, normal.z, 0.0);
```
no u.model

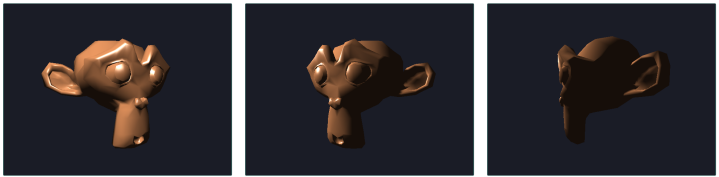
- Right: the lamp is up and left. The light stays there as she turns.
- Wrong: the light is glued to her face. It turns with her.

The lamp stays put while she turns

the normal turned with her ($w = 0$ line): right



the normal NOT turned: wrong



```
let turned = vec4f(normal.x, normal.y, normal.z, 0.0);    no u.model
```

- Right: the lamp is up and left. The light stays there as she turns.
- Wrong: the light is glued to her face. It turns with her.
- The fix is one matrix on the normal: slide 8.

You Try 3: a spooky light

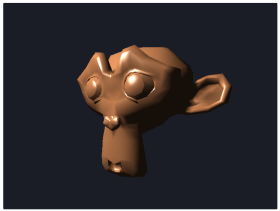
In the Suzanne Lab, choose **You Try 3**. Guess first, then change the code.

```
let lamp = vec3f(-3.0, 3.0, 3.0);
let l = normalize(lamp - P);
let diffuse = max(dot(n, l), 0.0);
```

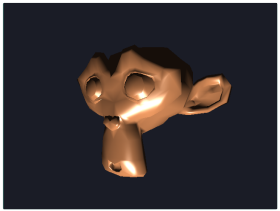
change this line

lamp: x right, y up, z toward the camera. Her middle is $(0, 0, 0)$.

now



make this



- **Put the lamp below her face: a torch under the chin.**
- Guess first: which numbers change, and to what?

You Try 3: a spooky light

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

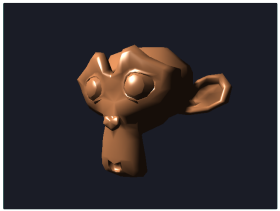
```
let lamp = vec3f(-3.0, 3.0, 3.0);
let l = normalize(lamp - P);
let diffuse = max(dot(n, l), 0.0);
```

change this line

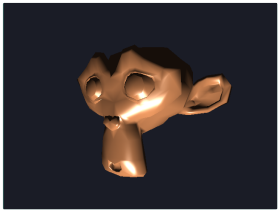
lamp: x right, y up, z toward the camera. Her middle is $(0, 0, 0)$.

answer: `let lamp = vec3f(0.0, -3.0, 3.0);`

now



make this



- **Put the lamp below her face: a torch under the chin.**
- Guess first: which numbers change, and to what?
- x to 0 (the middle), y to -3 (below). z stays: in front.

You Try 4: a gold Suzanne

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

```
let base = vec3f(0.85, 0.55, 0.35);
let shine = pow(max(dot(n, h), 0.0), 50.0);
let c = base * (ambient + diffuse) + shine;
```

change these two

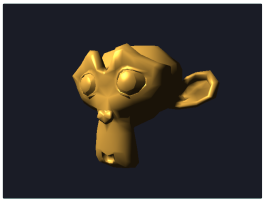
Lecture 17: a metal's highlight takes the metal's colour.

- **Make her gold, highlight and all: (1.0, 0.75, 0.3).**
- Guess first: how does the shine take her colour?

now



make this



You Try 4: a gold Suzanne

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

```
let base = vec3f(0.85, 0.55, 0.35);
let shine = pow(max(dot(n, h), 0.0), 50.0);
let c = base * (ambient + diffuse) + shine;
```

change these two

Lecture 17: a metal's highlight takes the metal's colour.

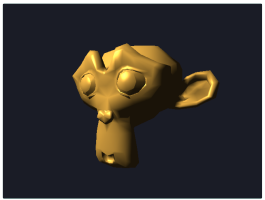
answer:

```
let base = vec3f(1.0, 0.75, 0.3);
let c = base * (ambient + diffuse + shine);
```

now



make this



- **Make her gold, highlight and all: (1.0, 0.75, 0.3).**
- Guess first: how does the shine take her colour?
- Multiply it by base: move shine inside the brackets.

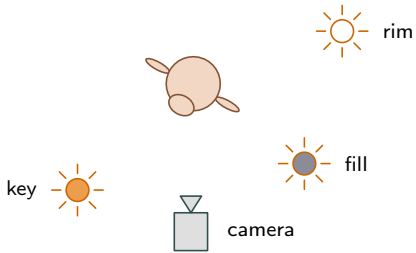
Round 3

Coloured lamps and a spotlight

interesting light, the way a film set does it

Three lamps, placed as on a film set

from above



- Key: bright and warm, in front, to one side.

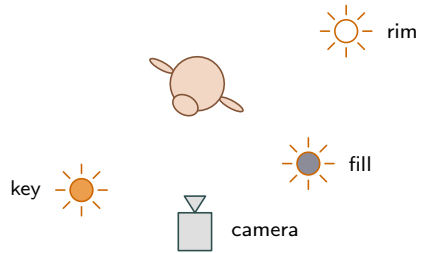






Three lamps, placed as on a film set

from above



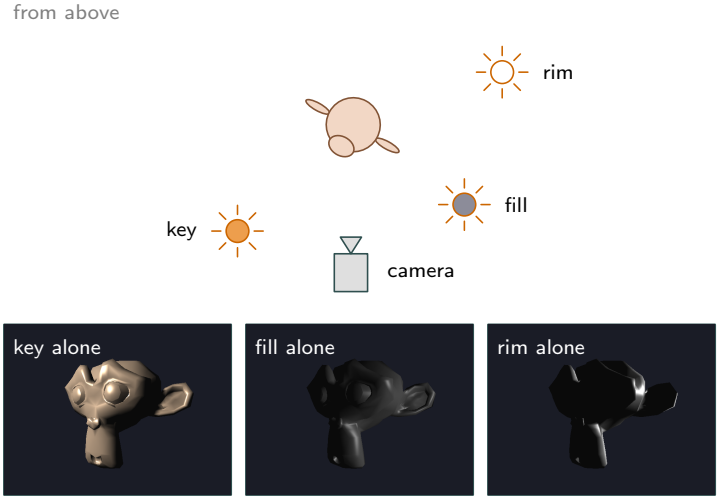






- Key: bright and warm, in front, to one side.
- Fill: dim, on the other side: the dark side is not black.

Three lamps, placed as on a film set



- Key: bright and warm, in front, to one side.
- Fill: dim, on the other side: the dark side is not black.
- Rim: behind her. It lights her edges, which lifts her off the background.

A loop adds the lamps' shares

```
var lamps = array<vec3f, 3>(
    vec3f(-3.0, 2.0, 3.0),      // key
    vec3f(3.0, 0.0, 2.0),       // fill
    vec3f(4.0, 1.0, -1.5));     // rim
var colours = array<vec3f, 3>(
    vec3f(1.0, 0.85, 0.7),     // key: warm white
    vec3f(0.3, 0.3, 0.3),      // fill: dim
    vec3f(1.0, 1.0, 1.0));     // rim: white
```



A loop adds the lamps' shares

```
var lamps = array<vec3f, 3>(
    vec3f(-3.0, 2.0, 3.0),      // key
    vec3f(3.0, 0.0, 2.0),       // fill
    vec3f(4.0, 1.0, -1.5));     // rim
var colours = array<vec3f, 3>(
    vec3f(1.0, 0.85, 0.7),     // key: warm white
    vec3f(0.3, 0.3, 0.3),      // fill: dim
    vec3f(1.0, 1.0, 1.0));     // rim: white

var c = base * 0.05;
for (var i = 0; i < 3; i++) {
    let l = normalize(lamps[i] - P);
    let h = normalize(l + v);
    let diffuse = max(dot(n, l), 0.0);
    let shine = pow(max(dot(n, h), 0.0), 50.0);
    c += colours[i] * (base * diffuse + shine);
}
```



var: a name whose value can change. c += x means c = c + x.

Lecture 16, slide 43: add every light's share. i++: add 1 to i.

A loop adds the lamps' shares

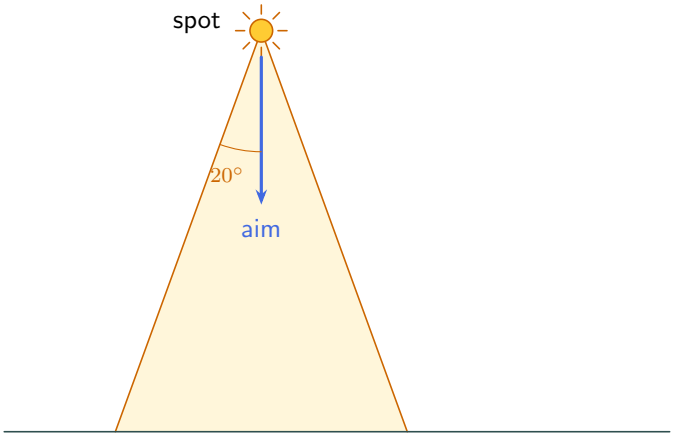
```
var lamps = array<vec3f, 3>(
    vec3f(-3.0, 2.0, 3.0),      // key
    vec3f(3.0, 0.0, 2.0),       // fill
    vec3f(4.0, 1.0, -1.5));     // rim
var colours = array<vec3f, 3>(
    vec3f(1.0, 0.85, 0.7),     // key: warm white
    vec3f(0.3, 0.3, 0.3),      // fill: dim
    vec3f(1.0, 1.0, 1.0));     // rim: white

var c = base * 0.05;
for (var i = 0; i < 3; i++) {
    let l = normalize(lamps[i] - P);
    let h = normalize(l + v);
    let diffuse = max(dot(n, l), 0.0);
    let shine = pow(max(dot(n, h), 0.0), 50.0);
    c += colours[i] * (base * diffuse + shine);
}
```



var: a name whose value can change. c += x means c = c + x.

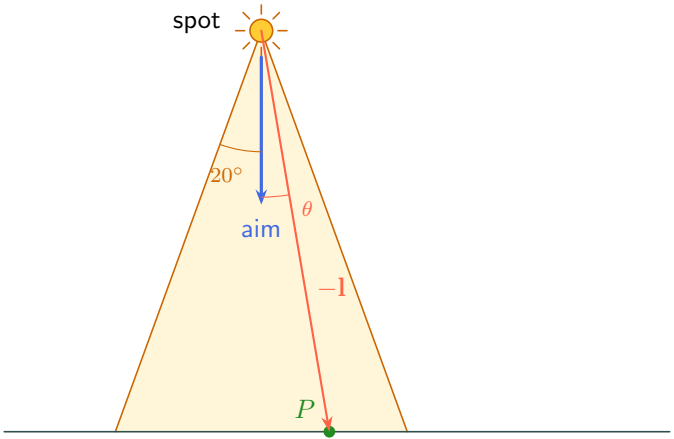
Lecture 16, slide 43: add every light's share. i++: add 1 to i.

A spotlight lights only a cone



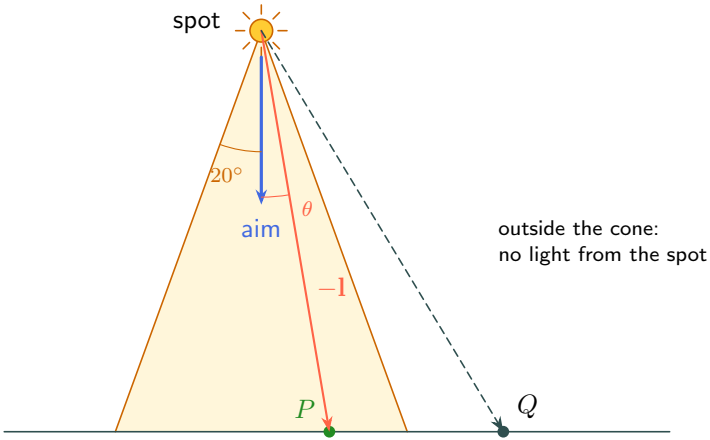
- A spotlight is a lamp with an aim and a cone.

A spotlight lights only a cone



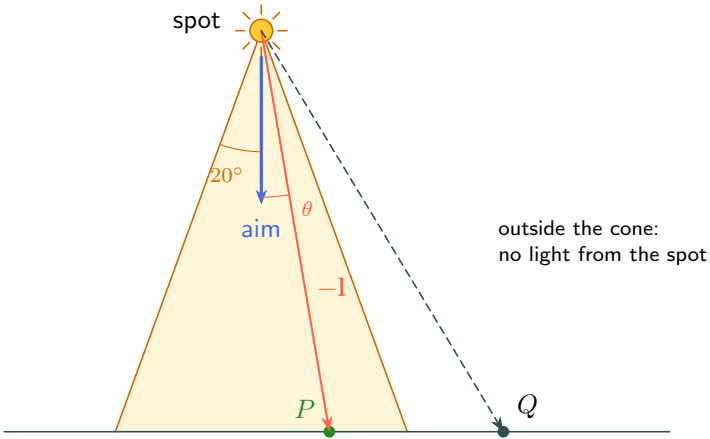
- A spotlight is a lamp with an aim and a cone.
- $-\mathbf{l}$: from the spot to the point. θ: its angle off the aim.

A spotlight lights only a cone



- A spotlight is a lamp with an aim and a cone.
- $-\mathbf{l}$: from the spot to the point. θ: its angle off the aim.
- Lit only if θ is less than the cone's $20°$.

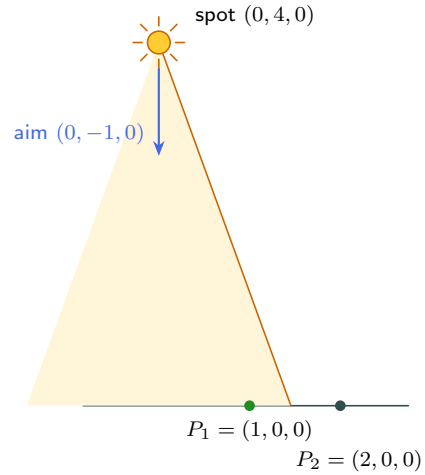
A spotlight lights only a cone



- A spotlight is a lamp with an aim and a cone.
- $-\mathbf{l}$: from the spot to the point. θ: its angle off the aim.
- Lit only if θ is less than the cone's $20°$.
- No angle needed: compare cosines. A smaller θ, a bigger $\cos\theta$.

`let inside = dot(-l, aim);` $= \cos\theta$, both of length 1

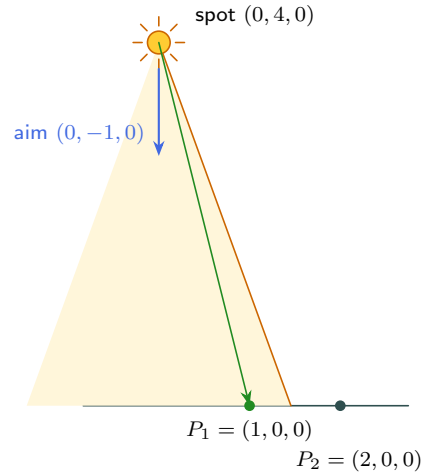
Worked: inside the cone, or not?



The cone: $20°$, $\cos 20° = 0.94$.

Try first: is each point lit by the spot?

Worked: inside the cone, or not?



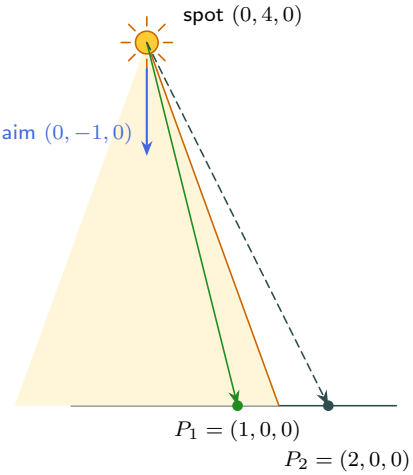
The cone: $20°$, $\cos 20° = 0.94$.

$P_1 -$ spot $= (1, -4, 0)$, length $\sqrt{17} = 4.12$

$-\mathbf{l} = (0.24,\ -0.97,\ 0)$, $-\mathbf{l} \cdot \mathbf{aim} = 0.97$

$0.97 > 0.94$: inside, lit

Worked: inside the cone, or not?



The cone: $20°$, $\cos 20° = 0.94$.

$P_1 -$ spot $= (1, -4, 0)$, length $\sqrt{17} = 4.12$

$-\mathbf{l} = (0.24,\ -0.97,\ 0)$, $-\mathbf{l} \cdot \mathbf{aim} = 0.97$

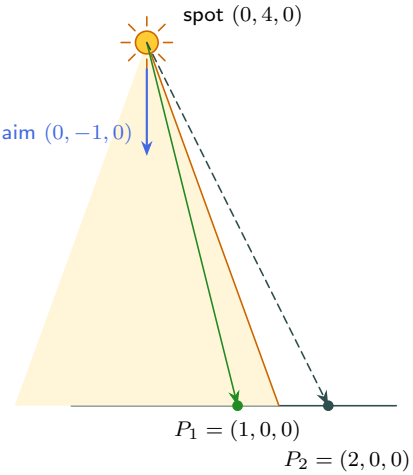
$0.97 > 0.94$: inside, lit

$P_2 -$ spot $= (2, -4, 0)$, length $\sqrt{20} = 4.47$

$-\mathbf{l} \cdot \mathbf{aim} = 4/4.47 = 0.89$

$0.89 < 0.94$: outside, dark

Worked: inside the cone, or not?



The cone: $20°$, $\cos 20° = 0.94$.

$P_1 -$ spot $= (1, -4, 0)$, length $\sqrt{17} = 4.12$

$-\mathbf{l} = (0.24,\ -0.97,\ 0)$, $-\mathbf{l} \cdot \mathbf{aim} = 0.97$

$0.97 > 0.94$: inside, lit

$P_2 -$ spot $= (2, -4, 0)$, length $\sqrt{20} = 4.47$

$-\mathbf{l} \cdot \mathbf{aim} = 4/4.47 = 0.89$

$0.89 < 0.94$: outside, dark

Inside means: $\cos\theta$ bigger than the cone's cosine.

A soft edge: smoothstep between two cones

```
let inner = cos(radians(6.0));   // full light inside 6°
let outer = cos(radians(9.0));   // no light outside 9°
let k = smoothstep(outer, inner, inside);
let c = base * (ambient + k * diffuse) + k * shine;
```

radians (built in): degrees to radians. smoothstep: lecture 26, slide 20.

step: a hard edge



- k: 1 inside the inner cone, 0 outside the outer.

▶ **In the Suzanne Lab:**
A spotlight

A soft edge: smoothstep between two cones

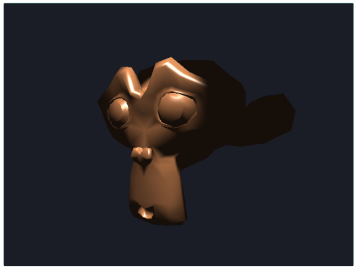
```
let inner = cos(radians(6.0));   // full light inside 6°
let outer = cos(radians(9.0));   // no light outside 9°
let k = smoothstep(outer, inner, inside);
let c = base * (ambient + k * diffuse) + k * shine;
```

radians (built in): degrees to radians. smoothstep: lecture 26, slide 20.

step: a hard edge



smoothstep: a soft edge



- k: 1 inside the inner cone, 0 outside the outer.
- Between the two, smoothstep fades it.
- k scales only the spot's own light. Ambient stays.

▶ **In the Suzanne Lab:**
A spotlight

You Try 5: a blue fill lamp

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

```
vec3f(1.0, 0.85, 0.7),      // key: warm white
vec3f(0.3, 0.3, 0.3),       // fill: dim
vec3f(1.0, 1.0, 1.0));      // rim: white
```

change this line

The colours array. The fill lamp is on the right of the picture.

- **Make the fill lamp blue: (0.1, 0.3, 0.8).**
- Before you run it: which side of her face turns blue?

now



after your change



You Try 5: a blue fill lamp

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

```
vec3f(1.0, 0.85, 0.7),      // key: warm white
vec3f(0.3, 0.3, 0.3),       // fill: dim
vec3f(1.0, 1.0, 1.0));      // rim: white
```

change this line

The colours array. The fill lamp is on the right of the picture.

answer:

```
vec3f(0.1, 0.3, 0.8),       // fill: cool blue
```

now



after your change



- **Make the fill lamp blue: (0.1, 0.3, 0.8).**
- Before you run it: which side of her face turns blue?
- The right, where the key does not reach. Warm key, cool fill: a film look.

You Try 6: a torch that follows the mouse

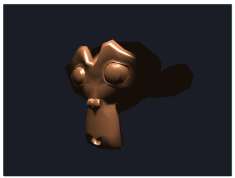
In the Suzanne Lab, choose **You Try 6**. Guess first, then change the code.

```
let mx = 1.73 * (u.mouse.x - u.resolution.x / 2.0) / (u.resolution.y / 2.0);
let spot = vec3f(0.0, 4.0, 3.0);
let aim_at = vec3f(-0.4, 0.1, 0.7);
```

change this line

mx, my: the mouse in world units (lecture 28's fy/dist, backwards). aim_at: her nose.

- **Aim the spotlight at the mouse.**
- Guess first: which line says where the spot is aimed?



now: aimed at her nose, whatever the mouse does

You Try 6: a torch that follows the mouse

In the Suzanne Lab, choose **You Try 6**. Guess first, then change the code.

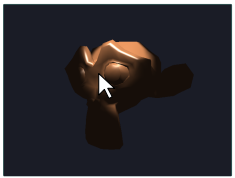
```
let mx = 1.73 * (u.mouse.x - u.resolution.x / 2.0) / (u.resolution.y / 2.0);
let spot = vec3f(0.0, 4.0, 3.0);
let aim_at = vec3f(-0.4, 0.1, 0.7);
```

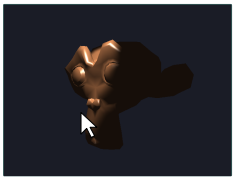
change this line

mx, my: the mouse in world units (lecture 28's $f\,y$/dist, backwards). aim_at: her nose.

answer: `let aim_at = vec3f(mx, my, 0.0);`



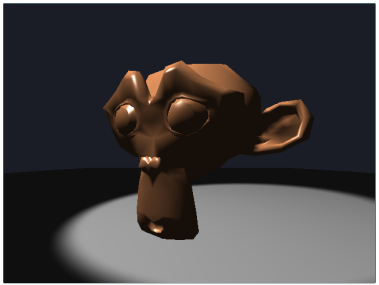




- **Aim the spotlight at the mouse.**
- Guess first: which line says where the spot is aimed?
- aim_at. The spot stays put; only its aim follows the mouse.

Still missing: her shadow

today: a spotlight from the upper left

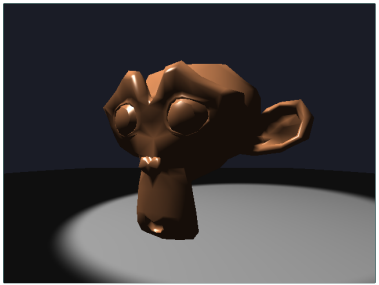


- The light goes straight through her to the floor.

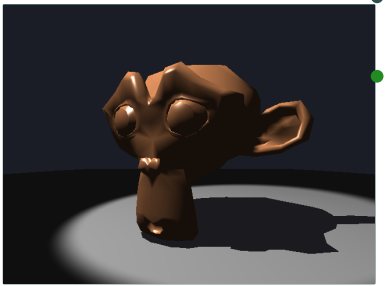
▶ **In the Suzanne Lab:**
On a floor: where is her shadow?

Still missing: her shadow

today: a spotlight from the upper left



where her shadow belongs

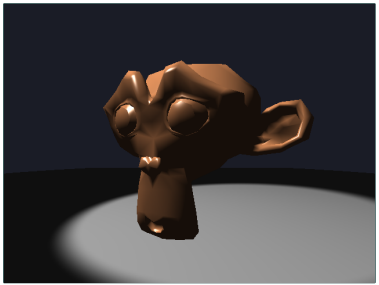


- The light goes straight through her to the floor.
- A real lamp is blocked by her head.

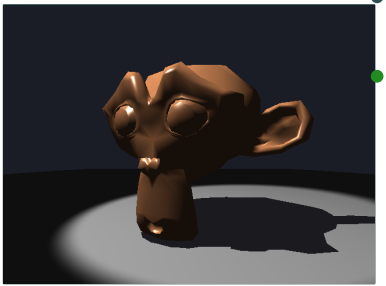
▶ **In the Suzanne Lab:**
On a floor: where is her shadow?

Still missing: her shadow

today: a spotlight from the upper left



where her shadow belongs



- The light goes straight through her to the floor.
- A real lamp is blocked by her head.

Each pixel knows only its own point, its normal and the lamp.

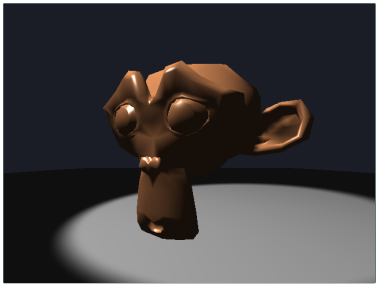
Nobody asks: *is something between this point and the lamp?*

The skipped ray: lecture 17's shadow ray (slide 7).

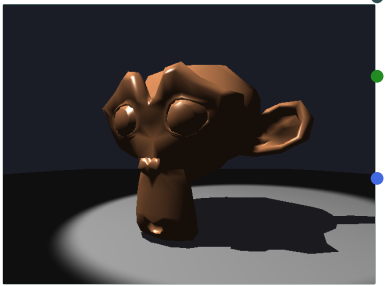
▶ **In the Suzanne Lab:**
On a floor: where is her shadow?

Still missing: her shadow

today: a spotlight from the upper left



where her shadow belongs



- The light goes straight through her to the floor.
- A real lamp is blocked by her head.
- Lecture 30: a shadow map. Lecture 31: the shadow ray itself.

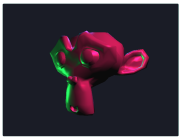
Each pixel knows only its own point, its normal and the lamp.

Nobody asks: *is something between this point and the lamp?*

The skipped ray: lecture 17's shadow ray (slide 7).

▶ **In the Suzanne Lab:**
On a floor: where is her shadow?

Now make your own picture



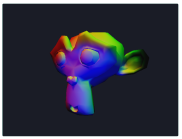
A disco

`lamps on circles: cos, sin`



Police lights

`flash = 0.5 + 0.5 * sin(8t)`



Red, green, blue spots

`red + green = yellow`



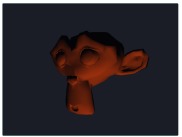
A cartoon Suzanne

`floor(diffuse * 3.0) / 3.0`



An X-ray ghost

`1.0 - abs(dot(n, v))`



A campfire

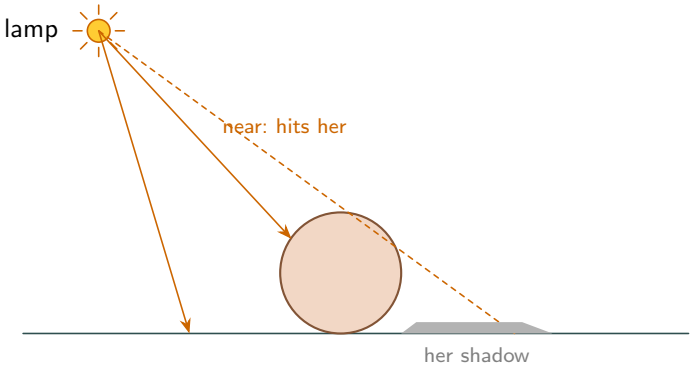
`a flickering orange lamp`

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

Summary

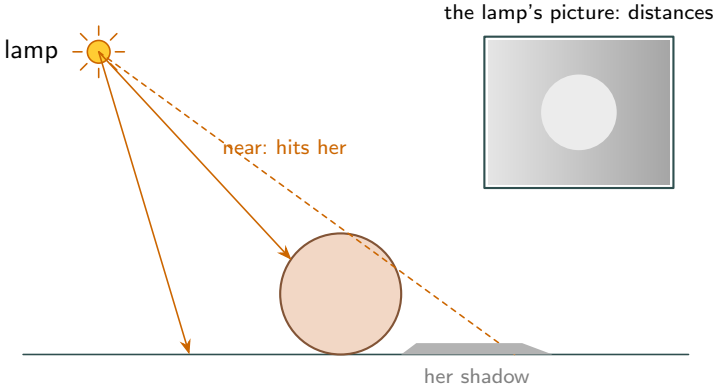
1 A model is a file	v, vn, f: corners, normals, triangles a corner: x y z and a normal; 2904 corners, draw(2904)
2 The normal turns	`u.model * vec4f(normal.x, normal.y, normal.z, 0.0)` $w = 0$: a direction turns but never moves
3 Blended, so normalize	`let n = normalize(normal);` smooth corner normals make flat triangles look round
4 Lamps add up	`c += colours[i] * (base * diffuse + shine)` $\mathbf{v}$ toward the real eye; key, fill and rim
5 A spotlight is a cone	`smoothstep(outer, inner, dot(-l, aim))` inside when $\cos\theta$ beats the cone's cosine; still no shadows

Next lecture: shadow maps



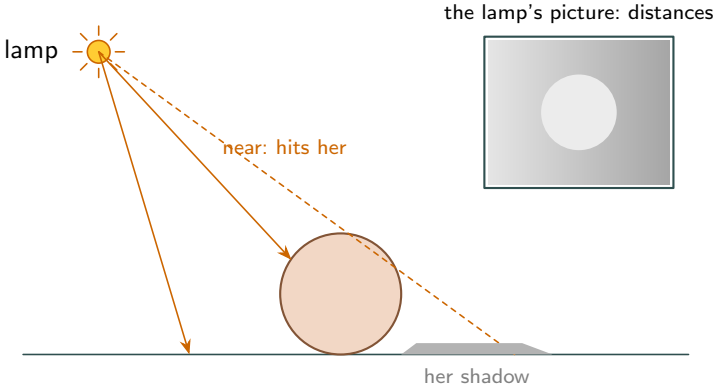
- Draw the scene once from the lamp, keeping only distances.

Next lecture: shadow maps



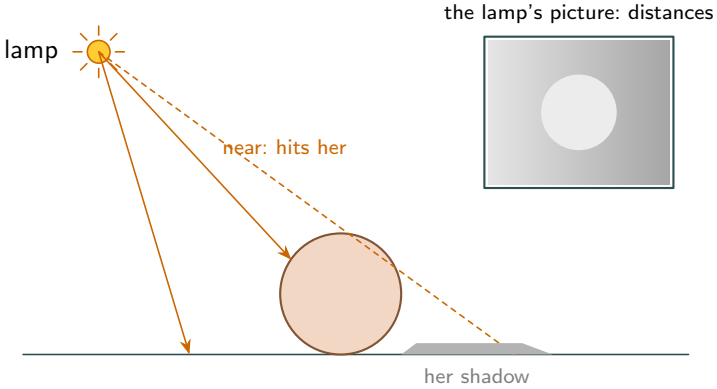
- Draw the scene once from the lamp, keeping only distances.
- Then each pixel asks: am I farther from the lamp than what it saw?

Next lecture: shadow maps



- Draw the scene once from the lamp, keeping only distances.
- Then each pixel asks: am I farther from the lamp than what it saw?
- If so, something blocks the lamp: shadow. A lookup, not a ray.

Next lecture: shadow maps



- Draw the scene once from the lamp, keeping only distances.
- Then each pixel asks: am I farther from the lamp than what it saw?
- If so, something blocks the lamp: shadow. A lookup, not a ray.
- Suzanne casts a shadow on the floor.

Exit Ticket

Today's question *(2 minutes)*

(a) A spotlight hangs at $(0, 5, 0)$, aimed straight down: aim $= (0, -1, 0)$. A point is at $P = (3, 1, 0)$. Find $-\mathbf{l}$, the unit direction from the spot to P.

(b) The cone is 30° wide on each side ($\cos 30^\circ = 0.87$). Is P lit by the spot?

CS 453 — Exit Ticket #29

Name: _____

Roll #: _____ Date: _____

Answer:

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