Lecture 32: The Rays Return, Part 2

A Mirror Floor, Soft Shadows and the Hybrid Frame

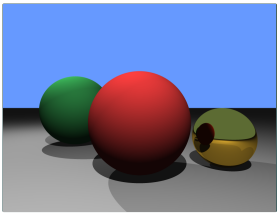
CS 453 — Computer Graphics

Fakhir Shaheen

Forman Christian University

Part A traced every ray; today two of them return

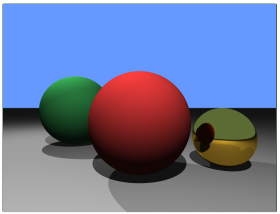
lecture 17: the mirror ray



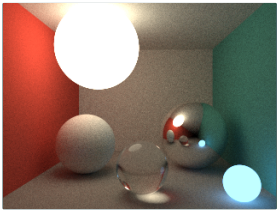
- Lecture 17: a mirror sends one more ray.

Part A traced every ray; today two of them return

lecture 17: the mirror ray



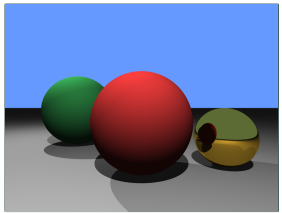
A7: a lamp with a size



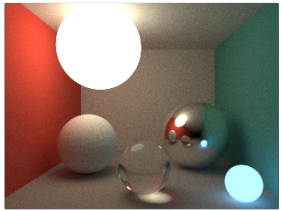
- Lecture 17: a mirror sends one more ray.
- A7: a lamp with a size, many random rays, averaged.

Part A traced every ray; today two of them return

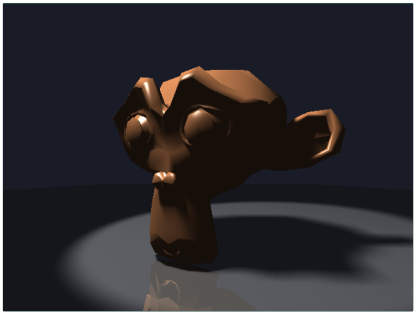
lecture 17: the mirror ray



A7: a lamp with a size



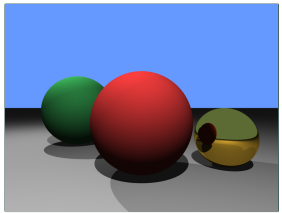
today: rasterized, plus both rays



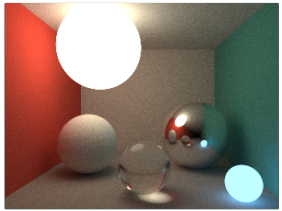
- Lecture 17: a mirror sends one more ray.
- A7: a lamp with a size, many random rays, averaged.
- Today: both rays, from fs_main, only where they count.

Part A traced every ray; today two of them return

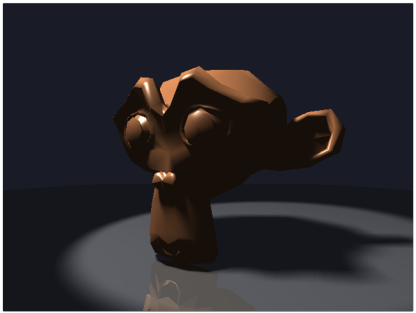
lecture 17: the mirror ray



A7: a lamp with a size



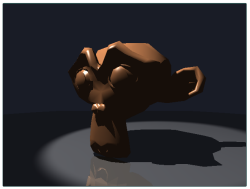
today: rasterized, plus both rays



- Lecture 17: a mirror sends one more ray.
- A7: a lamp with a size, many random rays, averaged.
- Today: both rays, from fs_main, only where they count.
- The rest of the frame stays rasterized.

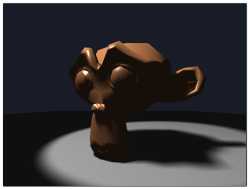
Today: three rounds of slides, then you try

1 **A mirror floor**



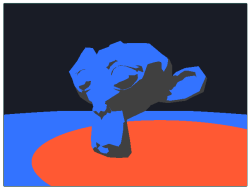
then You Try 1, 2

2 **Soft shadows**



then You Try 3, 4

3 **The hybrid frame**



then You Try 5

Open the Hybrid Lab now. It is on the course calendar, in lecture 32's row.

It is lecture 31's Ray Lab, plus a third pass that averages the frames.

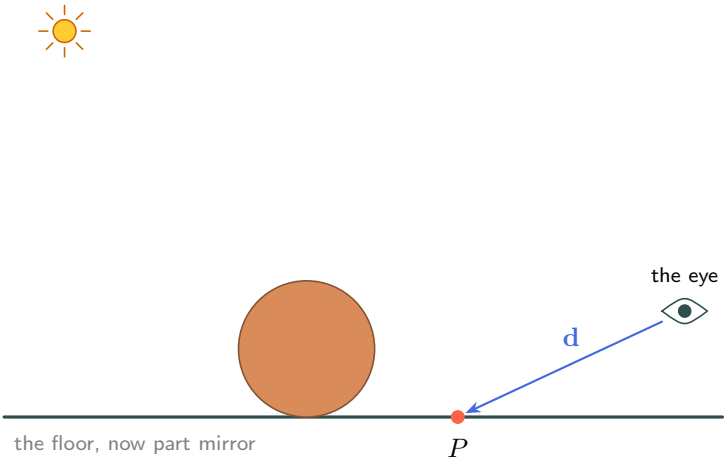
Watch `u.frame` count up, and the picture settle.

Round 1

A mirror floor

Lecture 17's mirror ray, sent only by the floor

What a mirror floor shows: lecture 17's mirror ray



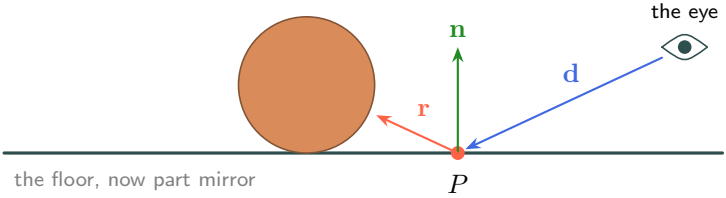
- The eye's ray $\mathbf{d}$ arrives at a floor point P.

What a mirror floor shows: lecture 17's mirror ray



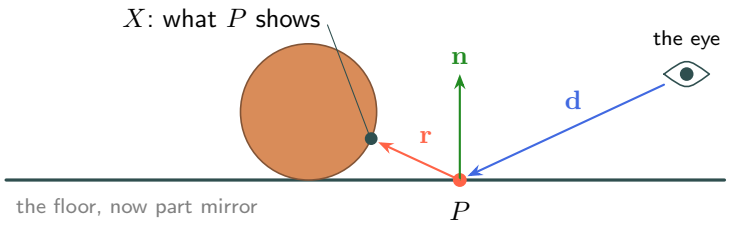
$$\mathbf{r} = \mathbf{d} - 2\,(\mathbf{d} \cdot \mathbf{n})\,\mathbf{n}$$

- The eye's ray $\mathbf{d}$ arrives at a floor point P.
- Lecture 17, slide 19: the mirror direction $\mathbf{r}$.



What a mirror floor shows: lecture 17's mirror ray

$$\mathbf{r} = \mathbf{d} - 2\,(\mathbf{d} \cdot \mathbf{n})\,\mathbf{n}$$

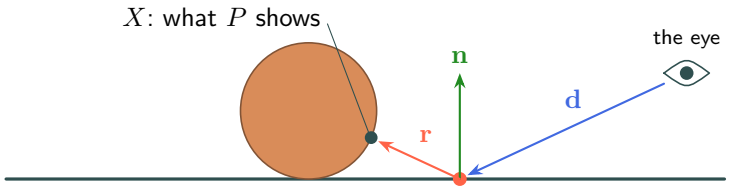


- The eye's ray $\mathbf{d}$ arrives at a floor point P.
- Lecture 17, slide 19: the mirror direction $\mathbf{r}$.
- The floor shows whatever $\mathbf{r}$ meets first: X on her.

What a mirror floor shows: lecture 17's mirror ray



$$\mathbf{r} = \mathbf{d} - 2\,(\mathbf{d} \cdot \mathbf{n})\,\mathbf{n}$$




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

```
let r = d - 2.0 * dot(d, n) * n;
```

- The eye's ray $\mathbf{d}$ arrives at a floor point P.
- Lecture 17, slide 19: the mirror direction $\mathbf{r}$.
- The floor shows whatever $\mathbf{r}$ meets first: X on her.
- fs_main knows P, $\mathbf{n}$ and `u.eye`: it can send $\mathbf{r}$.

Trace only what you must: only floor pixels send one



sends a mirror ray

sends none

- Her clay is not a mirror: no ray.

Trace only what you must: only floor pixels send one



sends a mirror ray

sends none

```
if (P.y < -0.99) {  // only the floor sends one
    ... the mirror ray ...
}
```

- Her clay is not a mirror: no ray.
- One `if`, as in lecture 31's skips.

Trace only what you must: only floor pixels send one



sends a mirror ray

sends none

```
if (P.y < -0.99) {  // only the floor sends one
    ... the mirror ray ...
}
```

113,302

floor pixels: mirror rays

- Her clay is not a mirror: no ray.
- One `if`, as in lecture 31's skips.
- About a third of the window.

Trace only what you must: only floor pixels send one



sends a mirror ray

sends none

```
if (P.y < -0.99) {  // only the floor sends one
    ... the mirror ray ...
}
```

113,302

floor pixels: mirror rays

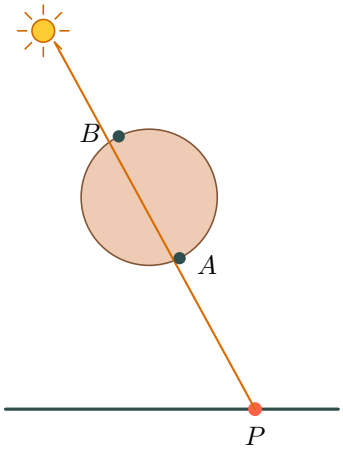
41,294

meet her ball:
only these loop

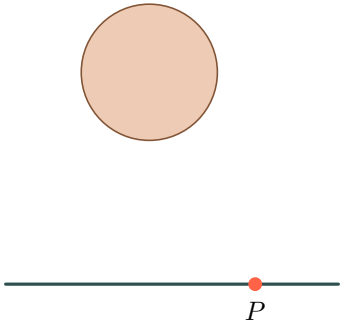
- Her clay is not a mirror: no ray.
- One `if`, as in lecture 31's skips.
- About a third of the window.
- Lecture 31's ball test works for any ray.

A shadow needs any hit; a mirror needs the nearest

a shadow ray (lecture 31)



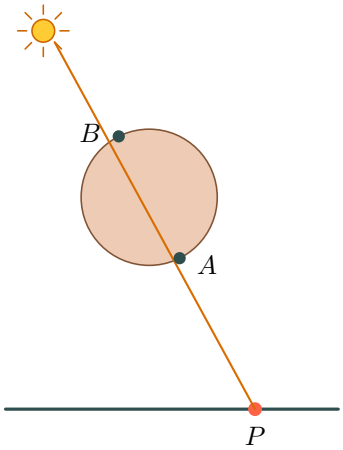
a mirror ray (today)



- One ray, two of her triangles in its way.

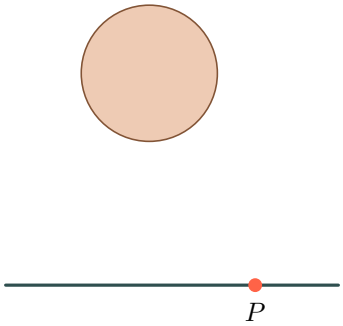
A shadow needs any hit; a mirror needs the nearest

a shadow ray (lecture 31)



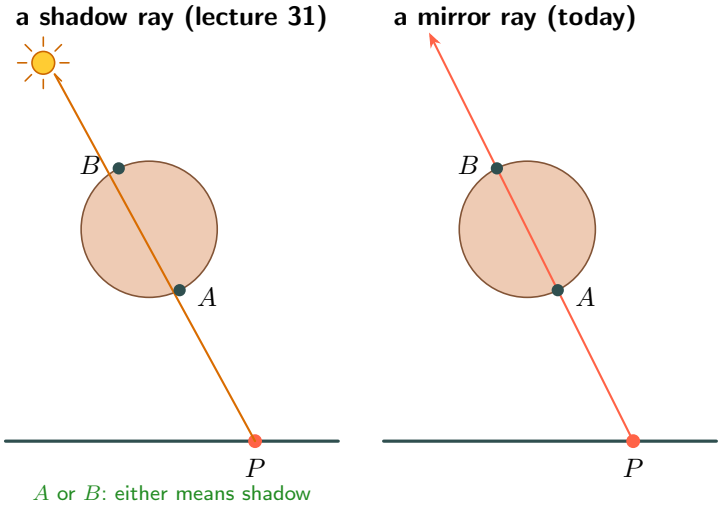
A or B: either means shadow

a mirror ray (today)



- One ray, two of her triangles in its way.
- A shadow: any hit will do. Stop at the first found.

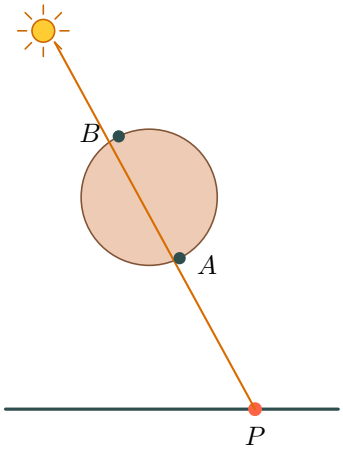
A shadow needs any hit; a mirror needs the nearest



- One ray, two of her triangles in its way.
- A shadow: any hit will do. Stop at the first found.
- A mirror shows one thing: the nearest hit.

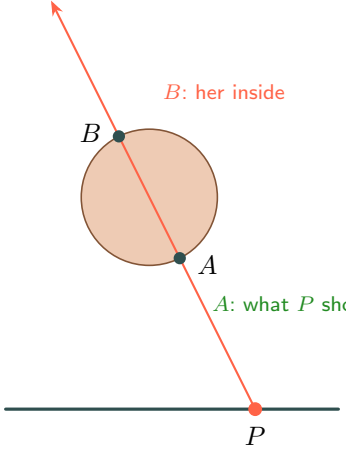
A shadow needs any hit; a mirror needs the nearest

a shadow ray (lecture 31)



A or B: either means shadow

a mirror ray (today)



the loop may meet B first

- One ray, two of her triangles in its way.
- A shadow: any hit will do. Stop at the first found.
- A mirror shows one thing: the nearest hit.
- The buffer's order is not the ray's order.

The nearest hit: after every hit, shrink far

the mirror ray from $P = (-0.4, -1, 1.15)$: the 968 triangles, in loop order



- Its ray meets four of her triangles.



The nearest hit: after every hit, shrink far

the mirror ray from $P = (-0.4, -1, 1.15)$: the 968 triangles, in loop order



- Its ray meets four of her triangles.



The nearest hit: after every hit, shrink far

the mirror ray from $P = (-0.4, -1, 1.15)$: the 968 triangles, in loop order



- Its ray meets four of her triangles.
- Each hit becomes the new far.



The nearest hit: after every hit, shrink far

the mirror ray from $P = (-0.4, -1, 1.15)$: the 968 triangles, in loop order



- Its ray meets four of her triangles.
- Each hit becomes the new far.
- Farther hits are refused at once.



The nearest hit: after every hit, shrink far

the mirror ray from $P = (-0.4, -1, 1.15)$: the 968 triangles, in loop order






```
let t = meet_triangle(P, r, 0.001, nearest, world(tri.a), ...);
if (t > 0.0) {     // nearer than every hit so far:
    nearest = t;   // from now on only nearer ones count
    hit = i;       // and remember which triangle
}
```

No break: the last one standing is the nearest. It sees triangle 239.

- Its ray meets four of her triangles.
- Each hit becomes the new far.
- Farther hits are refused at once.
- Lecture 15, slide 17: every hit shrinks t_1.

meet_triangle: lecture 31's test, answering where

```
fn meet_triangle(start: vec3f, dir: vec3f, eps: f32, far: f32,
                 a: vec3f, b: vec3f, c: vec3f) -> f32 {
    let n = cross(b - a, c - a);
    let d_dot_n = dot(dir, n);
    if (d_dot_n == 0.0) { return -1.0; }
    let t = dot(a - start, n) / d_dot_n;
    if (t < eps || t > far) { return -1.0; }
    let x = start + t * dir;
    if (dot(cross(b - a, x - a), n) < 0.0) { return -1.0; }
    if (dot(cross(c - b, x - b), n) < 0.0) { return -1.0; }
    if (dot(cross(a - c, x - c), n) < 0.0) { return -1.0; }
    return t;  // NEW: t, not true
}
```

- Lecture 31's two steps, line for line.

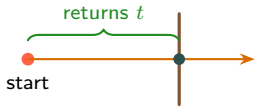
meet_triangle: lecture 31's test, answering where

```
fn meet_triangle(start: vec3f, dir: vec3f, eps: f32, far: f32,
                 a: vec3f, b: vec3f, c: vec3f) -> f32 {
    let n = cross(b - a, c - a);
    let d_dot_n = dot(dir, n);
    if (d_dot_n == 0.0) { return -1.0; }
    let t = dot(a - start, n) / d_dot_n;
    if (t < eps || t > far) { return -1.0; }
    let x = start + t * dir;
    if (dot(cross(b - a, x - a), n) < 0.0) { return -1.0; }
    if (dot(cross(c - b, x - b), n) < 0.0) { return -1.0; }
    if (dot(cross(a - c, x - c), n) < 0.0) { return -1.0; }
    return t;  // NEW: t, not true
}
```

- Lecture 31's two steps, line for line.
- No hit: -1.0. No real hit has a t below eps.

meet_triangle: lecture 31's test, answering where

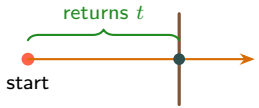
```
fn meet_triangle(start: vec3f, dir: vec3f, eps: f32, far: f32,
                 a: vec3f, b: vec3f, c: vec3f) -> f32 {
    let n = cross(b - a, c - a);
    let d_dot_n = dot(dir, n);
    if (d_dot_n == 0.0) { return -1.0; }
    let t = dot(a - start, n) / d_dot_n;
    if (t < eps || t > far) { return -1.0; }
    let x = start + t * dir;
    if (dot(cross(b - a, x - a), n) < 0.0) { return -1.0; }
    if (dot(cross(c - b, x - b), n) < 0.0) { return -1.0; }
    if (dot(cross(a - c, x - c), n) < 0.0) { return -1.0; }
    return t;  // NEW: t, not true
}
```



- Lecture 31's two steps, line for line.
- No hit: -1.0. No real hit has a t below eps.
- A hit: its t, so the loop can compare hits.

meet_triangle: lecture 31's test, answering where

```
fn meet_triangle(start: vec3f, dir: vec3f, eps: f32, far: f32,
                 a: vec3f, b: vec3f, c: vec3f) -> f32 {
    let n = cross(b - a, c - a);
    let d_dot_n = dot(dir, n);
    if (d_dot_n == 0.0) { return -1.0; }
    let t = dot(a - start, n) / d_dot_n;
    if (t < eps || t > far) { return -1.0; }
    let x = start + t * dir;
    if (dot(cross(b - a, x - a), n) < 0.0) { return -1.0; }
    if (dot(cross(c - b, x - b), n) < 0.0) { return -1.0; }
    if (dot(cross(a - c, x - c), n) < 0.0) { return -1.0; }
    return t;  // NEW: t, not true
}
```



- Lecture 31's two steps, line for line.
- No hit: -1.0. No real hit has a t below eps.
- A hit: its t, so the loop can compare hits.
- The shadow ray keeps lecture 31's hits_triangle.

The mirror ray, in WGSL

```
let shiny = 0.5;                    // how much of the floor is mirror
if (P.y < -0.99) {                // only the floor sends one
    let d = normalize(P - u.eye);
    let r = d - 2.0 * dot(d, n) * n;  // lecture 17
```

- Slide 5, in code.

```
}
```

▶ **In the Hybrid Lab:**
A mirror floor: the nearest hit

The mirror ray, in WGSL

```
let shiny = 0.5;                    // how much of the floor is mirror
if (P.y < -0.99) {                // only the floor sends one
    let d = normalize(P - u.eye);
    let r = d - 2.0 * dot(d, n) * n;  // lecture 17
    var seen = vec3f(0.10, 0.11, 0.15);  // a miss
```

- Slide 5, in code.
- No hit: the background.

```
}
```

▶ **In the Hybrid Lab:**
A mirror floor: the nearest hit

The mirror ray, in WGSL

```
let shiny = 0.5;                    // how much of the floor is mirror
if (P.y < -0.99) {                 // only the floor sends one
    let d = normalize(P - u.eye);
    let r = d - 2.0 * dot(d, n) * n;  // lecture 17
    var seen = vec3f(0.10, 0.11, 0.15);  // a miss
    if (meets_ball(P, r)) {           // lecture 31's ball
        var nearest = 1000.0;
        var hit = -1;
        for (var i = 0; i < 968; i++) {
            let tri = triangles[i];
            let t = meet_triangle(P, r, 0.001, nearest,
                        world(tri.a), world(tri.b), world(tri.c));
            if (t > 0.0) { nearest = t; hit = i; }
        }

    }

}
```

- Slide 5, in code.
- No hit: the background.
- The ball first; then all 968, no break.

▶ **In the Hybrid Lab:**
A mirror floor: the nearest hit

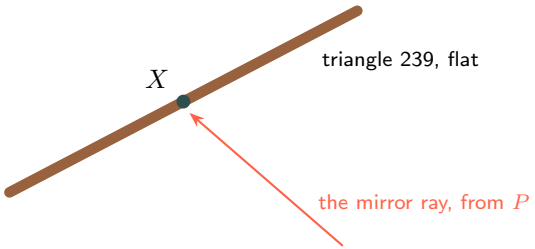
The mirror ray, in WGSL

```
let shiny = 0.5;                    // how much of the floor is mirror
if (P.y < -0.99) {                 // only the floor sends one
    let d = normalize(P - u.eye);
    let r = d - 2.0 * dot(d, n) * n;  // lecture 17
    var seen = vec3f(0.10, 0.11, 0.15);  // a miss
    if (meets_ball(P, r)) {           // lecture 31's ball
        var nearest = 1000.0;
        var hit = -1;
        for (var i = 0; i < 968; i++) {
            let tri = triangles[i];
            let t = meet_triangle(P, r, 0.001, nearest,
                        world(tri.a), world(tri.b), world(tri.c));
            if (t > 0.0) { nearest = t; hit = i; }
        }
        if (hit >= 0) { seen = ... (next slide) }
    }
    c = mix(c, seen, shiny);  // part floor, part mirror
}
```

- Slide 5, in code.
- No hit: the background.
- The ball first; then all 968, no break.
- Then mix it into the floor's colour.

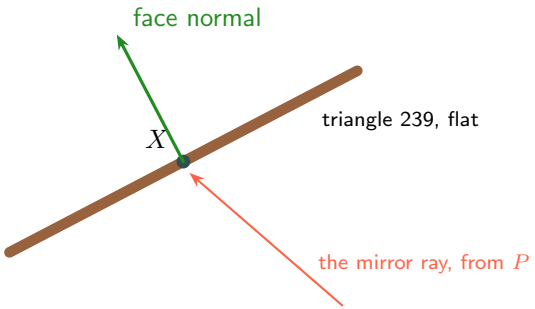
▶ **In the Hybrid Lab:**
A mirror floor: the nearest hit

What does the mirror see? Light X as fs_main lights her



- $X = P +$ nearest $\cdot\, \mathbf{r}$: the point P shows.

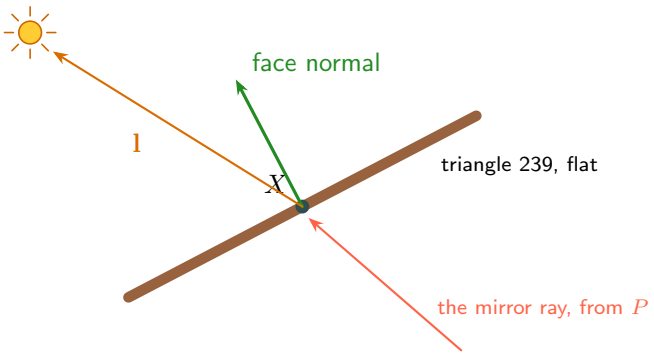
What does the mirror see? Light X as fs_main lights her




```
var hn = face_normal(triangles[hit]);
if (dot(hn, r) > 0.0) { hn = -hn; }  // face the ray
```

- $X = P +$ nearest $\cdot\, \mathbf{r}$: the point P shows.
- The buffer holds corners, no normals: use the flat one.

What does the mirror see? Light X as fs_main lights her

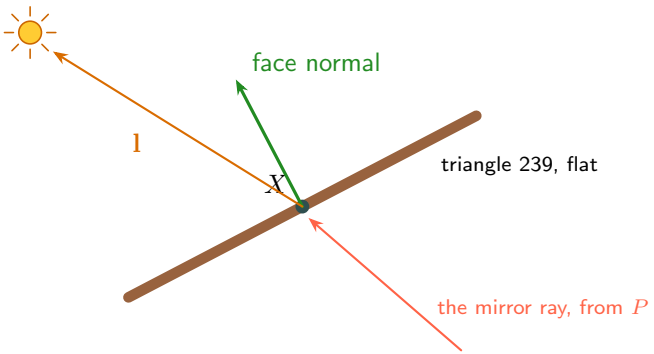


- $X = P +$ nearest $\cdot \mathbf{r}$: the point P shows.
- The buffer holds corners, no normals: use the flat one.
- Lecture 29's spot on X.

```
var hn = face_normal(triangles[hit]);
if (dot(hn, r) > 0.0) { hn = -hn; }  // face the ray
seen = vec3f(0.85, 0.55, 0.35) * (0.1 + spot_light(X, hn));
```

spot_light: lecture 29's cone times $\mathbf{n} \cdot \mathbf{l}$, as a function

What does the mirror see? Light X as fs_main lights her

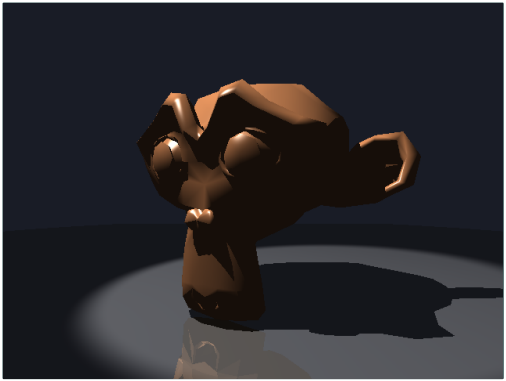



```
var hn = face_normal(triangles[hit]);
if (dot(hn, r) > 0.0) { hn = -hn; }  // face the ray
seen = vec3f(0.85, 0.55, 0.35) * (0.1 + spot_light(X, hn));
```

spot_light: lecture 29's cone times $\mathbf{n} \cdot \mathbf{l}$, as a function

- $X = P +$ nearest $\cdot\, \mathbf{r}$: the point P shows.
- The buffer holds corners, no normals: use the flat one.
- Lecture 29's spot on X.
- No shadow ray, no highlight at X: one bounce, then stop.

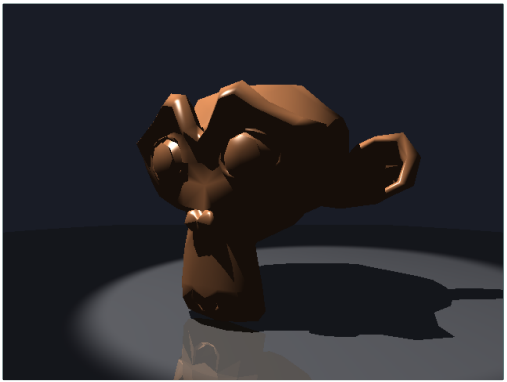
Part floor, part mirror: mix the two colours

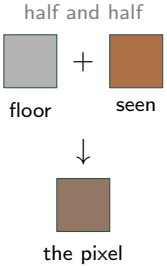


mirror.wgsl: lecture 31's hard shadow, plus the mirror ray

- Her reflection, under her chin.

Part floor, part mirror: mix the two colours



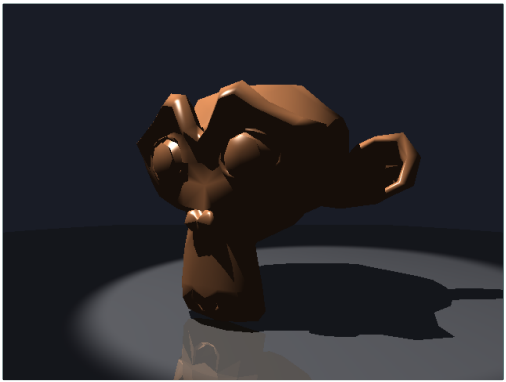


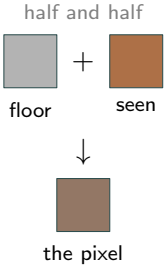
- Her reflection, under her chin.
- shiny $= 0.5$: half its own colour, half what it sees.

mirror.wgsl: lecture 31's hard shadow, plus the mirror ray

```
c = mix(c, seen, shiny);
```

Part floor, part mirror: mix the two colours



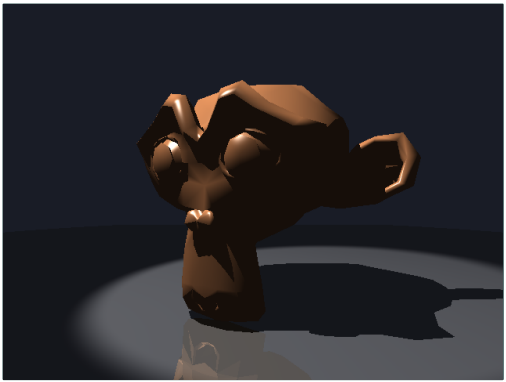


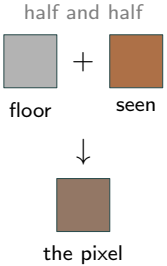
- Her reflection, under her chin.
- shiny $= 0.5$: half its own colour, half what it sees.
- Lecture 26's mix, with a colour from a ray.

mirror.wgsl: lecture 31's hard shadow, plus the mirror ray

```
c = mix(c, seen, shiny);
```

Part floor, part mirror: mix the two colours





mirror.wgsl: lecture 31's hard shadow, plus the mirror ray

```
c = mix(c, seen, shiny);
```

- Her reflection, under her chin.
- shiny $= 0.5$: half its own colour, half what it sees.
- Lecture 26's mix, with a colour from a ray.
- The dark floor shows her too: no lamp is needed to see a reflection.

Close up: what this reflection gets wrong



under her chin, rendered three times larger: her chin above, its reflection below

- The mirror image of her mouth and chin.

Close up: what this reflection gets wrong



under her chin, rendered three times larger: her chin above, its reflection below

- The mirror image of her mouth and chin.
- Flat facets: X has its triangle's own normal.

Close up: what this reflection gets wrong



under her chin, rendered three times larger: her chin above, its reflection below

- The mirror image of her mouth and chin.
- Flat facets: X has its triangle's own normal.
- Dim, no shadows inside it: X sends no rays.

Close up: what this reflection gets wrong



under her chin, rendered three times larger: her chin above, its reflection below

- The mirror image of her mouth and chin.
- Flat facets: X has its triangle's own normal.
- Dim, no shadows inside it: X sends no rays.
- Each fix costs another ray per pixel. We stop here.

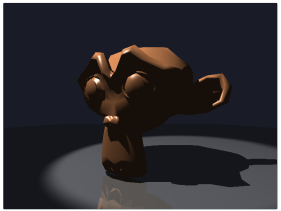
You Try 1: a perfect mirror

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

```
let shiny = 0.5;
if (P.y < -0.99) {
```

change this line

now



- **Set shiny to 1.0: the floor becomes all mirror.**
- Guess first: what happens to the spot's bright pool on the floor?

You Try 1: a perfect mirror

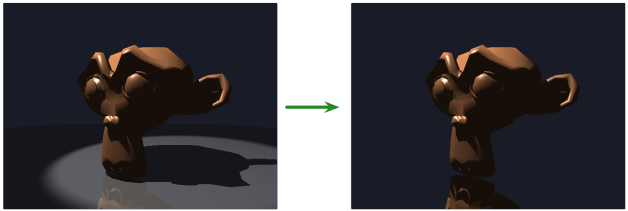
In the Hybrid Lab, choose **You Try 1**. Guess first, then change the code.

```
let shiny = 0.5;
if (P.y < -0.99) {
```

change this line

answer: `let shiny = 1.0;`

now



- **Set shiny to 1.0: the floor becomes all mirror.**
- Guess first: what happens to the spot's bright pool on the floor?
- It vanishes. A perfect mirror shows only what $\mathbf{r}$ meets (lecture 17, slide 27).

You Try 2: stop at the first hit

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

```
if (t > 0.0) {
    nearest = t;
    hit = i;
    // your line here
}
```

add a line here

- **Add a break: the loop stops at the first hit it finds.**
- Guess first: does the reflection change? Where?



now, close up under her chin

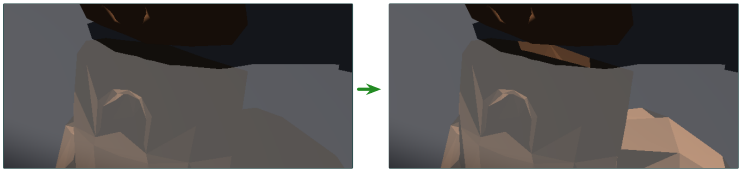
You Try 2: stop at the first hit

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

```
if (t > 0.0) {
    nearest = t;
    hit = i;
    // your line here
}
```

add a line here

answer: `break;`



now, close up under her chin

- **Add a break: the loop stops at the first hit it finds.**
- Guess first: does the reflection change? Where?
- Bright patches: triangles behind the nearest one, seen through her.

Round 2

Soft shadows

A lamp with a size: one random ray a frame, averaged

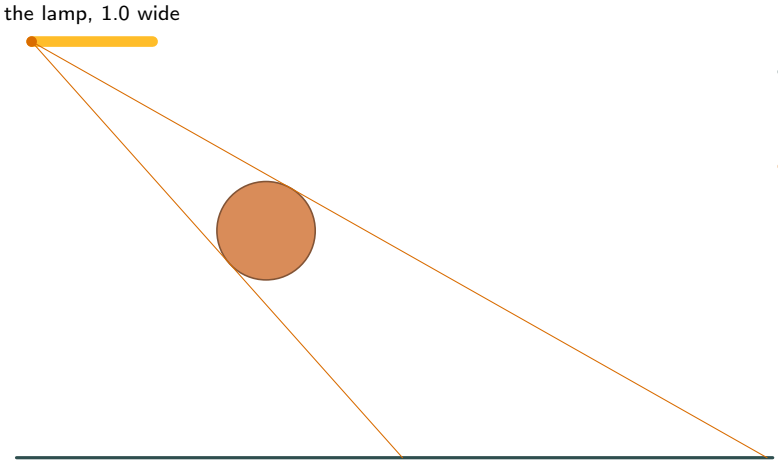
A lamp with a size: some points see part of it

the lamp, 1.0 wide



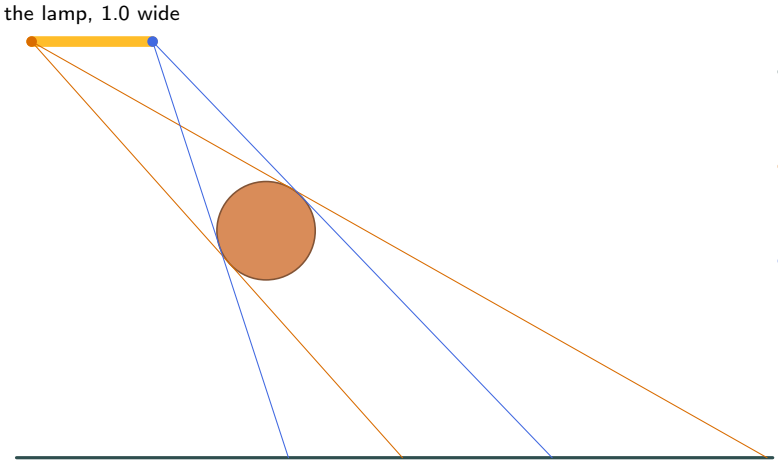
- Lecture 31's spot was a point: seen, or not.

A lamp with a size: some points see part of it



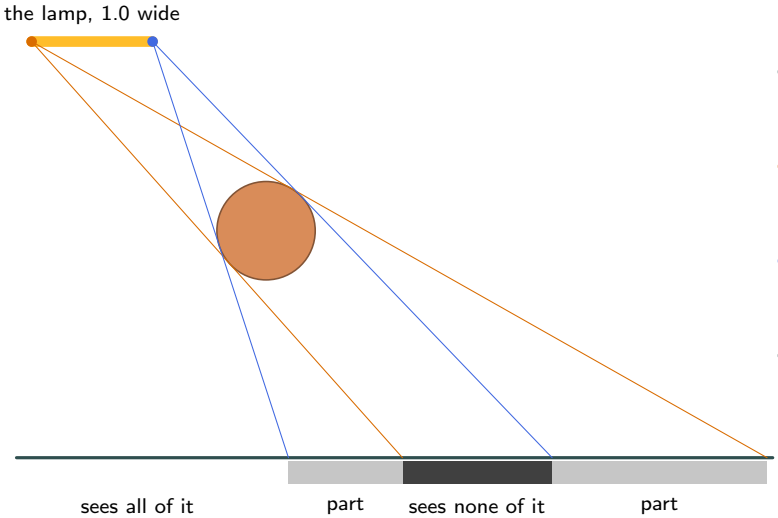
- Lecture 31's spot was a point: seen, or not.
- Each end of a lamp casts its own shadow.

A lamp with a size: some points see part of it



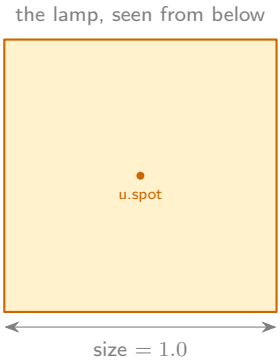
- Lecture 31's spot was a point: seen, or not.
- Each end of a lamp casts its own shadow.
- The two shadows do not line up.

A lamp with a size: some points see part of it



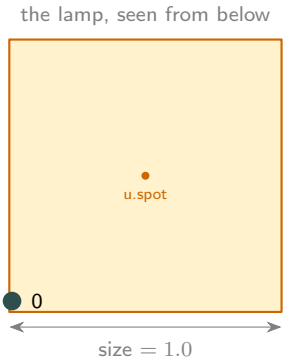
- Lecture 31's spot was a point: seen, or not.
- Each end of a lamp casts its own shadow.
- The two shadows do not line up.
- Between them, a soft edge. A7's lamps had a radius too.

One random point on the lamp, each frame



- The lamp: a square, 1.0 on a side, facing down.

One random point on the lamp, each frame

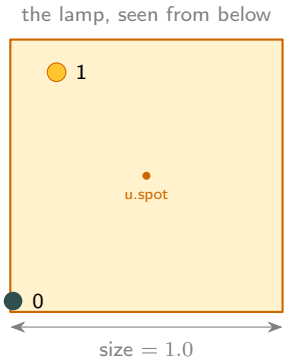


pixel (428, 330), frame by frame:

- its ray gets there: lit
- Suzanne is in the way

- The lamp: a square, 1.0 on a side, facing down.
- Each frame, a pixel aims at a new point on it.

One random point on the lamp, each frame

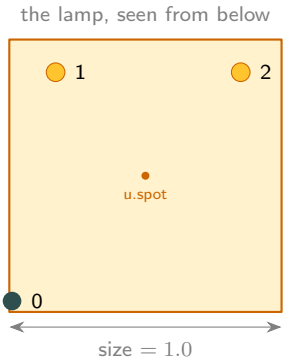


pixel (428, 330), frame by frame:

- its ray gets there: lit
- Suzanne is in the way

- The lamp: a square, 1.0 on a side, facing down.
- Each frame, a pixel aims at a new point on it.

One random point on the lamp, each frame

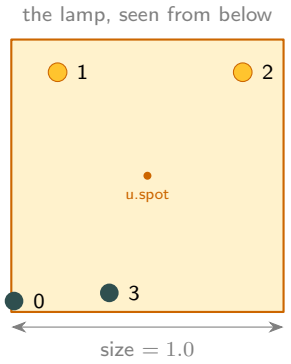


pixel (428, 330), frame by frame:

- its ray gets there: lit
- Suzanne is in the way

- The lamp: a square, 1.0 on a side, facing down.
- Each frame, a pixel aims at a new point on it.

One random point on the lamp, each frame

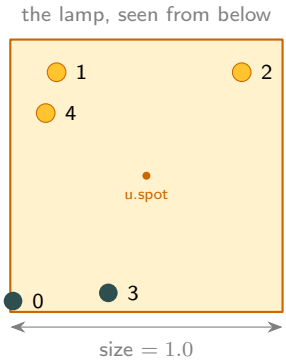


pixel (428, 330), frame by frame:

- its ray gets there: lit
- Suzanne is in the way

- The lamp: a square, 1.0 on a side, facing down.
- Each frame, a pixel aims at a new point on it.

One random point on the lamp, each frame

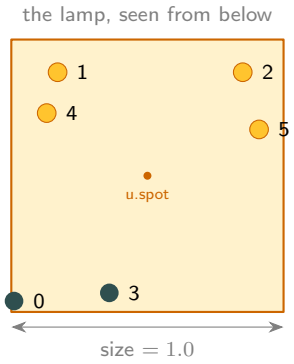


pixel (428, 330), frame by frame:

- its ray gets there: lit
- Suzanne is in the way

- The lamp: a square, 1.0 on a side, facing down.
- Each frame, a pixel aims at a new point on it.

One random point on the lamp, each frame

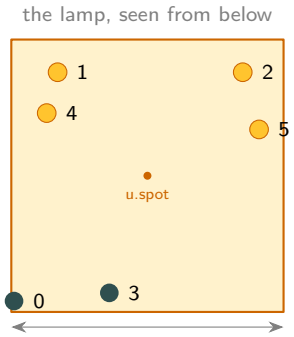


pixel (428, 330), frame by frame:

- its ray gets there: lit
- Suzanne is in the way

- The lamp: a square, 1.0 on a side, facing down.
- Each frame, a pixel aims at a new point on it.
- Lit: 0, 1, 1, 0, 1, 1. It sees 65% of the lamp.

One random point on the lamp, each frame



pixel (428, 330), frame by frame:

- its ray gets there: lit
- Suzanne is in the way

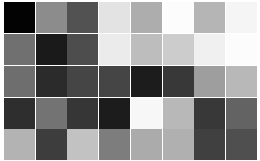
- The lamp: a square, 1.0 on a side, facing down.
- Each frame, a pixel aims at a new point on it.
- Lit: 0, 1, 1, 0, 1, 1. It sees 65% of the lamp.
- Still one shadow ray a pixel a frame.

```
let across = (random(seed) - 0.5) * size;      // -size/2 to size/2
let deep = (random(seed + 1u) - 0.5) * size;
let lamp_point = u.spot + vec3f(across, 0.0, deep);
let far = length(lamp_point - P);    let dir = (lamp_point - P) / far;
... then lecture 31's two skips and loop, toward dir ...
```

random(seed): a new number for every pixel and every frame

random(seed), 8×5 pixels from (428, 330): black 0, white 1

frame 0



```
let seed = u32(pos.x) * 1973u + u32(pos.y) * 9277u
           + u32(u.frame) * 26699u;
```

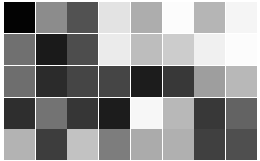
- The seed: from the pixel and the frame.

pos: the pixel, as in lecture 26. u.frame: frames since the last change (NEW in u).

random(seed): a new number for every pixel and every frame

random(seed), 8×5 pixels from (428, 330): black 0, white 1

frame 0



- The seed: from the pixel and the frame.
- random: the same seed, the same number, every time.

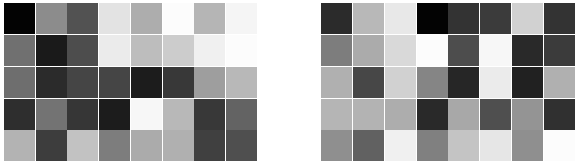
```
let seed = u32(pos.x) * 1973u + u32(pos.y) * 9277u
           + u32(u.frame) * 26699u;
fn random(seed: u32) -> f32 {  // a hash called PCG: given, no need to read inside
    var x = seed * 747796405u + 2891336453u;  ...
    return f32(x) / 4294967295.0;       // 0 to 1
}
```

pos: the pixel, as in lecture 26. u.frame: frames since the last change (NEW in u).

random(seed): a new number for every pixel and every frame

random(seed), 8 × 5 pixels from (428, 330): black 0, white 1

frame 0 frame 1

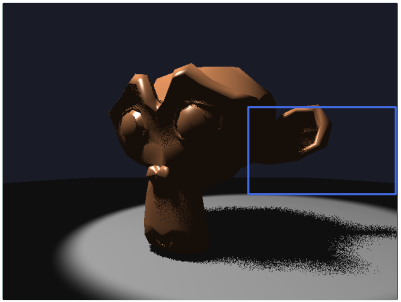


```
let seed = u32(pos.x) * 1973u + u32(pos.y) * 9277u
           + u32(u.frame) * 26699u;
fn random(seed: u32) -> f32 {  // a hash called PCG: given, no need to read inside
    var x = seed * 747796405u + 2891336453u;  ...
    return f32(x) / 4294967295.0;       // 0 to 1
}
```

- The seed: from the pixel and the frame.
- random: the same seed, the same number, every time.
- Next pixel or next frame: an unrelated number.

pos: the pixel, as in lecture 26. u.frame: frames since the last change (NEW in u).

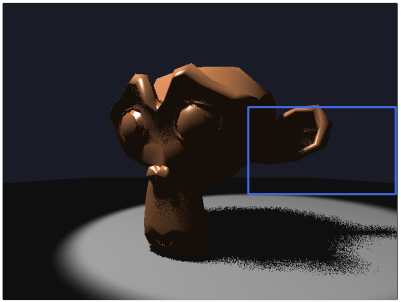
One frame: one ray a pixel, so the soft edge is grain



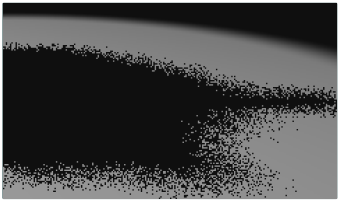
soft.wgsl, frame 0

- Each pixel, this frame: lit or not.

One frame: one ray a pixel, so the soft edge is grain



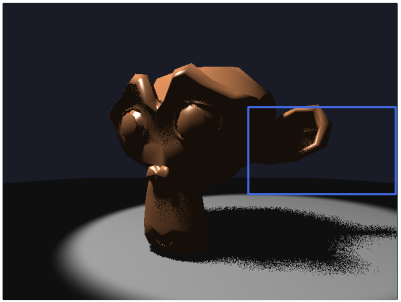
soft.wgsl, frame 0



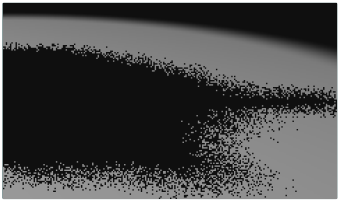
the blue box, twice as large

- Each pixel, this frame: lit or not.
- At the edge, neighbours disagree: grain.

One frame: one ray a pixel, so the soft edge is grain



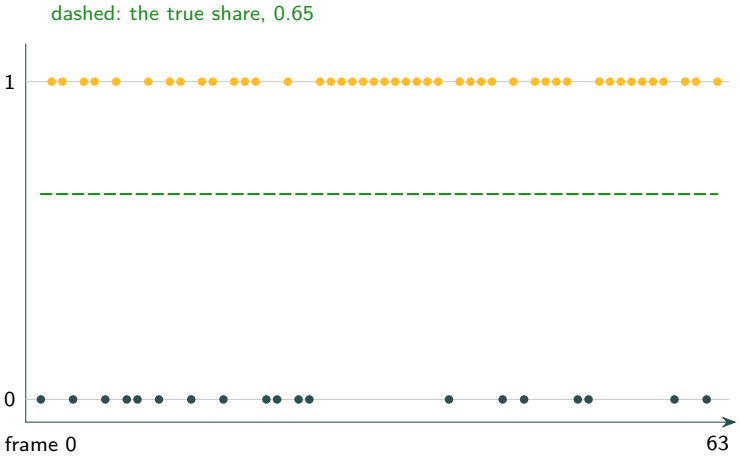
soft.wgsl, frame 0



the blue box, twice as large

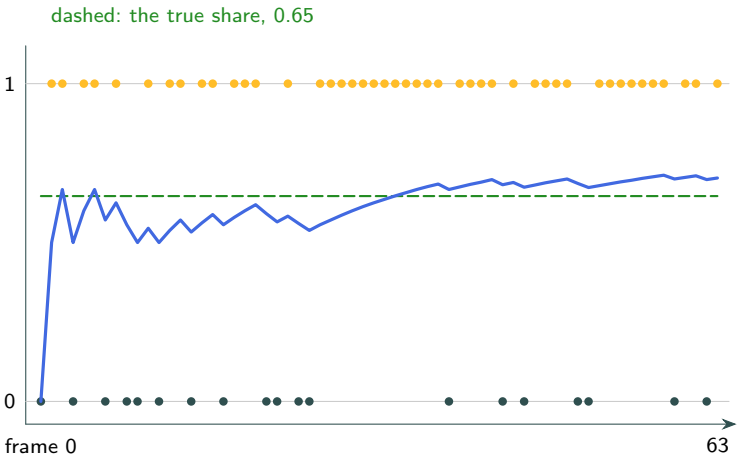
- Each pixel, this frame: lit or not.
- At the edge, neighbours disagree: grain.
- The cost is lecture 31's: one shadow ray a pixel.

One pixel, 64 frames: the average settles



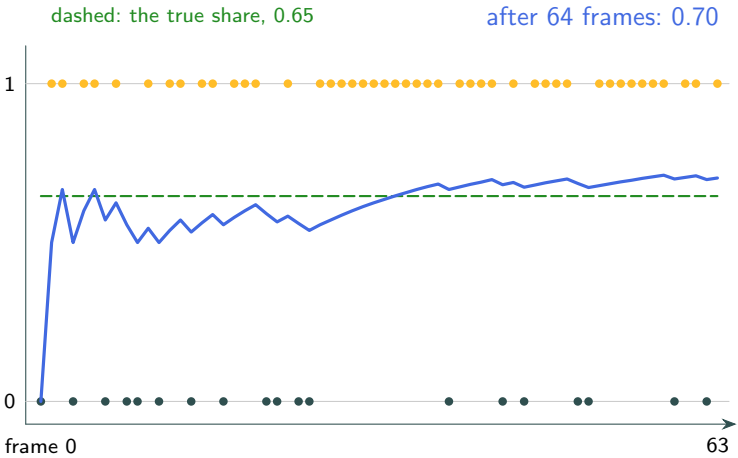
- Dots: pixel (428, 330), lit or not, each frame.

One pixel, 64 frames: the average settles



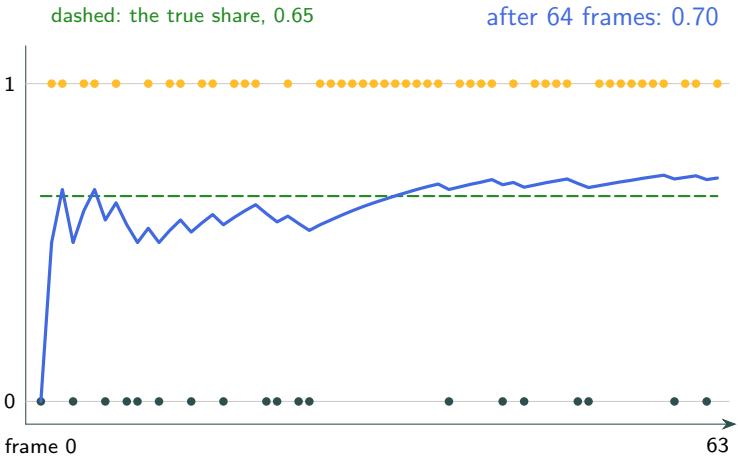
- Dots: pixel (428, 330), lit or not, each frame.
- Line: the average of every frame so far.

One pixel, 64 frames: the average settles



- Dots: pixel (428, 330), lit or not, each frame.
- Line: the average of every frame so far.
- It wanders, then settles near the true share.

One pixel, 64 frames: the average settles



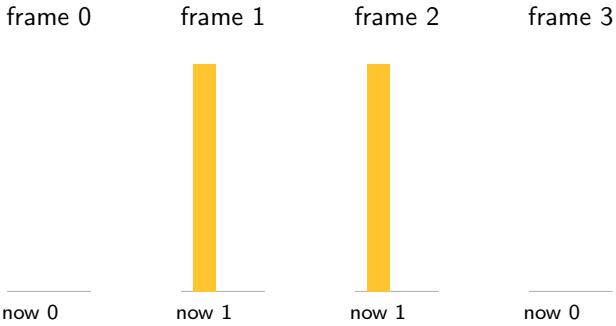
- Dots: pixel (428, 330), lit or not, each frame.
- Line: the average of every frame so far.
- It wanders, then settles near the true share.
- A7's estimator: an average of random rays.

The running average: one number a pixel, not every frame

avg $\leftarrow$ mix(avg, now, $\frac{1}{n+1}$) $\quad n$: `u.frame`

Slide 21's first four frames: lit 0, 1, 1, 0. **Try first:** each average.

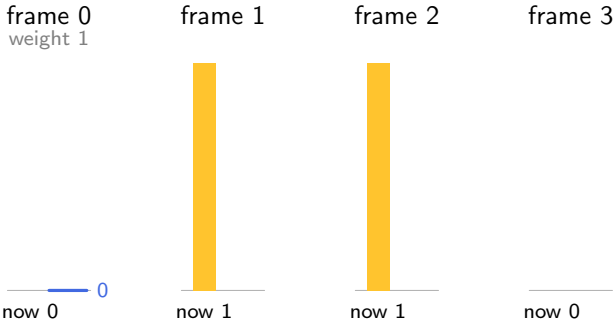
- Frame 0: weight 1, the average is that frame.



The running average: one number a pixel, not every frame

avg $\leftarrow$ mix(avg, now, $\frac{1}{n+1}$) n: `u.frame`

Slide 21's first four frames: lit 0, 1, 1, 0. **Try first:** each average.

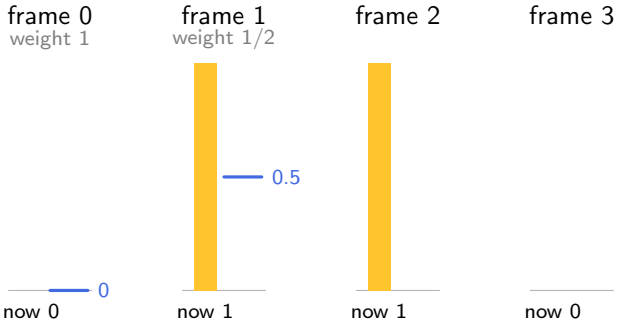


- Frame 0: weight 1, the average is that frame.

The running average: one number a pixel, not every frame

avg $\leftarrow$ mix(avg, now, $\frac{1}{n+1}$) $\quad n$: `u.frame`

Slide 21's first four frames: lit 0, 1, 1, 0. **Try first:** each average.

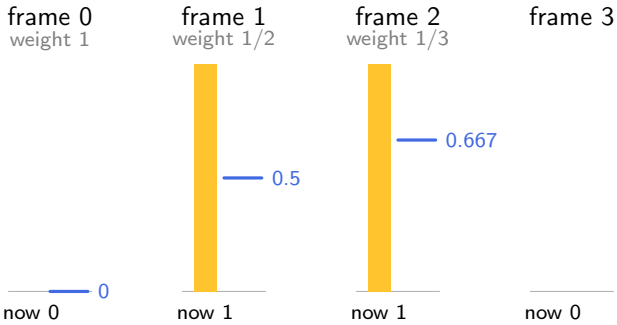


- Frame 0: weight 1, the average is that frame.
- mix(0, 1, 1/2) = 0.5.

The running average: one number a pixel, not every frame

avg $\leftarrow$ mix(avg, now, $\frac{1}{n+1}$) n: `u.frame`

Slide 21's first four frames: lit 0, 1, 1, 0. **Try first:** each average.

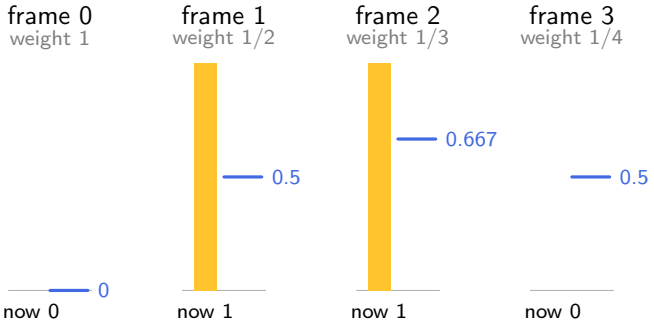


- Frame 0: weight 1, the average is that frame.
- mix(0, 1, 1/2) = 0.5.
- mix(0.5, 1, 1/3) = 0.667.

The running average: one number a pixel, not every frame

avg $\leftarrow$ mix(avg, now, $\frac{1}{n+1}$) $\quad n$: `u.frame`

Slide 21's first four frames: lit 0, 1, 1, 0. **Try first:** each average.

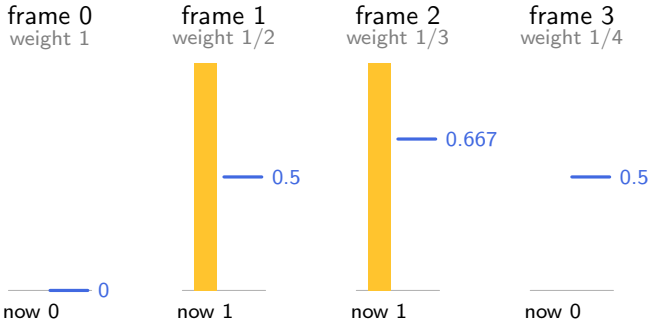


- Frame 0: weight 1, the average is that frame.
- mix(0, 1, 1/2) = 0.5.
- mix(0.5, 1, 1/3) = 0.667.
- mix(0.667, 0, 1/4) = 0.5.

The running average: one number a pixel, not every frame

avg $\leftarrow$ mix(avg, now, $\frac{1}{n+1}$) $\quad n$: `u.frame`

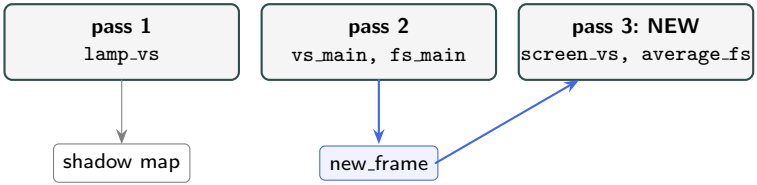
Slide 21's first four frames: lit 0, 1, 1, 0. **Try first:** each average.



- Frame 0: weight 1, the average is that frame.
- mix(0, 1, 1/2) = 0.5.
- mix(0.5, 1, 1/3) = 0.667.
- mix(0.667, 0, 1/4) = 0.5.

Check: $(0+1+1+0)/4 = 0.5$. The same as averaging a list.

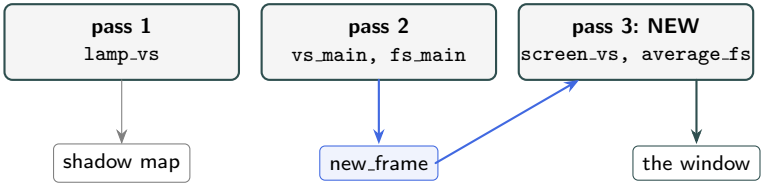
Pass 3: average_fs keeps the average in history



- Pass 2 now draws into a texture, new_frame.

▶ **In the Hybrid Lab:**
Soft shadows: a random ray a frame, averaged

Pass 3: average_fs keeps the average in history



- Pass 2 now draws into a texture, new_frame.
- Pass 3: average_fs runs once a pixel.

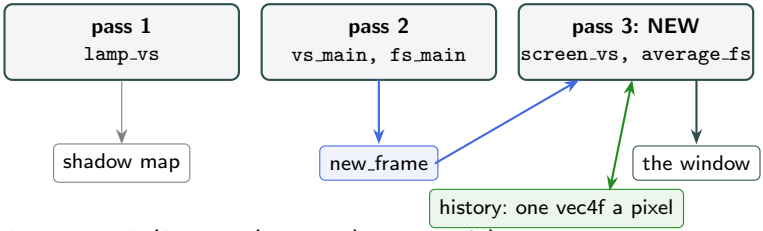
```
fn average_fs(@builtin(position) pos: vec4f)
        -> @location(0) vec4f {
    let x = i32(pos.x);
    let y = i32(pos.y);
    let i = y * i32(u.resolution.x) + x;      // its place
    let now = textureLoad(new_frame, vec2i(x, y), 0);



}
```

▶ **In the Hybrid Lab:**
Soft shadows: a random ray a frame, averaged

Pass 3: average_fs keeps the average in history

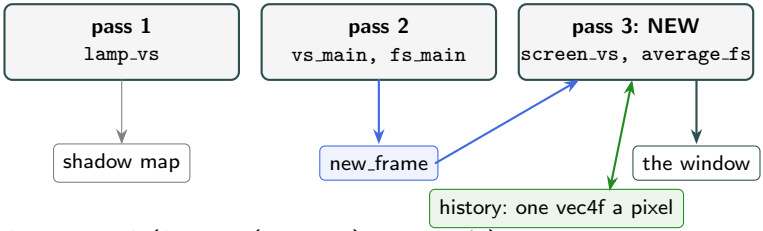



```
fn average_fs(@builtin(position) pos: vec4f)
        -> @location(0) vec4f {
    let x = i32(pos.x);
    let y = i32(pos.y);
    let i = y * i32(u.resolution.x) + x;       // its place
    let now = textureLoad(new_frame, vec2i(x, y), 0);
    let weight = 1.0 / (u.frame + 1.0);
    history[i] = mix(history[i], now, weight);  // slide 22
    return history[i];
}
```

- Pass 2 now draws into a texture, new_frame.
- Pass 3: average_fs runs once a pixel.
- history stays from frame to frame.

▶ **In the Hybrid Lab:**
Soft shadows: a random ray a frame, averaged

Pass 3: average_fs keeps the average in history




```
fn average_fs(@builtin(position) pos: vec4f)
        -> @location(0) vec4f {
    let x = i32(pos.x);
    let y = i32(pos.y);
    let i = y * i32(u.resolution.x) + x;       // its place
    let now = textureLoad(new_frame, vec2i(x, y), 0);
    let weight = 1.0 / (u.frame + 1.0);
    history[i] = mix(history[i], now, weight);  // slide 22
    return history[i];
}
```

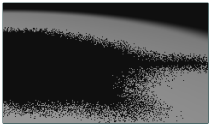
screen_vs, given: one triangle that covers the window.

- Pass 2 now draws into a texture, new_frame.
- Pass 3: average_fs runs once a pixel.
- history stays from frame to frame.
- Anything changes: u.frame back to 0.

▶ **In the Hybrid Lab:**
Soft shadows: a random ray a frame, averaged

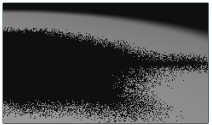
Four times the frames, half the grain

1 frame

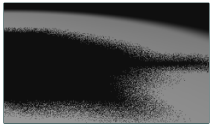


Four times the frames, half the grain

1 frame

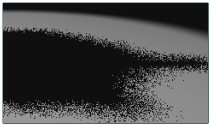


4 frames

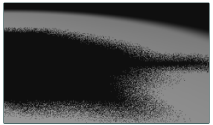


Four times the frames, half the grain

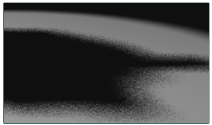
1 frame



4 frames

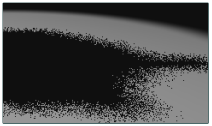


16 frames

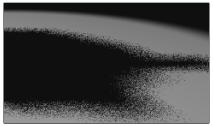


Four times the frames, half the grain

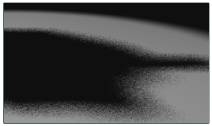
1 frame



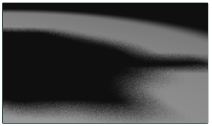
4 frames



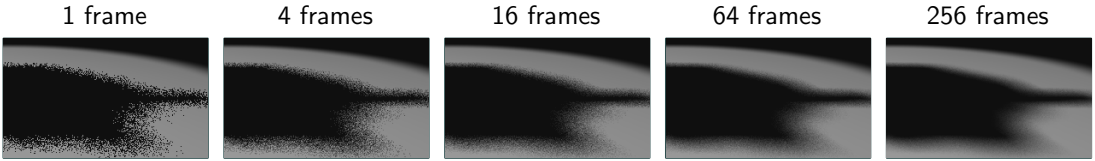
16 frames



64 frames



Four times the frames, half the grain



A7: the grain falls as $1/\sqrt{N}$: four times the samples, half the grain.

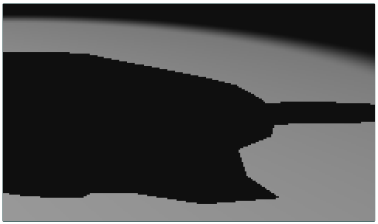
Today: the same rule, with frames for samples.

The cost of a frame never changes: one shadow ray a pixel.

At 60 frames a second, 256 frames take about four seconds, if nothing moves.

Hard against soft: lecture 31's point, today's square

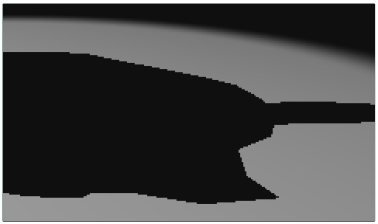
lecture 31: a point



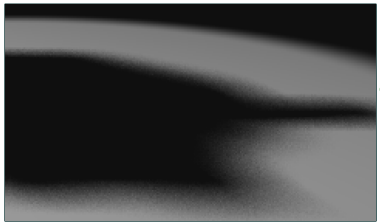
- One ray to one point: all or nothing.

Hard against soft: lecture 31's point, today's square

lecture 31: a point



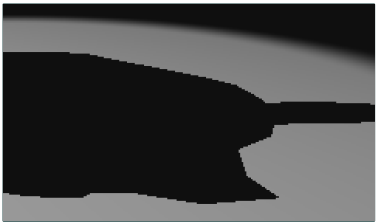
today: a square, 256 frames



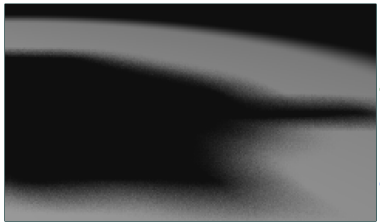
- One ray to one point: all or nothing.
- Many points, averaged: a share of the lamp.

Hard against soft: lecture 31's point, today's square

lecture 31: a point



today: a square, 256 frames

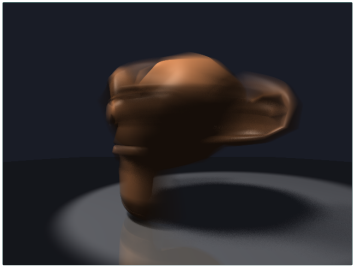


Near her (left): still sharp. The ear's tip is far from its shadow (right): wide and soft.

- One ray to one point: all or nothing.
- Many points, averaged: a share of the lamp.
- Real shadows do this: sharp at the contact, soft far away.

The artefact: when she moves, the average remembers

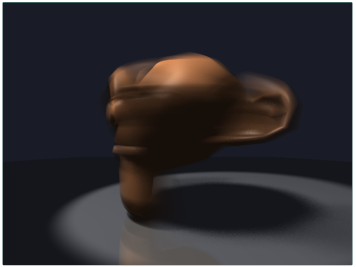
She turns 1 radian a second, 60 frames a second; one second later:



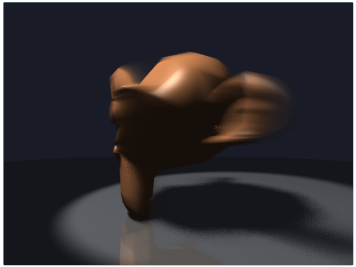
average every frame:
a smear

The artefact: when she moves, the average remembers

She turns 1 radian a second, 60 frames a second; one second later:



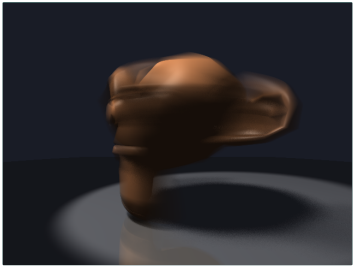
average every frame:
a smear



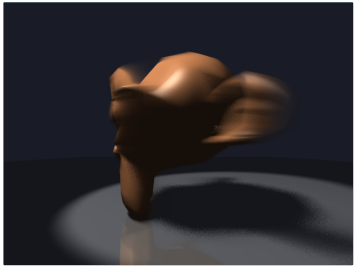
weight 0.1, the last 10 or so:
a shorter trail

The artefact: when she moves, the average remembers

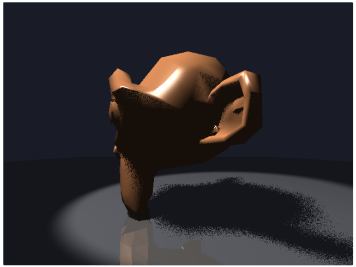
She turns 1 radian a second, 60 frames a second; one second later:



average every frame:
a smear



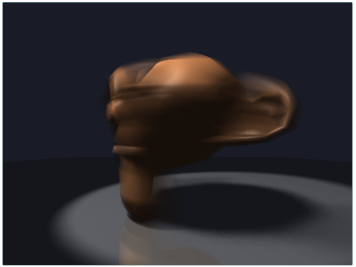
weight 0.1, the last 10 or so:
a shorter trail



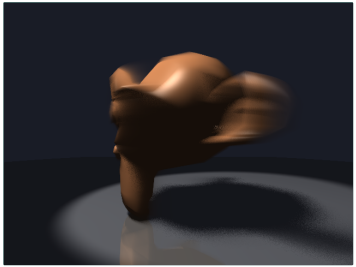
start again every frame:
sharp, but grainy

The artefact: when she moves, the average remembers

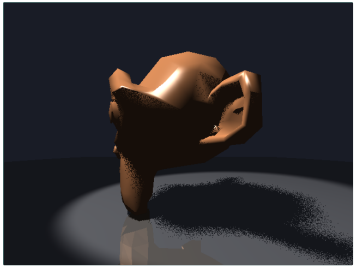
She turns 1 radian a second, 60 frames a second; one second later:



average every frame:
a smear



weight 0.1, the last 10 or so:
a shorter trail



start again every frame:
sharp, but grainy

hybrid.py starts again whenever she turns. Games **reproject**: they find where each pixel was last frame and keep only that part of the average. Then a **denoiser** hides the grain.

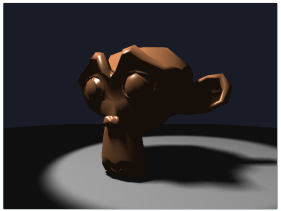
You Try 3: a bigger lamp

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

```
let size = 1.0;
let seed = u32(pos.x) * 1973u + ...;
```

change this line

now



- **Set size to 3.0: a square lamp 3 on a side.**
- Guess first: the soft edge? The grain in the first frames?

You Try 3: a bigger lamp

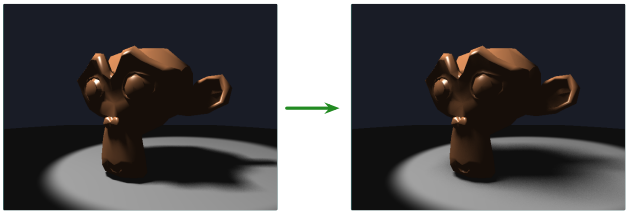
In the Hybrid Lab, choose **You Try 3**. Guess first, then change the code.

```
let size = 1.0;
let seed = u32(pos.x) * 1973u + ...;
```

change this line

answer: `let size = 3.0;`

now



- **Set size to 3.0: a square lamp 3 on a side.**
- Guess first: the soft edge? The grain in the first frames?
- A wider, softer edge. More grain at first: more pixels disagree, so it needs more frames.

You Try 4: keep only this frame

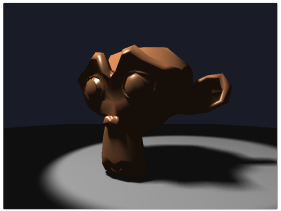
In the Hybrid Lab, choose **You Try 4**. Guess first, then change the code.

```
let weight = 1.0 / (u.frame + 1.0);
history[i] = mix(history[i], now, weight);
```

change this line

- **In average_fs, set weight to 1.0.**
- Guess first: what does the shadow's edge look like now?

now



You Try 4: keep only this frame

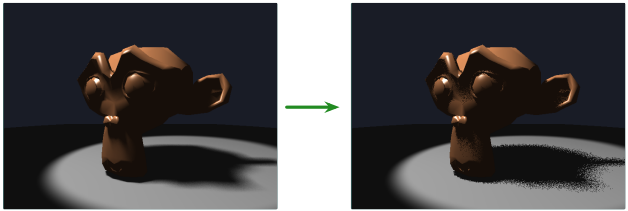
In the Hybrid Lab, choose **You Try 4**. Guess first, then change the code.

```
let weight = 1.0 / (u.frame + 1.0);
history[i] = mix(history[i], now, weight);
```

change this line

answer: `let weight = 1.0;`

now



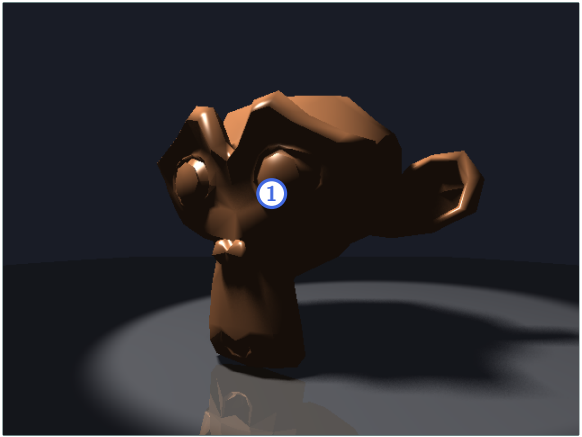
- **In average_fs, set weight to 1.0.**
- Guess first: what does the shadow's edge look like now?
- One frame's grain, new every frame: it shimmers. Nothing is averaged.

Round 3

The hybrid frame

Rasterize what you can, trace only what you must

The hybrid frame: raster, shadow rays, mirror rays

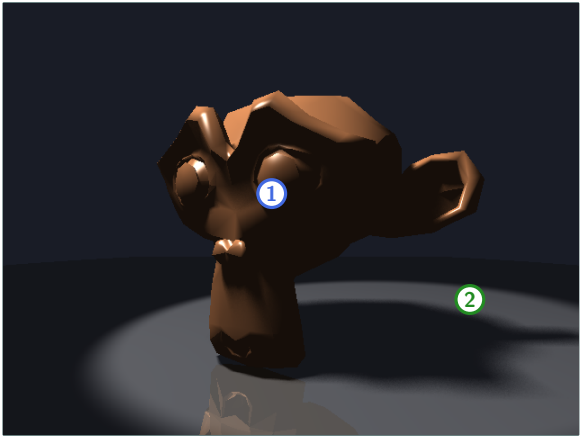


hybrid.wgsl, 256 frames

- **1** Rasterized: lecture 28's pipeline, lecture 29's light.

▶ **In the Hybrid Lab:**
The hybrid frame

The hybrid frame: raster, shadow rays, mirror rays



hybrid.wgsl, 256 frames

- **1** Rasterized: lecture 28's pipeline, lecture 29's light.
- **2** Soft shadows: one shadow ray a pixel, averaged.

▶ **In the Hybrid Lab:**
The hybrid frame

The hybrid frame: raster, shadow rays, mirror rays



hybrid.wgsl, 256 frames

- **1** Rasterized: lecture 28's pipeline, lecture 29's light.
- **2** Soft shadows: one shadow ray a pixel, averaged.
- **3** The reflection: one mirror ray from each floor pixel.

▶ **In the Hybrid Lab:**
The hybrid frame

The hybrid frame: raster, shadow rays, mirror rays

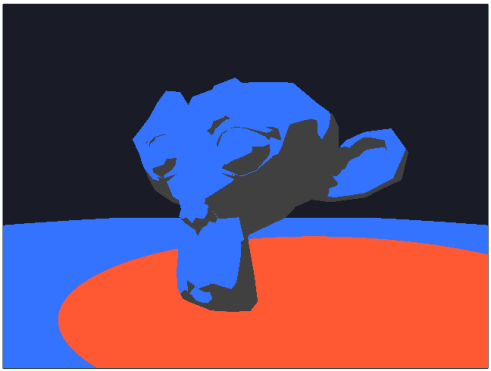


hybrid.wgsl, 256 frames

- **1** Rasterized: lecture 28's pipeline, lecture 29's light.
- **2** Soft shadows: one shadow ray a pixel, averaged.
- **3** The reflection: one mirror ray from each floor pixel.
- Nothing else is traced.

▶ **In the Hybrid Lab:**
The hybrid frame

Rays per pixel: 0, 1 or 2

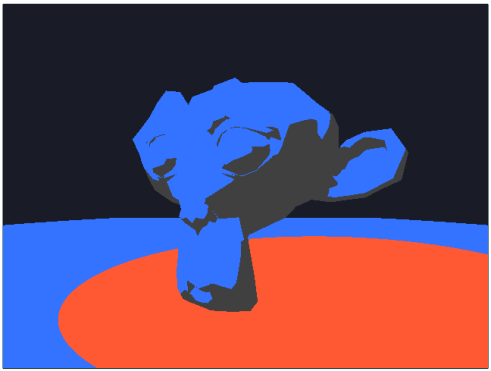


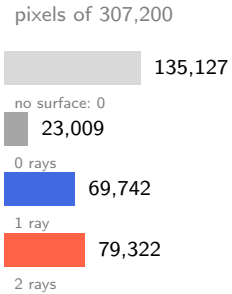
- 0: her dark side
- 1: a shadow ray, or a mirror ray
- 2: a lit floor point sends both

- count.wgsl: lecture 31's rule and today's, nothing traced.

▶ In the Hybrid Lab:
Rays per pixel: grey 0, blue 1, red 2

Rays per pixel: 0, 1 or 2



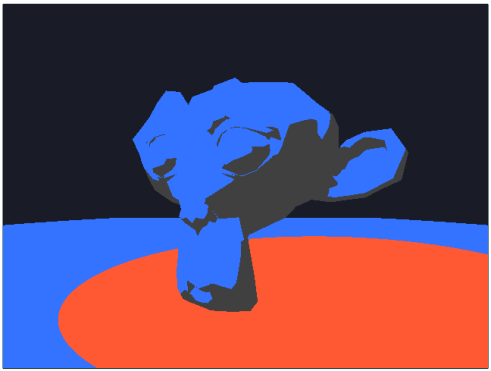


- count.wgsl: lecture 31's rule and today's, nothing traced.
- Read back from the picture.

0: her dark side
1: a shadow ray, or a mirror ray
2: a lit floor point sends both

▶ **In the Hybrid Lab:**
Rays per pixel: grey 0, blue 1, red 2

Rays per pixel: 0, 1 or 2



0: her dark side
1: a shadow ray, or a mirror ray
2: a lit floor point sends both

pixels of 307,200

no surface: 0



0 rays



1 ray



2 rays

228,386

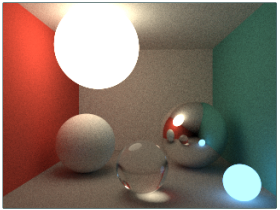
rays a frame: 0.74 a pixel

- count.wgsl: lecture 31's rule and today's, nothing traced.
- Read back from the picture.
- Never more than 2 rays a pixel a frame.

▶ **In the Hybrid Lab:**
Rays per pixel: grey 0, blue 1, red 2

Part A against the hybrid: how many rays a pixel?

A7's path tracer

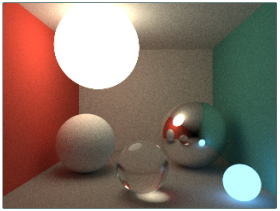


every ray traced: the first hit,
every bounce, up to 10;
512 samples a pixel;
about four minutes (PyPy)

- A7: up to 5,120 rays a pixel, every bounce.

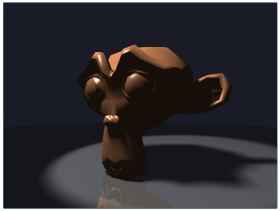
Part A against the hybrid: how many rays a pixel?

A7's path tracer



every ray traced: the first hit,
every bounce, up to 10;
512 samples a pixel;
about four minutes (PyPy)

today's hybrid

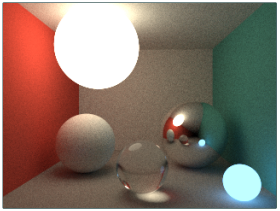


the first hit rasterized;
one shadow ray, one mirror ray;
at most 2 rays a pixel a frame;
averaged over frames

- A7: up to 5,120 rays a pixel, every bounce.
- Today: 0.74 a pixel a frame.

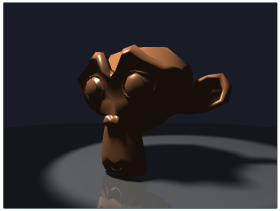
Part A against the hybrid: how many rays a pixel?

A7's path tracer



every ray traced: the first hit,
every bounce, up to 10;
512 samples a pixel;
about four minutes (PyPy)

today's hybrid

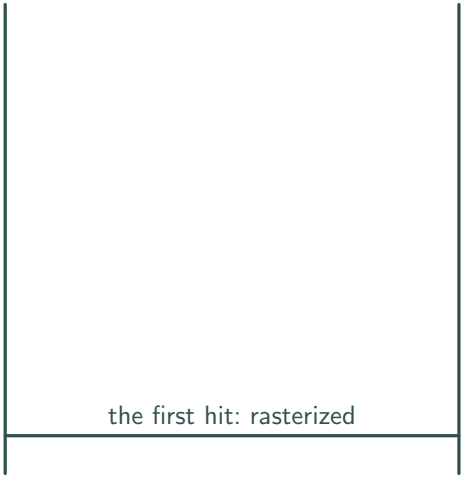


the first hit rasterized;
one shadow ray, one mirror ray;
at most 2 rays a pixel a frame;
averaged over frames

- A7: up to 5,120 rays a pixel, every bounce.
- Today: 0.74 a pixel a frame.
- Today traces only the rays you can see best: shadows and mirrors.

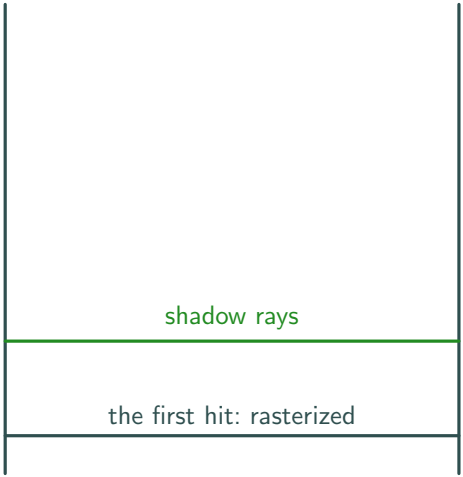
The same idea in both: an average of random rays.

Which rays do real-time engines trace?



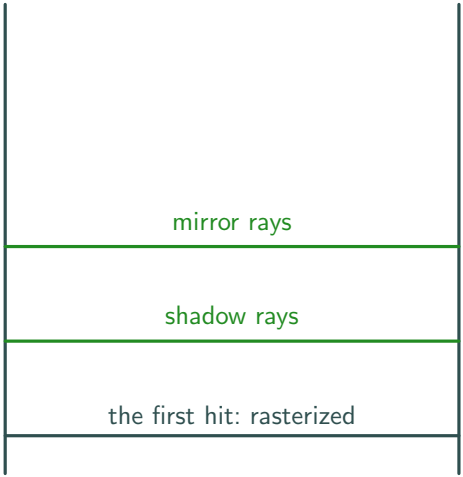
lectures 24 to 29

Which rays do real-time engines trace?



- A game's "ray tracing" setting: the low rungs, as today.

Which rays do real-time engines trace?



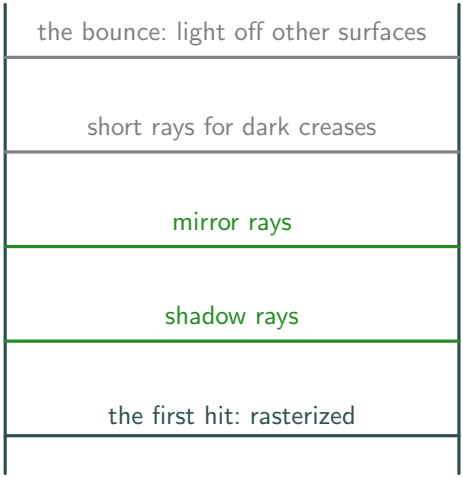
- A game's "ray tracing" setting: the low rungs, as today.

Which rays do real-time engines trace?



- A game's "ray tracing" setting: the low rungs, as today.
- Its "path tracing" setting: every rung, as in A7.

Which rays do real-time engines trace?



- A game's "ray tracing" setting: the low rungs, as today.
- Its "path tracing" setting: every rung, as in A7.
- The rays return one at a time, the most visible first.

You Try 5: let Suzanne send a mirror ray too

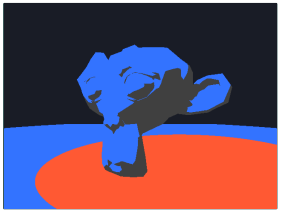
In the Hybrid Lab, choose **You Try 5**. Guess first, then change the code.

```
if (P.y < -0.99) {     // on the floor:
    rays += 1;         // one mirror ray
```

change this line

- **Every point sends a mirror ray: change the floor test.**
- Guess first: which colours change, and where?

now



You Try 5: let Suzanne send a mirror ray too

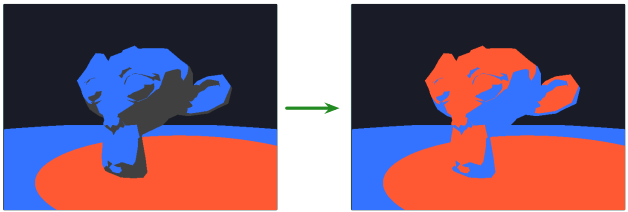
In the Hybrid Lab, choose **You Try 5**. Guess first, then change the code.

```
if (P.y < -0.99) {     // on the floor:
    rays += 1;         // one mirror ray
```

change this line

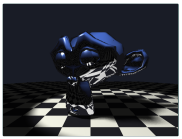
answer: `if (true) {`

now



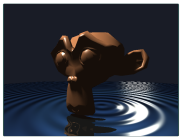
- **Every point sends a mirror ray: change the floor test.**
- Guess first: which colours change, and where?
- Her lit side turns red, her dark side blue: 287,157 rays a frame, 26% more.

Now make your own picture



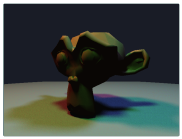
A chrome Suzanne

`r meets the floor plane`



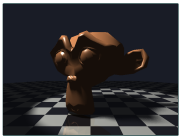
A pond

`ripples tilt the normal`



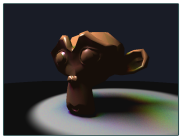
Three coloured lamps

`one lamp at random`



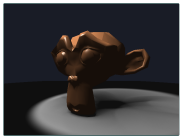
Whitted's floor, 1980

`mix(checker, seen, 0.35)`



A rainbow lamp

`hue from the lamp point`



A long exposure

`the lamp circles`

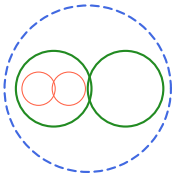
All six are in the Hybrid Lab, under **Ideas**. Change one. Mix two. Show your neighbour.

Summary

1	The mirror ray	lecture 17's $\mathbf{r}$, sent only by floor pixels the nearest hit: every hit shrinks far; no break
2	Soft shadows	a lamp with a size: one random point a pixel a frame random(seed) from the pixel and `u.frame`
3	The running average	pass 3: history $\leftarrow$ mix(history, now, $1/(n+1)$) four times the frames, half the grain; motion smears
4	The hybrid frame	rasterize the first hit; trace shadows and mirrors at most 2 rays a pixel a frame, 0.74 on average
5	Part A returns	the same rays as A7, far fewer of them as the budget grows, the rays return one at a time

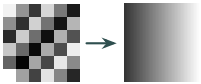
Where the rays go next

a BVH in hardware



RTX cards walk lecture 31's tree

a denoiser



one frame, cleaned up, in place of 256

every rung traced



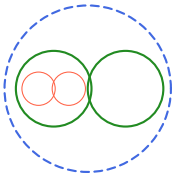
A7, at 60 frames a second

- Lecture 31, slide 22: the BVH, now built into the chip.

To read on: *Real-Time Rendering*, 4th edition, chapter 26, "Real-Time Ray Tracing" (free online); *Ray Tracing Gems*, free from its authors.

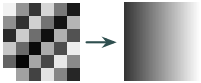
Where the rays go next

a BVH in hardware



RTX cards walk lecture 31's tree

a denoiser



one frame, cleaned up, in place of 256

every rung traced



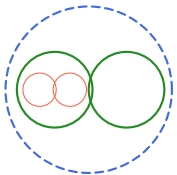
A7, at 60 frames a second

- Lecture 31, slide 22: the BVH, now built into the chip.
- A blur that respects edges, guided by depth and normals.

To read on: *Real-Time Rendering*, 4th edition, chapter 26, "Real-Time Ray Tracing" (free online); *Ray Tracing Gems*, free from its authors.

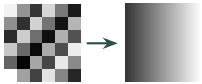
Where the rays go next

a BVH in hardware



RTX cards walk lecture 31's tree

a denoiser



one frame, cleaned up, in place of 256

every rung traced



A7, at 60 frames a second

- Lecture 31, slide 22: the BVH, now built into the chip.
- A blur that respects edges, guided by depth and normals.
- Path tracing in real time.

To read on: *Real-Time Rendering*, 4th edition, chapter 26, "Real-Time Ray Tracing" (free online); *Ray Tracing Gems*, free from its authors.

Exit Ticket

Today's question *(2 minutes)*

(a) The eye's ray reaches the floor with $\mathbf{d} = (0.6, -0.8, 0)$; the floor's normal is $\mathbf{n} = (0, 1, 0)$. Find the mirror direction $\mathbf{r}$.

(b) A pixel's shadow ray answers 1, 0, 0, 1, 1 in frames 0 to 4. What does the running average show after each frame?

CS 453 — Exit Ticket #32

Name: _____

Roll #: _____ Date: _____

Answer:

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