Lecture 34: Ray Marching 2

Shapes that Melt, Repeat and Cast Shadows

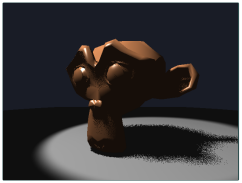
CS 453 — Computer Graphics

Fakhir Shaheen

Forman Christian University

Lecture 32's soft shadow, from one march

lecture 32: one frame

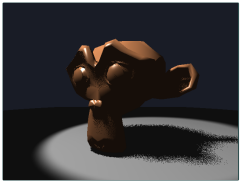


one random point
on the lamp: grain

- Lecture 32 (slide 20): a random point on the lamp in each frame.

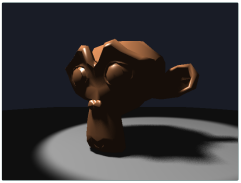
Lecture 32's soft shadow, from one march

lecture 32: one frame



one random point
on the lamp: grain

lecture 32: 64 frames

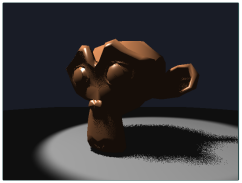


the average of
64 frames: soft

- Lecture 32 (slide 20): a random point on the lamp in each frame.
- Soft only after 64 frames were averaged (slide 24).

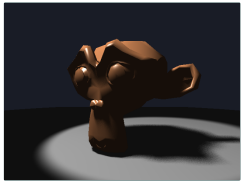
Lecture 32's soft shadow, from one march

lecture 32: one frame



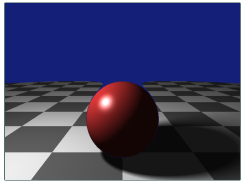
one random point on the lamp: grain

lecture 32: 64 frames



the average of 64 frames: soft

today: one frame

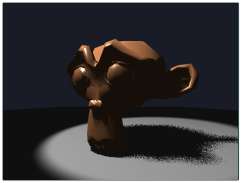


one march toward the lamp: soft at once

- Lecture 32 (slide 20): a random point on the lamp in each frame.
- Soft only after 64 frames were averaged (slide 24).
- Today: march from the point toward the lamp. Soft in one frame.

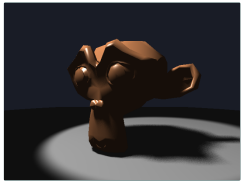
Lecture 32's soft shadow, from one march

lecture 32: one frame



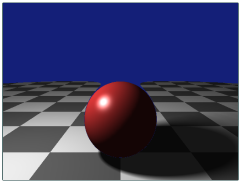
one random point on the lamp: grain

lecture 32: 64 frames



the average of 64 frames: soft

today: one frame



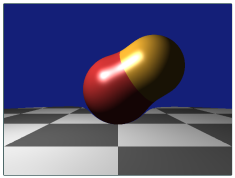
one march toward the lamp: soft at once

- Lecture 32 (slide 20): a random point on the lamp in each frame.
- Soft only after 64 frames were averaged (slide 24).
- Today: march from the point toward the lamp. Soft in one frame.

Measured: the march asks 20.2 distances a pixel; the shadow asks 7.7 more.

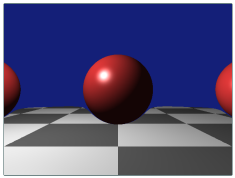
Today: three rounds of slides, then you try

1 **Combine and melt**



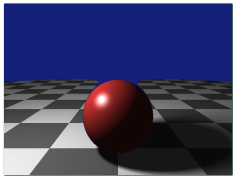
then You Try 1, 2

2 **Bend the space**



then You Try 3, 4

3 **Shadows from the march**



then You Try 5, 6

Open the March Lab now. It is on the course calendar, in lecture 34's row.

A laptop or a phone will do. Work in pairs: one screen for two people is enough.

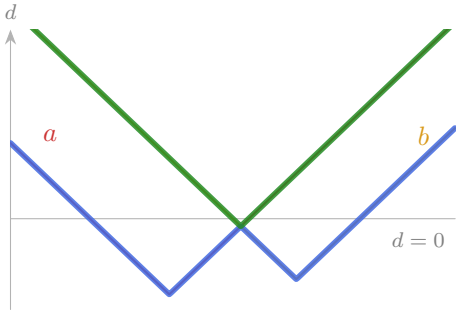
Every shader starts from lecture 33's: the same ball, floor, march and light.

Round 1

Combine and melt

two distances, one shape

min keeps both shapes; max keeps only the overlap



balls

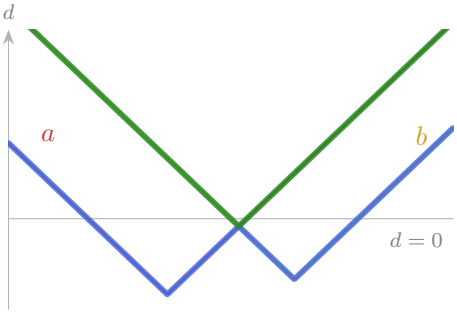
the distances along one line through both balls

- Along one line, each ball's distance is a V: below 0 inside.

lecture 33's slice: white is $d = 0$

▶ **Run it in the March Lab:**
Slice: min keeps both

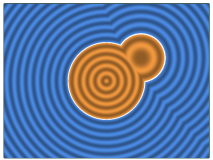
min keeps both shapes; max keeps only the overlap





the distances along one line through both balls

min: both balls

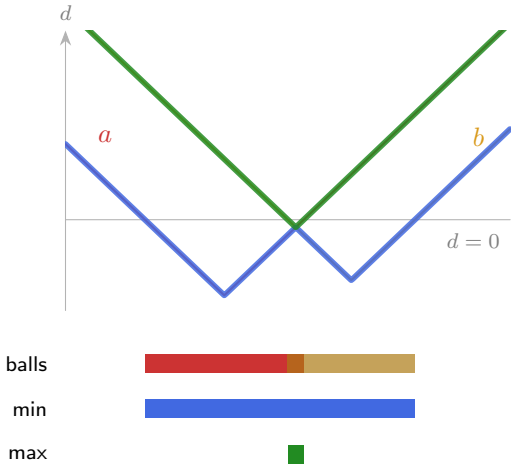


- Along one line, each ball's distance is a V: below 0 inside.
- min: the lower one. Below 0 if either is: both balls (lecture 33).

lecture 33's slice: white is $d = 0$

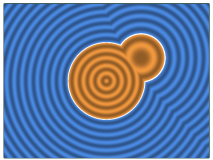
▶ **Run it in the March Lab:**
Slice: min keeps both

min keeps both shapes; max keeps only the overlap

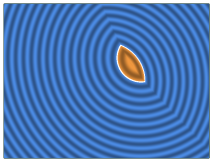


the distances along one line through both balls

min: both balls



max: the overlap

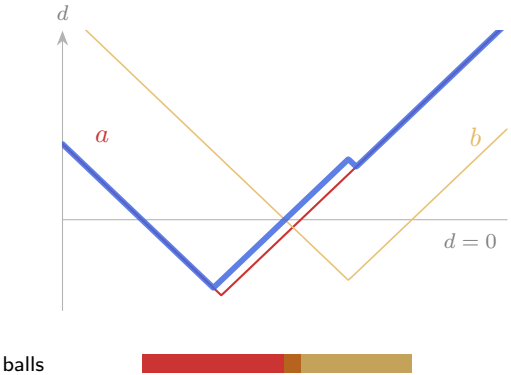


lecture 33's slice: white is $d = 0$

- Along one line, each ball's distance is a V: below 0 inside.
- min: the lower one. Below 0 if either is: both balls (lecture 33).
- max: the higher one. Below 0 only if both are: the overlap.

▶ **Run it in the March Lab:**
Slice: min keeps both

Cut one shape out of another: `max(a, -b)`

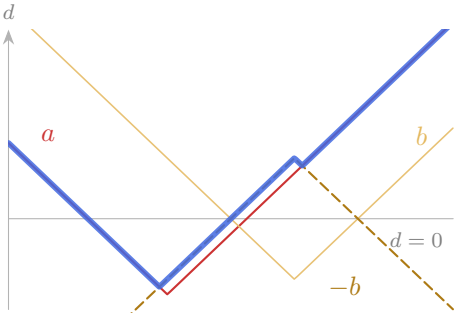


the same line, the same two balls

- To cut, we want: inside a, and **outside** b.

▶ **Run it in the March Lab:**
Slice: max(a, -b) cuts

Cut one shape out of another: `max(a, -b)`



balls

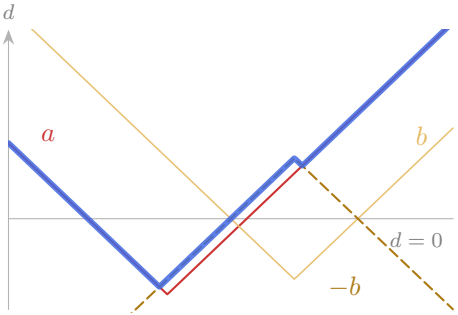


the same line, the same two balls

- To cut, we want: inside a, and **outside** b.
- $-b$ turns b inside out: below 0 everywhere outside the gold ball.

▶ **Run it in the March Lab:**
Slice: max(a, -b) cuts

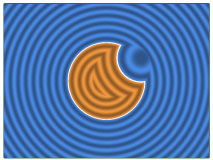
Cut one shape out of another: `max(a, -b)`





the same line, the same two balls

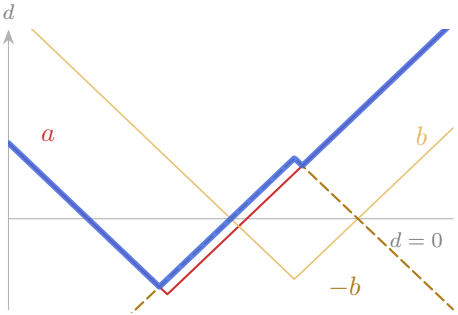
the gold ball cut out



- To cut, we want: inside a, and **outside** b.
- $-b$ turns b inside out: below 0 everywhere outside the gold ball.
- So $\max(a, -b)$ is below 0 only inside a and outside b.

▶ **Run it in the March Lab:**
Slice: max(a, -b) cuts

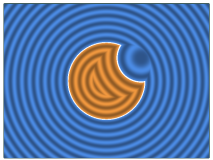
Cut one shape out of another: `max(a, -b)`





the same line, the same two balls

the gold ball cut out



```
max(ball(p), -other(p))
```

inside the red ball **and** outside the gold one

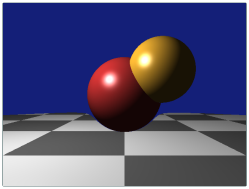
- To cut, we want: inside a, and **outside** b.
- $-b$ turns b inside out: below 0 everywhere outside the gold ball.
- So $\max(a, -b)$ is below 0 only inside a and outside b.

▶ **Run it in the March Lab:**
Slice: max(a, -b) cuts

In 3D: one line in scene changes the shape

```
return min(min(ball(p), other(p)), ground(p));
```

the floor joins with min, as in lecture 33; only the blue part changes



`min(ball(p), other(p))`

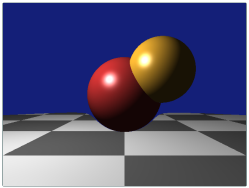
- min: both balls.

▶ **Run it in the March Lab:**
One cut from the other

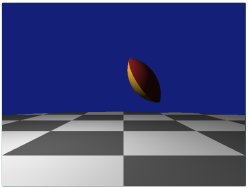
In 3D: one line in scene changes the shape

```
return min(min(ball(p), other(p)), ground(p));
```

the floor joins with min, as in lecture 33; only the blue part changes



min(ball(p), other(p))



max(ball(p), other(p))

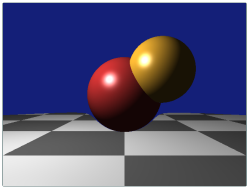
- min: both balls.
- max: only the lens where they overlap.

▶ **Run it in the March Lab:**
One cut from the other

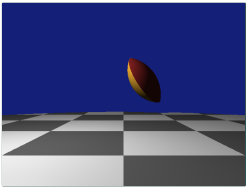
In 3D: one line in scene changes the shape

```
return min(min(ball(p), other(p)), ground(p));
```

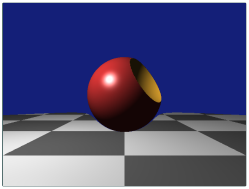
the floor joins with min, as in lecture 33; only the blue part changes



`min(ball(p), other(p))`



`max(ball(p), other(p))`



`max(ball(p), -other(p))`

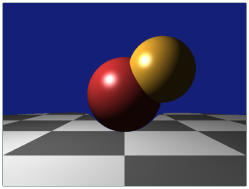
- min: both balls.
- max: only the lens where they overlap.
- max(a, -b): a bite out of the red ball.

▶ **Run it in the March Lab:**
One cut from the other

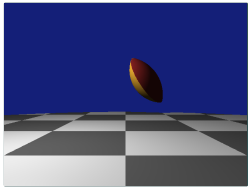
In 3D: one line in scene changes the shape

```
return min(min(ball(p), other(p)), ground(p));
```

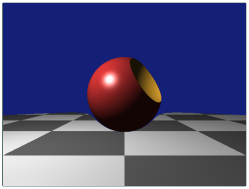
the floor joins with min, as in lecture 33; only the blue part changes



`min(ball(p), other(p))`



`max(ball(p), other(p))`



`max(ball(p), -other(p))`

- min: both balls.
- max: only the lens where they overlap.
- max(a, -b): a bite out of the red ball.
- The inside of the bite is gold: it is the gold ball's surface.

```
if (abs(other(P)) < abs(ball(P))) { base = gold }
```

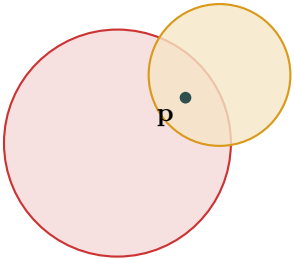
gold: the hit is nearer the gold ball's surface than the red one's

▶ **Run it in the March Lab:**
One cut from the other

Worked: inside or outside?

At $\mathbf{p} = (0.3, 0.2, 0)$: ball$(\mathbf{p}) = -0.14$, other$(\mathbf{p}) = -0.09$.

Try first: is $\mathbf{p}$ inside min? inside max? inside $\max(a, -b)$?



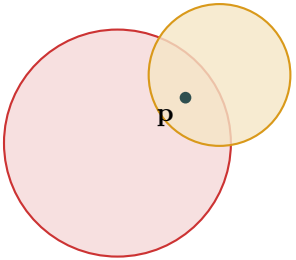
the slice $z = 0$

- Both distances are negative: $\mathbf{p}$ is inside both balls.

Worked: inside or outside?

At $\mathbf{p} = (0.3, 0.2, 0)$: ball$(\mathbf{p}) = -0.14$, other$(\mathbf{p}) = -0.09$.

Try first: is $\mathbf{p}$ inside min? inside max? inside $\max(a, -b)$?



the slice $z = 0$

$\min(-0.14, -0.09) = -0.14$

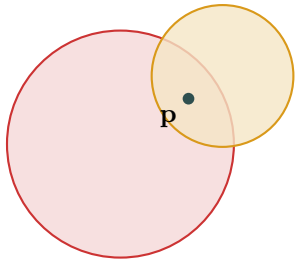
inside: in both balls

- Both distances are negative: $\mathbf{p}$ is inside both balls.
- min keeps the lower: still inside.

Worked: inside or outside?

At $\mathbf{p} = (0.3, 0.2, 0)$: ball$(\mathbf{p}) = -0.14$, other$(\mathbf{p}) = -0.09$.

Try first: is $\mathbf{p}$ inside min? inside max? inside $\max(a, -b)$?



the slice $z = 0$

$\min(-0.14, -0.09) = -0.14$

inside: in both balls

$\max(-0.14, -0.09) = -0.09$

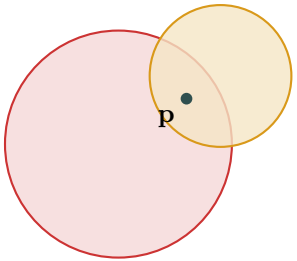
inside: in the overlap

- Both distances are negative: $\mathbf{p}$ is inside both balls.
- min keeps the lower: still inside.
- max keeps the higher: still inside.

Worked: inside or outside?

At $\mathbf{p} = (0.3, 0.2, 0)$: ball$(\mathbf{p}) = -0.14$, other$(\mathbf{p}) = -0.09$.

Try first: is $\mathbf{p}$ inside min? inside max? inside $\max(a, -b)$?



the slice $z = 0$

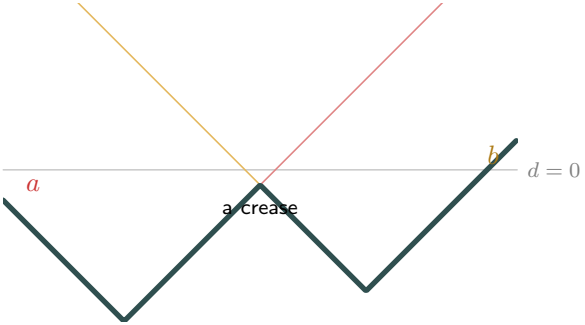
$\min(-0.14, -0.09) = -0.14$
inside: in both balls

$\max(-0.14, -0.09) = -0.09$
inside: in the overlap

$\max(-0.14, 0.09) = 0.09$
outside: $\mathbf{p}$ is in the bite

- Both distances are negative: $\mathbf{p}$ is inside both balls.
- min keeps the lower: still inside.
- max keeps the higher: still inside.
- $-b = +0.09$: the cut throws $\mathbf{p}$ out.

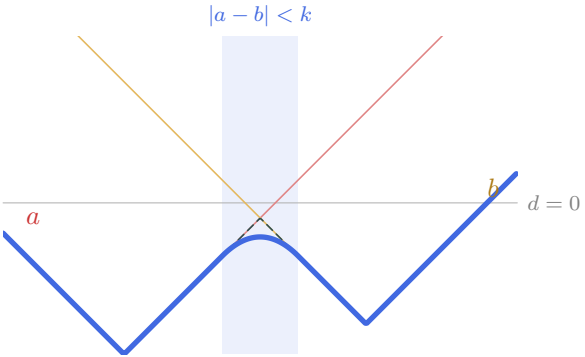
The smooth minimum: two shapes melt



- Where $a = b$, min has a sharp corner: the crease between the balls.

▶ **Run it in the March Lab:**
Slice: the smooth minimum

The smooth minimum: two shapes melt



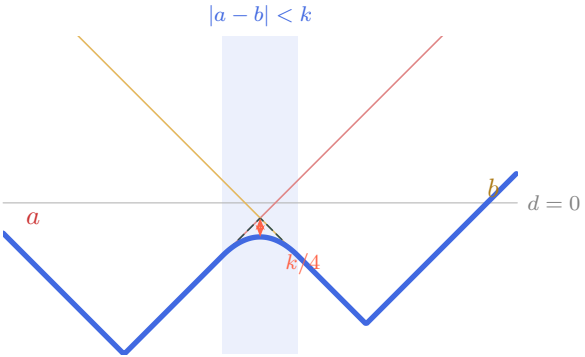

```
fn smin(a:  f32, b:  f32, k:  f32) -> f32 {
    let h = max(k - abs(a - b), 0.0) / k;
    return min(a, b) - 0.25 * k * h * h;
}
```

h: 1 where $a = b$, 0 once they differ by k

- Where $a = b$, min has a sharp corner: the crease between the balls.
- smin: where the two differ by less than k, take a little off.

▶ **Run it in the March Lab:**
Slice: the smooth minimum

The smooth minimum: two shapes melt



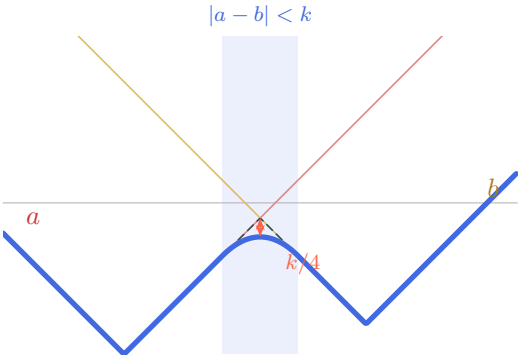

```
fn smin(a:  f32, b:  f32, k:  f32) -> f32 {
    let h = max(k - abs(a - b), 0.0) / k;
    return min(a, b) - 0.25 * k * h * h;
}
```

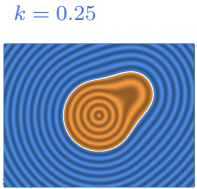
h: 1 where $a = b$, 0 once they differ by k

- Where $a = b$, min has a sharp corner: the crease between the balls.
- smin: where the two differ by less than k, take a little off.
- Most at the crease: $k/4$. The gap fills in.

▶ **Run it in the March Lab:**
Slice: the smooth minimum

The smooth minimum: two shapes melt





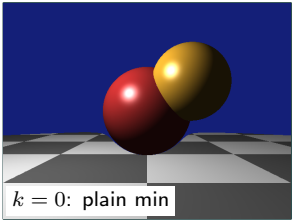

```
fn smin(a: f32, b: f32, k: f32) -> f32 {
    let h = max(k - abs(a - b), 0.0) / k;
    return min(a, b) - 0.25 * k * h * h;
}
```

h: 1 where $a = b$, 0 once they differ by k

- Where $a = b$, min has a sharp corner: the crease between the balls.
- smin: where the two differ by less than k, take a little off.
- Most at the crease: $k/4$. The gap fills in.
- On the slice: one blob.

▶ **Run it in the March Lab:**
Slice: the smooth minimum

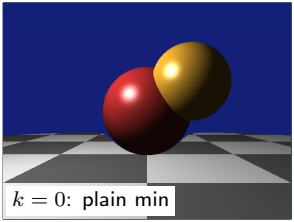
k: how far the melt reaches

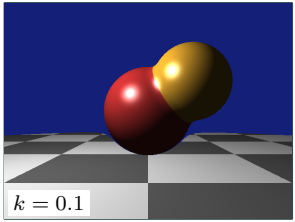



```
smin(ball(p),
     other(p), 0.25)
```

▶ **Run it in the March Lab:**
Two balls melted

k: how far the melt reaches



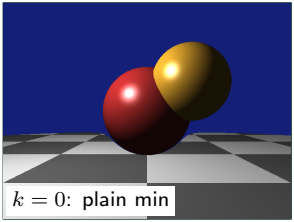


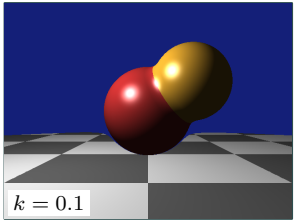

```
smin(ball(p),
     other(p), 0.25)
```

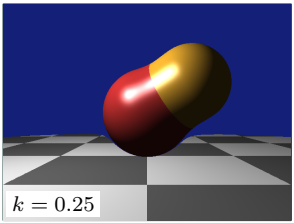
- k: how near the two surfaces must be to melt.

▶ **Run it in the March Lab:**
Two balls melted

k: how far the melt reaches





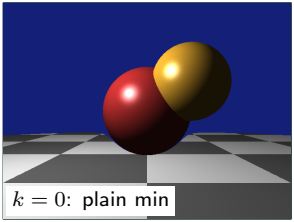


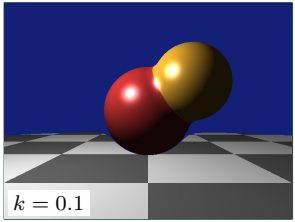

```
smin(ball(p),
     other(p), 0.25)
```

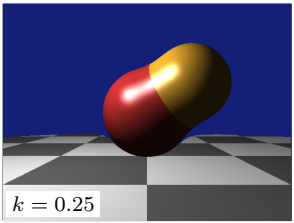
- k: how near the two surfaces must be to melt.
- A bigger k: a wider neck.

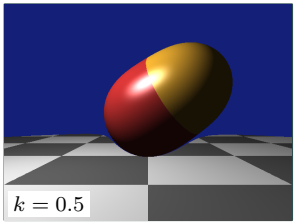
▶ **Run it in the March Lab:**
Two balls melted

k: how far the melt reaches










```
smin(ball(p),
     other(p), 0.25)
```

- k: how near the two surfaces must be to melt.
- A bigger k: a wider neck.
- $k = 0.5$: one blob. Every distance shrinks, so the whole shape swells.

▶ **Run it in the March Lab:**
Two balls melted

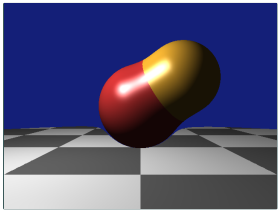
You Try 1: melt into the floor

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

```
fn scene(p:  vec3f) -> f32 {
    return min(smin(ball(p), other(p), 0.25), ground(p));
}
```

The outer min joins the floor. Melt it in too.

now



after your change



- **Melt the two balls into the floor too.**
- Guess first: what happens where the ball meets the floor?

You Try 1: melt into the floor

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

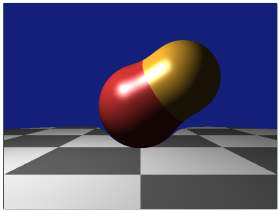
```
fn scene(p:  vec3f) -> f32 {
    return min(smin(ball(p), other(p), 0.25), ground(p));
}
```

The outer min joins the floor. Melt it in too.

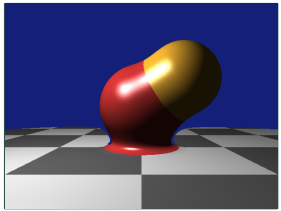
answer:

```
return smin(smin(ball(p), other(p), 0.25),
            ground(p), 0.25);
```

now



after your change



- **Melt the two balls into the floor too.**
- Guess first: what happens where the ball meets the floor?
- A smooth foot, like a drop of honey. The checks stop where the foot begins.

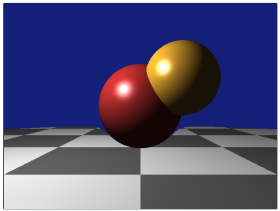
You Try 2: cut a hole

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

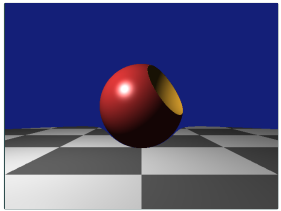
```
fn scene(p:  vec3f) -> f32 {
    return min(min(ball(p), other(p)), ground(p));
}
```

- **Cut the gold ball out of the red one.**
- Before you run it: what colour is the inside of the hole?

now



make this



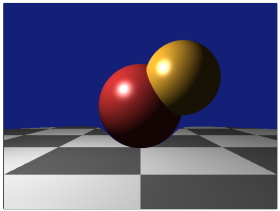
You Try 2: cut a hole

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

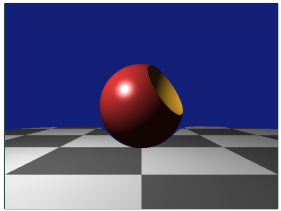
```
fn scene(p:  vec3f) -> f32 {
    return min(min(ball(p), other(p)), ground(p));
}
```

answer: `return min(max(ball(p), -other(p)), ground(p));`

now



make this



- **Cut the gold ball out of the red one.**
- Before you run it: what colour is the inside of the hole?
- Gold: the hole's surface is the gold ball's surface (slide 7).

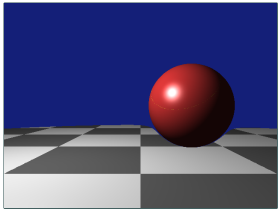
Round 2

Bend the space

move the point, not the shape

Change the point, then ask its distance

Move

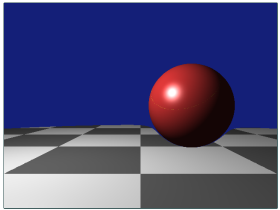


```
length(p - centre) - r
```

lecture 33 did it already: centre (0.6, 0, 0) moves the ball right

Change the point, then ask its distance

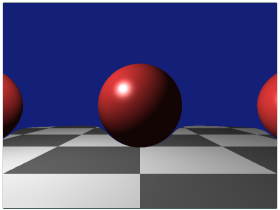
Move



```
length(p - centre) - r
```

lecture 33 did it already: centre (0.6, 0, 0) moves the ball right

Repeat

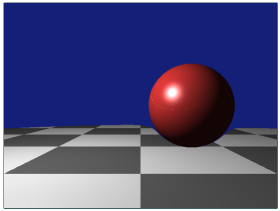


```
fract(p.x / cell + 0.5)
```

fold every cell onto one: one ball, many copies

Change the point, then ask its distance

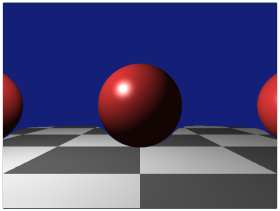
Move



```
length(p - centre) - r
```

lecture 33 did it already: centre (0.6, 0, 0) moves the ball right

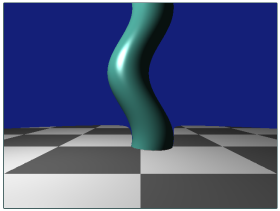
Repeat



```
fract(p.x / cell + 0.5)
```

fold every cell onto one: one ball, many copies

Bend

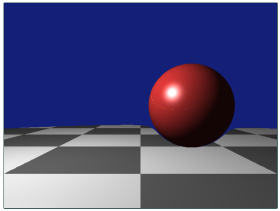


```
p.x + 0.15 * sin(4.0 * p.y)
```

push each point sideways by its height: a wave

Change the point, then ask its distance

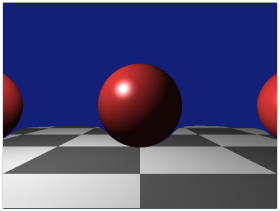
Move



```
length(p - centre) - r
```

lecture 33 did it already: centre (0.6, 0, 0) moves the ball right

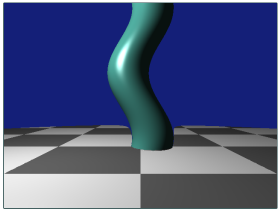
Repeat



```
fract(p.x / cell + 0.5)
```

fold every cell onto one: one ball, many copies

Bend

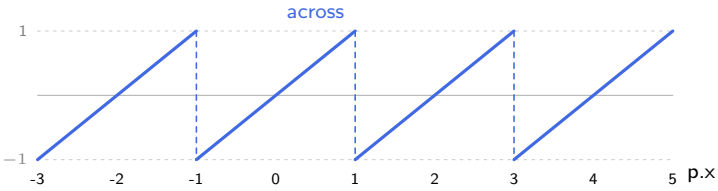


```
p.x + 0.15 * sin(4.0 * p.y)
```

push each point sideways by its height: a wave

No new distance function: the ball asks about a changed point, and comes out moved, repeated or bent.

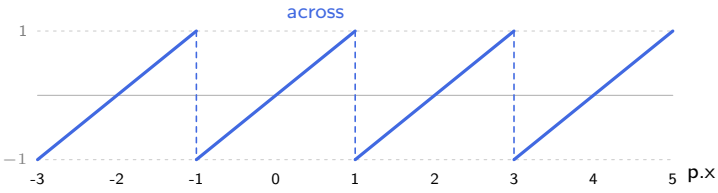
fract folds space into cells: one ball, many copies



- fract (lecture 26, slide 26) runs from 0 to 1, again and again.

▶ **Run it in the March Lab:**
A row of balls: fract

fract folds space into cells: one ball, many copies



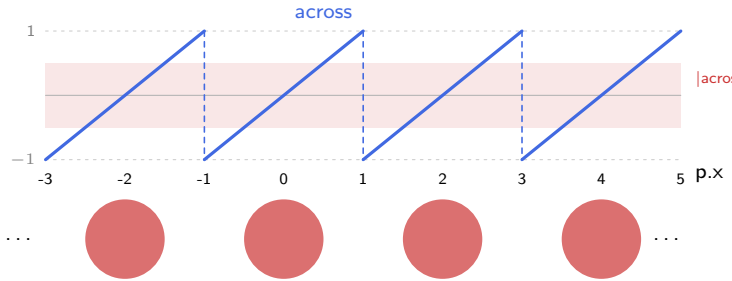
- fract (lecture 26, slide 26) runs from 0 to 1, again and again.
- So across runs from -1 to 1 in every cell of 2.0.

```
let cell = 2.0;
let across = cell * fract(p.x / cell + 0.5) - 0.5 * cell;
let q = vec3f(across, p.y, p.z);
return length(q - centre) - r;
```

y and z stay as they were: the row runs along x only.

▶ **Run it in the March Lab:**
A row of balls: fract

fract folds space into cells: one ball, many copies




```
let cell = 2.0;
let across = cell * fract(p.x / cell + 0.5) - 0.5 * cell;
let q = vec3f(across, p.y, p.z);
return length(q - centre) - r;
```

y and z stay as they were: the row runs along x only.

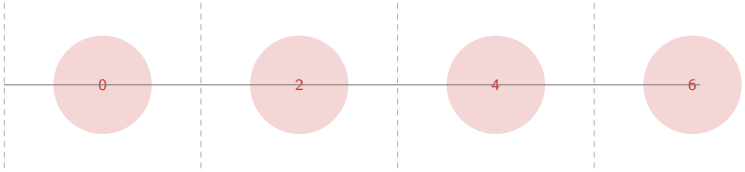
- fract (lecture 26, slide 26) runs from 0 to 1, again and again.
- So across runs from -1 to 1 in every cell of 2.0.
- The ball sees only across: one ball in every cell.

▶ **Run it in the March Lab:**
A row of balls: fract

Worked: which ball is nearest?

across $= 2\,\mathrm{fract}(\mathrm{p.x}/2 + 0.5) - 1$, cell 2.0, balls at 0, 2, 4, ...

Try first: across for p.x $= 3.3,\ 0.9,\ 5.0$. Which ball is nearest?



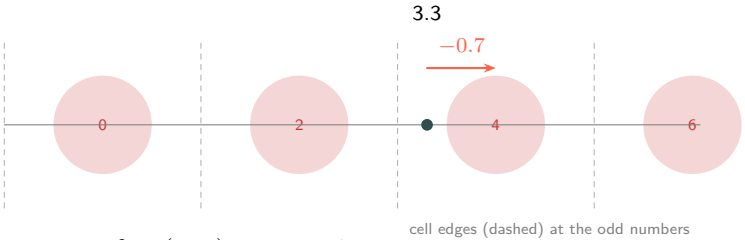
cell edges (dashed) at the odd numbers

Worked: which ball is nearest?

across $= 2\,\mathrm{fract}(\mathrm{p.x}/2 + 0.5) - 1$, cell 2.0, balls at 0, 2, 4, ...

Try first: across for p.x $= 3.3,\ 0.9,\ 5.0$. Which ball is nearest?

- 3.3: 0.7 left of a centre. The ball at 4.

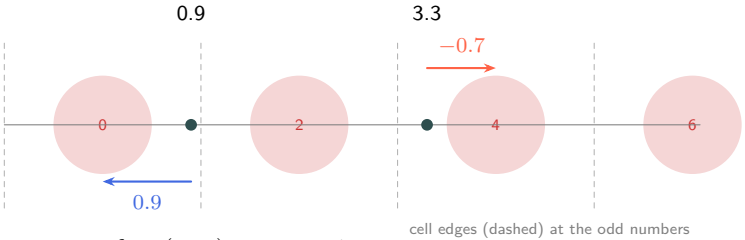


p.x $= 3.3$: $2\,\mathrm{fract}(2.15) - 1 = -0.7$

Worked: which ball is nearest?

across $= 2\,\mathrm{fract}(\mathrm{p.x}/2 + 0.5) - 1$, cell 2.0, balls at 0, 2, 4, ...

Try first: across for p.x $= 3.3,\ 0.9,\ 5.0$. Which ball is nearest?



cell edges (dashed) at the odd numbers

p.x $= 3.3$: $2\,\mathrm{fract}(2.15) - 1 = -0.7$

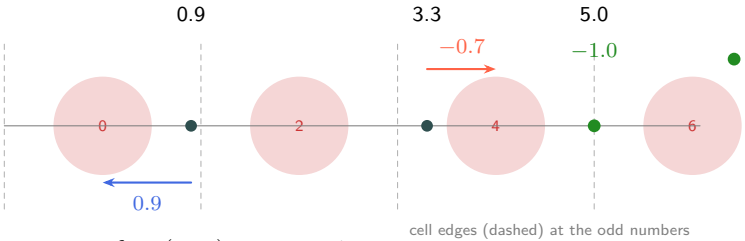
p.x $= 0.9$: $2\,\mathrm{fract}(0.95) - 1 = 0.9$

- 3.3: 0.7 left of a centre. The ball at 4.
- 0.9: 0.9 right of a centre. The ball at 0.

Worked: which ball is nearest?

across $= 2\,\text{fract}(\text{p.x}/2 + 0.5) - 1$, cell 2.0, balls at 0, 2, 4, ...

Try first: across for p.x $= 3.3,\ 0.9,\ 5.0$. Which ball is nearest?



cell edges (dashed) at the odd numbers

p.x $= 3.3$: $2\,\text{fract}(2.15) - 1 = -0.7$

p.x $= 0.9$: $2\,\text{fract}(0.95) - 1 = 0.9$

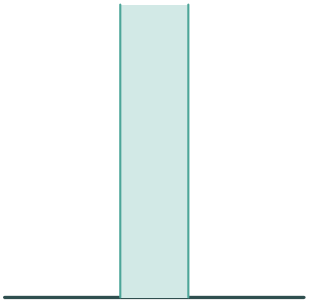
p.x $= 5.0$: $2\,\text{fract}(3.0) - 1 = -1$

- 3.3: 0.7 left of a centre. The ball at 4.
- 0.9: 0.9 right of a centre. The ball at 0.
- 5.0: on a cell edge, halfway between the balls at 4 and 6.

Bend: push each point sideways, then ask

```
fn column(p:  vec3f) -> f32 {

    return length(vec2f(q.x, q.z)) - 0.25;
}
```



from the front

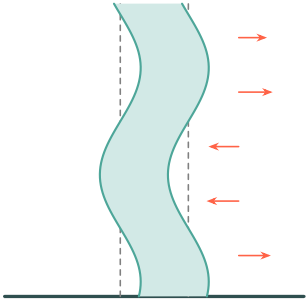
- A column: lecture 26's circle in x and z. The height is ignored.

▶ **Run it in the March Lab:**
A bent column

Bend: push each point sideways, then ask

```
fn column(p:  vec3f) -> f32 {
    let q = vec3f(p.x + 0.15 * sin(4.0 * p.y), p.y, p.z);
    return length(vec2f(q.x, q.z)) - 0.25;
}
```

the push, at each height (drawn larger)



from the front

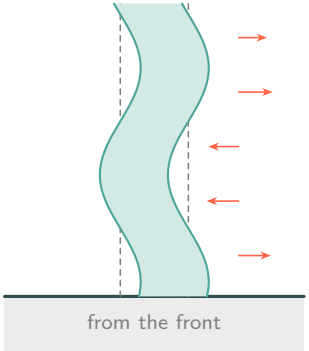
- A column: lecture 26's circle in x and z. The height is ignored.
- Before asking, push each point sideways by a sine of its height.

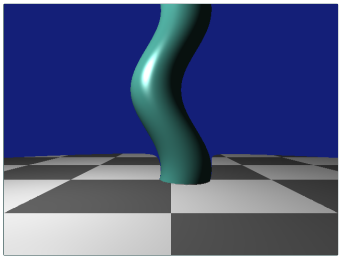
▶ **Run it in the March Lab:**
A bent column

Bend: push each point sideways, then ask

```
fn column(p:  vec3f) -> f32 {
    let q = vec3f(p.x + 0.15 * sin(4.0 * p.y), p.y, p.z);
    return length(vec2f(q.x, q.z)) - 0.25;
}
```

the push, at each height (drawn larger)





- A column: lecture 26's circle in x and z. The height is ignored.
- Before asking, push each point sideways by a sine of its height.
- The straight column comes out as a wave.

▶ **Run it in the March Lab:**
A bent column

You Try 3: use the folded point

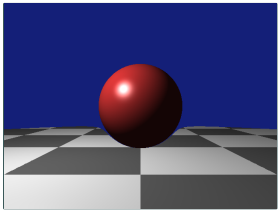
In the March Lab, choose **You Try 3**. Guess first, then change the code.

```
let q = vec3f(across, p.y, p.z);
let centre = vec3f(0.0, 0.0, 0.0);
let r = 0.5;
return length(p - centre) - r;
```

change this line

- **The folded point q is worked out but never used. Use it.**
- Guess first: how far apart are the balls' edges?

now



after your change



You Try 3: use the folded point

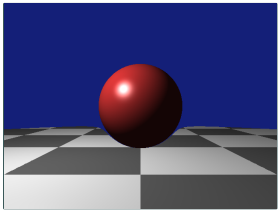
In the March Lab, choose **You Try 3**. Guess first, then change the code.

```
let q = vec3f(across, p.y, p.z);
let centre = vec3f(0.0, 0.0, 0.0);
let r = 0.5;
return length(p - centre) - r;
```

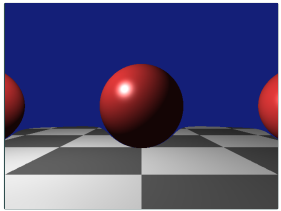
change this line

answer: `return length(q - centre) - r;`

now



after your change



- **The folded point q is worked out but never used. Use it.**
- Guess first: how far apart are the balls' edges?
- 1.0: centres 2.0 apart, minus a radius of 0.5 on each side.

You Try 4: a cell 1.0 wide

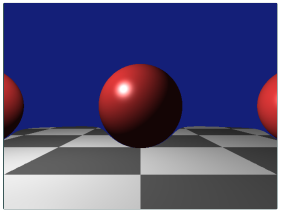
In the March Lab, choose **You Try 4**. Guess first, then change the code.

```
let cell = 2.0;
let across = cell * fract(p.x / cell + 0.5) - 0.5 * cell;
let q = vec3f(across, p.y, p.z);
```

change this line

- **Make the cell 1.0 wide.**
- Before you run it: do the balls touch, overlap, or keep a gap?

now



after your change



You Try 4: a cell 1.0 wide

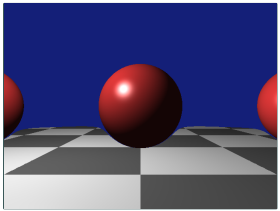
In the March Lab, choose **You Try 4**. Guess first, then change the code.

```
let cell = 2.0;
let across = cell * fract(p.x / cell + 0.5) - 0.5 * cell;
let q = vec3f(across, p.y, p.z);
```

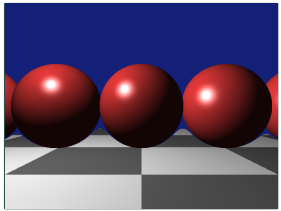
change this line

answer: `let cell = 1.0;`

now



after your change



- **Make the cell 1.0 wide.**
- Before you run it: do the balls touch, overlap, or keep a gap?
- They just touch: centres 1.0 apart, two radii of 0.5. A string of beads.

Round 3

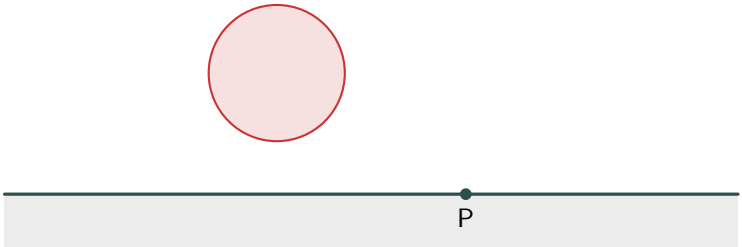
Shadows from the march

march again, toward the lamp

March again, from the hit toward the lamp

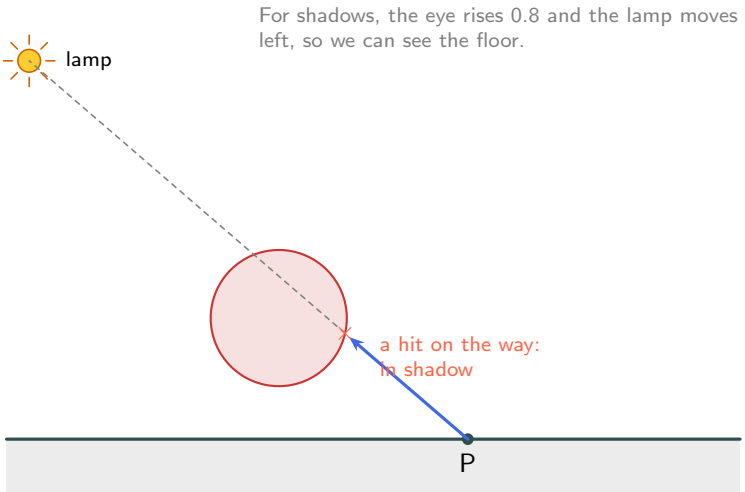


For shadows, the eye rises 0.8 and the lamp moves left, so we can see the floor.



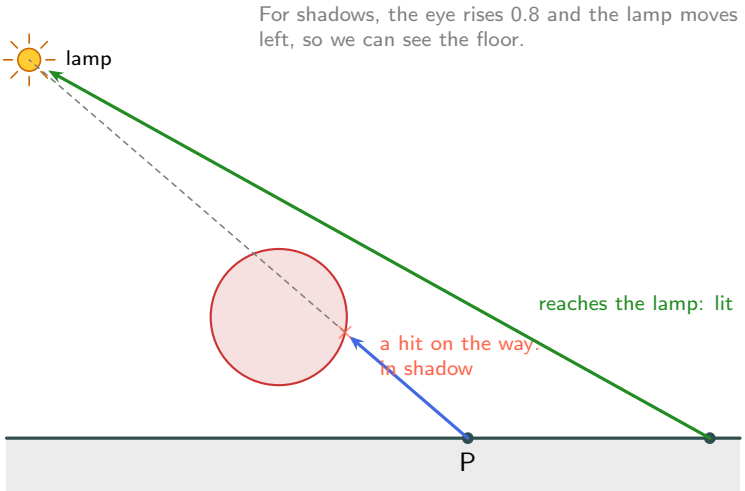
- The first march found P. Is the lamp hidden from P?

March again, from the hit toward the lamp



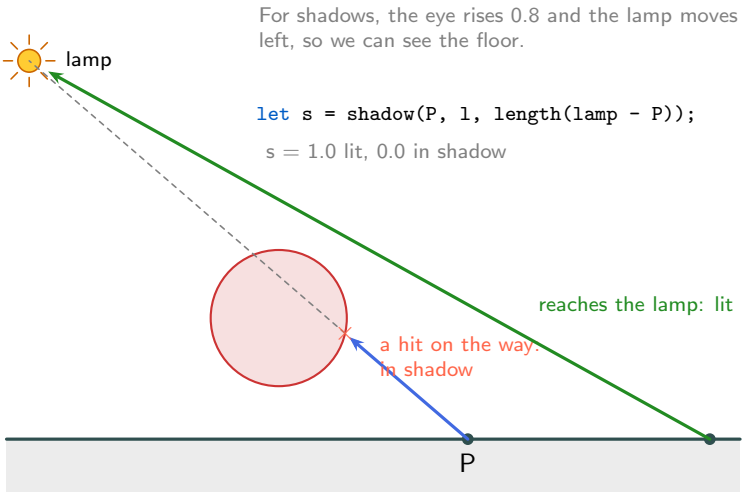
- The first march found P. Is the lamp hidden from P?
- March from P toward the lamp. Something on the way: in shadow.

March again, from the hit toward the lamp



- The first march found P. Is the lamp hidden from P?
- March from P toward the lamp. Something on the way: in shadow.
- Nothing on the way to the lamp: lit.

March again, from the hit toward the lamp



- The first march found P. Is the lamp hidden from P?
- March from P toward the lamp. Something on the way: in shadow.
- Nothing on the way to the lamp: lit.
- Lecture 31's shadow ray, done by marching.

A hard shadow in code

```
fn shadow(start: vec3f, l: vec3f, far: f32) -> f32 {
    var t = 0.02;                          start a little off the surface
    for (var i = 0; i < 64; i++) {
        let d = scene(start + t * l);
        if (d < 0.001) { return 0.0; }     a hit on the way: dark
        t = t + d;
        if (t > far) { break; }            past the lamp: lit
    }
    return 1.0;
}
```

- shadow: lecture 33's march, from P along l.

▶ **Run it in the March Lab:**
A hard shadow

A hard shadow in code

```
fn shadow(start:  vec3f, l:  vec3f, far:  f32) -> f32 {
    var t = 0.02;                          start a little off the surface
    for (var i = 0; i < 64; i++) {
        let d = scene(start + t * l);
        if (d < 0.001) { return 0.0; }     a hit on the way: dark
        t = t + d;
        if (t > far) { break; }            past the lamp: lit
    }
    return 1.0;
}
```

```
let s = shadow(P, l, length(lamp - P));
let c = base * (ambient + s * diffuse) + s * shine;
```

in shadow: only the ambient light is left

- shadow: lecture 33's march, from P along l.
- s scales the lamp's light: 0 leaves the ambient.

▶ **Run it in the March Lab:**
A hard shadow

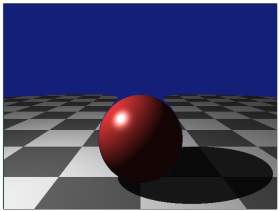
A hard shadow in code

```
fn shadow(start: vec3f, l: vec3f, far: f32) -> f32 {
    var t = 0.02;                          start a little off the surface
    for (var i = 0; i < 64; i++) {
        let d = scene(start + t * l);
        if (d < 0.001) { return 0.0; }     a hit on the way: dark
        t = t + d;
        if (t > far) { break; }            past the lamp: lit
    }
    return 1.0;
}
```

a sharp edge: lit or not, nothing between

```
let s = shadow(P, l, length(lamp - P));
let c = base * (ambient + s * diffuse) + s * shine;
```

in shadow: only the ambient light is left



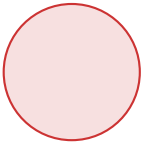
- shadow: lecture 33's march, from P along l.
- s scales the lamp's light: 0 leaves the ambient.
- Hard: a point lamp is either seen or not.

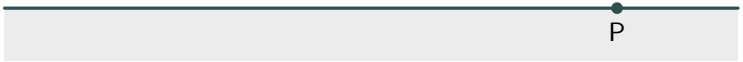
▶ **Run it in the March Lab:**
A hard shadow

Soft: how close did the ray pass?



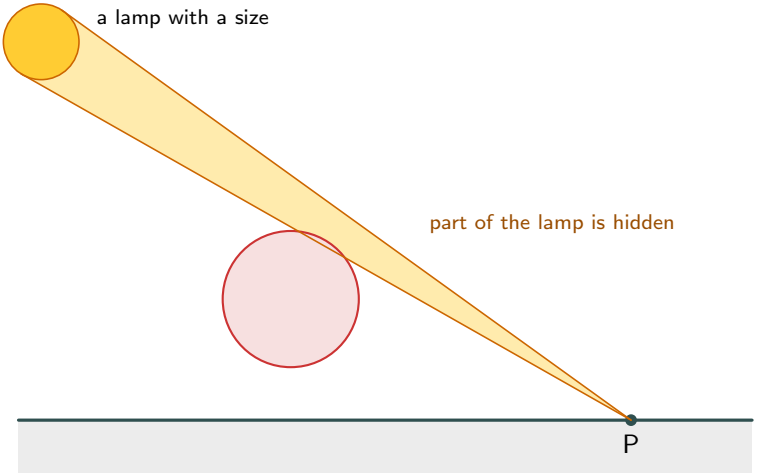
a lamp with a size





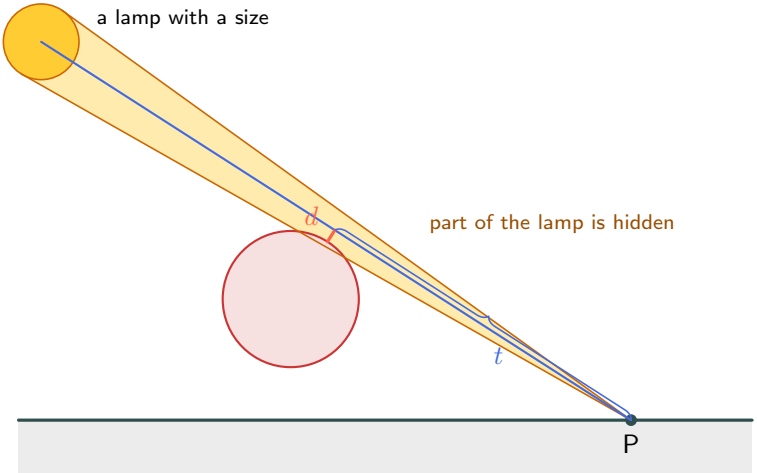
- A real lamp has a size: from P, part of it can hide behind the ball.

Soft: how close did the ray pass?



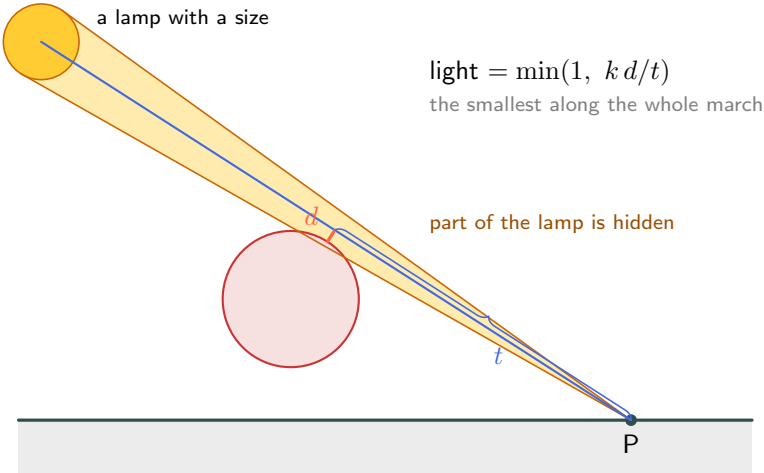
- A real lamp has a size: from P, part of it can hide behind the ball.
- Part hidden: part lit. That is the soft edge.

Soft: how close did the ray pass?



- A real lamp has a size: from P, part of it can hide behind the ball.
- Part hidden: part lit. That is the soft edge.
- The ray passes the ball at distance d, a length t from P. Seen from P, the gap is about d/t wide.

Soft: how close did the ray pass?



- A real lamp has a size: from P, part of it can hide behind the ball.
- Part hidden: part lit. That is the soft edge.
- The ray passes the ball at distance d, a length t from P. Seen from P, the gap is about d/t wide.
- A small d/t: most of the lamp hidden.

A soft shadow: three lines change

```
fn shadow(start:  vec3f, l:  vec3f, far:  f32) -> f32 {
    var light = 1.0;
    var t = 0.02;
    for (var i = 0; i < 64; i++) {
        let d = scene(start + t * l);
        if (d < 0.001) { return 0.0; }
        light = min(light, 8.0 * d / t);
        t = t + d;
        if (t > far) { break; }
    }
    return light;
}
```

- Light starts at 1. Every step may lower it.

▶ **Run it in the March Lab:**
A soft shadow

A soft shadow: three lines change

```
fn shadow(start: vec3f, l: vec3f, far: f32) -> f32 {
    var light = 1.0;
    var t = 0.02;
    for (var i = 0; i < 64; i++) {
        let d = scene(start + t * l);
        if (d < 0.001) { return 0.0; }
        light = min(light, 8.0 * d / t);
        t = t + d;
        if (t > far) { break; }
    }
    return light;
}
```

8.0 is k. Each step may lower light; it never rises again.

- Light starts at 1. Every step may lower it.
- A near pass far from P: a small d/t, a dark step.

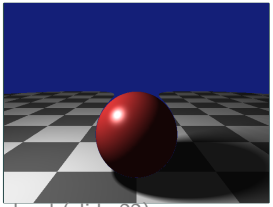
▶ **Run it in the March Lab:**
A soft shadow

A soft shadow: three lines change

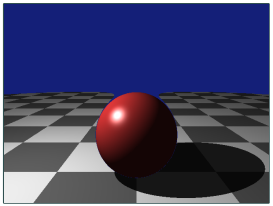
```
fn shadow(start: vec3f, l: vec3f, far: f32) -> f32 {
    var light = 1.0;
    var t = 0.02;
    for (var i = 0; i < 64; i++) {
        let d = scene(start + t * l);
        if (d < 0.001) { return 0.0; }
        light = min(light, 8.0 * d / t);
        t = t + d;
        if (t > far) { break; }
    }
    return light;
}
```

soft

8.0 is k. Each step may lower light; it never rises again.



hard (slide 22)



- Light starts at 1. Every step may lower it.
- A near pass far from P: a small d/t, a dark step.
- The same march, the same steps: the soft edge is one line.

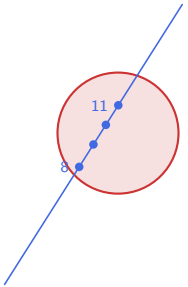
▶ **Run it in the March Lab:**
A soft shadow

Worked: how much light?

A shadow ray from a floor point in the soft edge, $k = 8$. Steps 8 to 11:

Try first: $8\,d/t$ for each step. How much light reaches the point?

step	t	d	$8\,d/t$
8	0.577	0.063	
9	0.640	0.051	
10	0.691	0.046	
11	0.736	0.045	



not to scale

Worked: how much light?

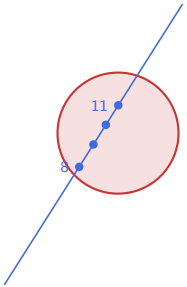
A shadow ray from a floor point in the soft edge, $k = 8$. Steps 8 to 11:

Try first: $8\,d/t$ for each step. How much light reaches the point?

step	t	d	$8\,d/t$
8	0.577	0.063	0.87
9	0.640	0.051	0.64
10	0.691	0.046	0.53
11	0.736	0.045	0.49

after step 11, d grows again: the ray is past the ball

- $8 \times 0.063/0.577 = 0.87$, and so on.



not to scale

Worked: how much light?

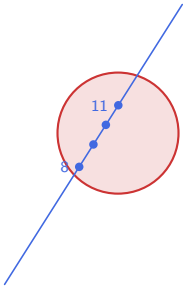
A shadow ray from a floor point in the soft edge, $k = 8$. Steps 8 to 11:

Try first: $8\,d/t$ for each step. How much light reaches the point?

step	t	d	$8\,d/t$
8	0.577	0.063	0.87
9	0.640	0.051	0.64
10	0.691	0.046	0.53
11	0.736	0.045	0.49

after step 11, d grows again: the ray is past the ball

light $= 0.49$: about half the lamp is seen.



not to scale

- $8 \times 0.063/0.577 = 0.87$, and so on.
- The smallest wins: step 11, where the ray passes closest.

Worked: how much light?

A shadow ray from a floor point in the soft edge, $k = 8$. Steps 8 to 11:

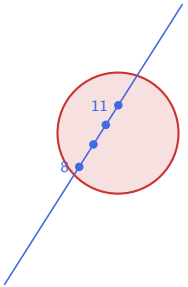
Try first: $8\,d/t$ for each step. How much light reaches the point?

step	t	d	$8\,d/t$
8	0.577	0.063	0.87
9	0.640	0.051	0.64
10	0.691	0.046	0.53
11	0.736	0.045	0.49

after step 11, d grows again: the ray is past the ball

light $= 0.49$: about half the lamp is seen.

measured over all 20 steps: 0.491

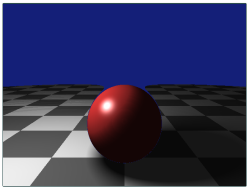


not to scale

- $8 \times 0.063/0.577 = 0.87$, and so on.
- The smallest wins: step 11, where the ray passes closest.
- 0.49: this pixel sits in the middle of the soft edge.

k: the softness number

$k = 2$

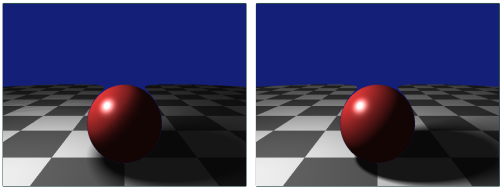


```
light = min(light, k * d / t);
```

- $k = 2$: a wide, soft edge, like a big lamp.

k: the softness number

$k = 2$ $k = 8$

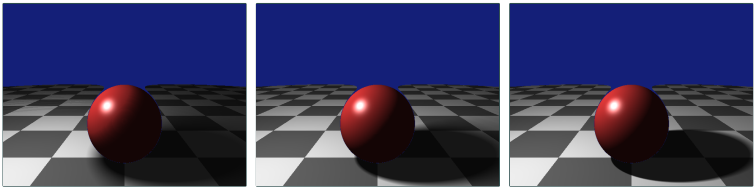


```
light = min(light, k * d / t);
```

- $k = 2$: a wide, soft edge, like a big lamp.
- $k = 8$: today's.

k: the softness number

$k = 2$ $k = 8$ $k = 32$

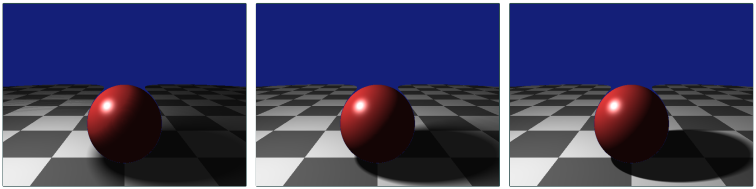


- $k = 2$: a wide, soft edge, like a big lamp.
- $k = 8$: today's.
- $k = 32$: nearly the hard shadow: a small lamp.

```
light = min(light, k * d / t);
```

k: the softness number

$k = 2$ $k = 8$ $k = 32$



```
light = min(light, k * d / t);
```

Rays that leave the far floor at a low angle pass close to the floor itself, so a small k darkens them too.

- $k = 2$: a wide, soft edge, like a big lamp.
- $k = 8$: today's.
- $k = 32$: nearly the hard shadow: a small lamp.
- k plays the lamp's size, backwards. It is a knob, not a measurement.

Where distance fields are used



Letters (Valve, 2007)

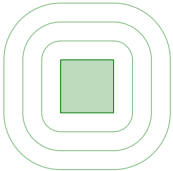
Chris Green stored each letter as a small grid of distances. The edge is where the distance is 0: sharp at any size. Used in Team Fortress 2.

Where distance fields are used



Letters (Valve, 2007)

Chris Green stored each letter as a small grid of distances. The edge is where the distance is 0: sharp at any size. Used in Team Fortress 2.



Unreal Engine

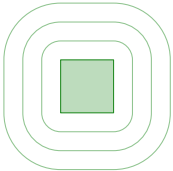
A grid of distances for every mesh, made ahead of time. Soft shadows from one march, as today; Lumen's software ray tracing marches the same grids.

Where distance fields are used



Letters (Valve, 2007)

Chris Green stored each letter as a small grid of distances. The edge is where the distance is 0: sharp at any size. Used in Team Fortress 2.



Unreal Engine

A grid of distances for every mesh, made ahead of time. Soft shadows from one march, as today; Lumen's software ray tracing marches the same grids.



Dreams (PS4, 2020)

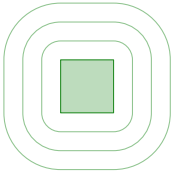
Players sculpt with shapes added and cut away: our min, max(a, -b) and the same smooth min.

Where distance fields are used



Letters (Valve, 2007)

Chris Green stored each letter as a small grid of distances. The edge is where the distance is 0: sharp at any size. Used in Team Fortress 2.



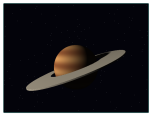
Unreal Engine

A grid of distances for every mesh, made ahead of time. Soft shadows from one march, as today; Lumen's software ray tracing marches the same grids.



Dreams (PS4, 2020)

Players sculpt with shapes added and cut away: our min, max(a, -b) and the same smooth min.



Shadertoy (2013)

A web site of fragment shaders, like the March Lab, by Pol Jeremias and Inigo Quilez. Many whole scenes there are marched.

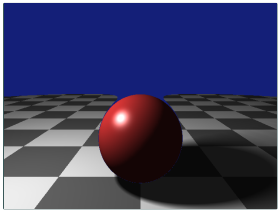
You Try 5: a low lamp

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

```
let lamp = vec3f(-1.5, 1.5, 0.5);
let l = normalize(lamp - P);
```

change this line

now



after your change



- **Move the lamp down low: (-2.0, 0.5, 0.5).**
- Guess first: does the shadow get longer or shorter? Softer or sharper at its far end?

You Try 5: a low lamp

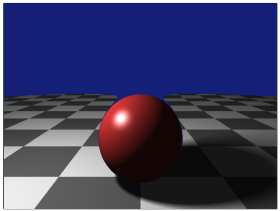
In the March Lab, choose **You Try 5**. Guess first, then change the code.

```
let lamp = vec3f(-1.5, 1.5, 0.5);
let l = normalize(lamp - P);
```

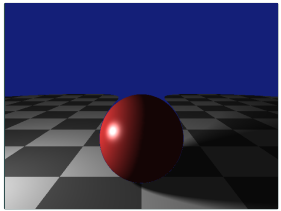
change this line

answer: `let lamp = vec3f(-2.0, 0.5, 0.5);`

now



after your change



- **Move the lamp down low: (-2.0, 0.5, 0.5).**
- Guess first: does the shadow get longer or shorter? Softer or sharper at its far end?
- Longer, and softer at the far end: there t is big, so d/t is small.

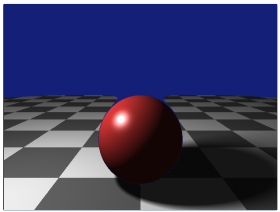
You Try 6: a shadow on the ball

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

```
fn scene(p:  vec3f) -> f32 {
    return min(ball(p), ground(p));
}
```

change this line

now



after your change



- **Add a small ball, radius 0.15, at (-0.55, 0.55, 0.2).**
- It hangs between the lamp and the big ball. Guess first: where does its shadow fall?

You Try 6: a shadow on the ball

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

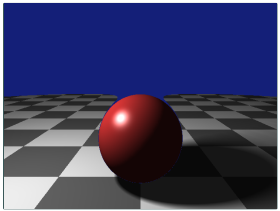
```
fn scene(p:  vec3f) -> f32 {
    return min(ball(p), ground(p));
}
```

change this line

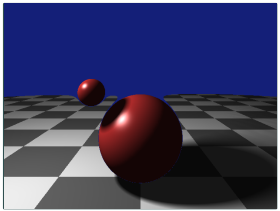
answer:

```
let small = length(p - vec3f(-0.55, 0.55, 0.2)) - 0.15;
return min(min(ball(p), small), ground(p));
```

now



after your change



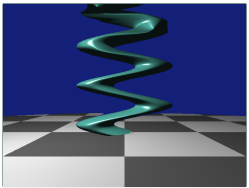
- **Add a small ball, radius 0.15, at (-0.55, 0.55, 0.2).**
- It hangs between the lamp and the big ball. Guess first: where does its shadow fall?
- On the big ball: a soft dark spot. The shadow march sees every shape, not only the floor.

What bent space gets wrong: the distance is too big

```
let q = vec3f(p.x + 0.4 * sin(12.0 * p.y), p.y, p.z);
```

slide 17's bend, pushed much harder

step d, 64 steps



22 steps a pixel

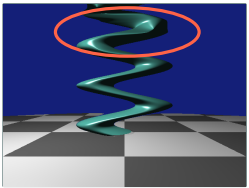
- A strong bend: a point's distance can be up to about 5 times too big.

What bent space gets wrong: the distance is too big

```
let q = vec3f(p.x + 0.4 * sin(12.0 * p.y), p.y, p.z);
```

slide 17's bend, pushed much harder

step d, 64 steps



22 steps a pixel

tears: the march stepped past the surface

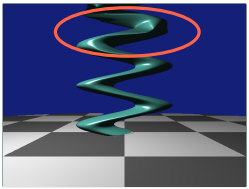
- A strong bend: a point's distance can be up to about 5 times too big.
- Then a step of d jumps through the surface: tears.

What bent space gets wrong: the distance is too big

```
let q = vec3f(p.x + 0.4 * sin(12.0 * p.y), p.y, p.z);
```

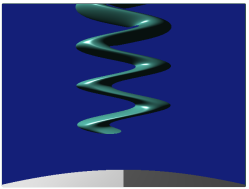
slide 17's bend, pushed much harder

step d, 64 steps



22 steps a pixel

step $0.25\,d$, 64 steps



55 steps a pixel

tears: the march stepped past the surface

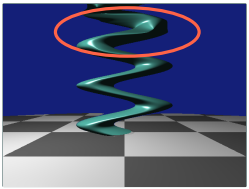
- A strong bend: a point's distance can be up to about 5 times too big.
- Then a step of d jumps through the surface: tears.
- Step a quarter of d: no tears, but 64 steps run out: the floor is lost.

What bent space gets wrong: the distance is too big

```
let q = vec3f(p.x + 0.4 * sin(12.0 * p.y), p.y, p.z);
```

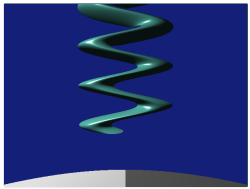
slide 17's bend, pushed much harder

step d, 64 steps



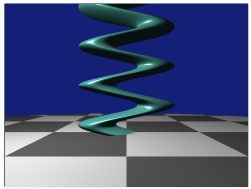
22 steps a pixel

step $0.25\,d$, 64 steps



55 steps a pixel

step $0.25\,d$, 256 steps



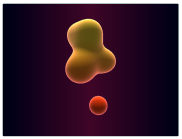
91 steps a pixel

tears: the march stepped past the surface

Right again, but four times the work.

- A strong bend: a point's distance can be up to about 5 times too big.
- Then a step of d jumps through the surface: tears.
- Step a quarter of d: no tears, but 64 steps run out: the floor is lost.
- Allow 256 steps: right again, at 91 steps a pixel.

Now make your own picture



A lava lamp

`smin(d, blob, 0.5)`



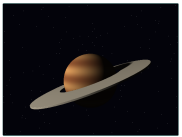
A snowman

`three balls, two smins`



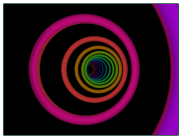
Endless columns

`fract in x and z`



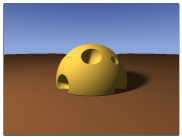
A ringed planet

`its shadow on the ring`



A tunnel of rings

`fract(p.z), bent by sin`



Swiss cheese

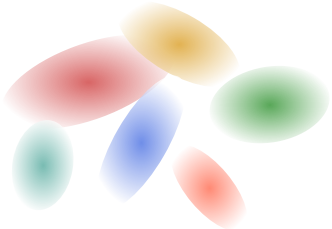
`max(wheel, -holes)`

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

Summary

1	Combine	`min(a, b)`, `max(a, b)`, `max(a, -b)` both shapes; only the overlap; one cut from the other
2	Melt	`smin(a, b, k)` within k of each other, take a little off: at most $k/4$
3	Bend the space	change the point, then ask fract folds it into cells; a sine pushes it sideways
4	Shadows	march from P toward the lamp hard: anything on the way; soft: the smallest $k\,d/t$
5	What it gets wrong	a strong bend makes the distance too big tears; smaller steps fix them, at four times the work

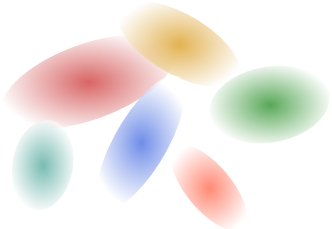
Next lecture: a picture from blobs



no triangles, no march: soft blobs

- Lecture 35: a picture made of soft, coloured blobs, blended one over another.

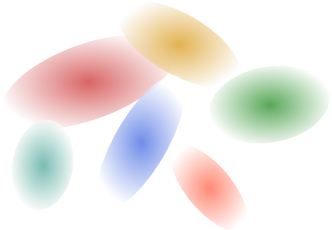
Next lecture: a picture from blobs



no triangles, no march: soft blobs

- Lecture 35: a picture made of soft, coloured blobs, blended one over another.
- The first step to Gaussian splatting: whole scenes captured from photos.

Next lecture: a picture from blobs



no triangles, no march: soft blobs

- Lecture 35: a picture made of soft, coloured blobs, blended one over another.
- The first step to Gaussian splatting: whole scenes captured from photos.
- The order of the blobs matters. Why?

Exit Ticket

Today's question *(2 minutes)*

(a) At a point, ball A's distance is 0.3 and ball B's is -0.1. Is the point inside $\min(a, b)$? Inside $\max(a, b)$? Inside $\max(a, -b)$?

(b) Work out $\text{smin}(0.1,\ 0.2,\ 0.3)$.

CS 453 — Exit Ticket #34

Name: _____

Roll #: _____ Date: _____

Answer:

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