Lecture 33: Ray Marching

A Scene from a Distance

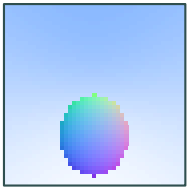
CS 453 — Computer Graphics

Fakhir Shaheen

Forman Christian University

Lecture 14's sphere, with no equation

lecture 14



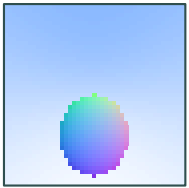
$$At^2 + Bt + C = 0$$

solve an equation: a new one for each shape

- Lecture 14 (slide 8) solved a quadratic to meet the sphere.

Lecture 14's sphere, with no equation

lecture 14



$At^2 + Bt + C = 0$

solve an equation: a new one for each shape

today



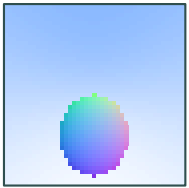
ask d, step d

one loop for every shape

- Lecture 14 (slide 8) solved a quadratic to meet the sphere.
- Today: no equation. Ask how far the nearest surface is, and step that far.

Lecture 14's sphere, with no equation

lecture 14



$At^2 + Bt + C = 0$

solve an equation: a new one for each shape

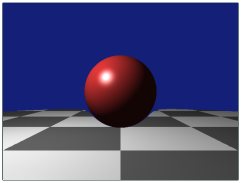
today



ask d, step d

one loop for every shape

the same loop



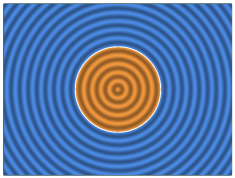
a whole scene

a ball, a floor, a lamp: no triangles

- Lecture 14 (slide 8) solved a quadratic to meet the sphere.
- Today: no equation. Ask how far the nearest surface is, and step that far.
- Any shape with a distance can be drawn: a whole scene in one fragment shader.

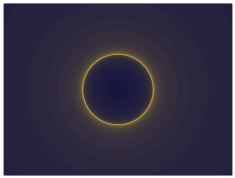
Today: three rounds of slides, then you try

1 **A distance for every point**



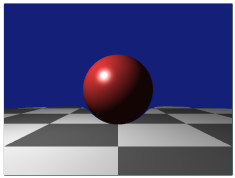
then You Try 1, 2

2 **The march**



then You Try 3, 4

3 **Light and a floor**



then You Try 5, 6

Open the March Lab now. It is on the course calendar, in lecture 33's row.

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

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

Round 1

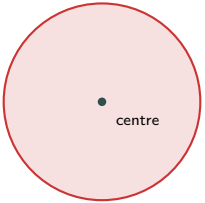
A distance for every point

How far is the nearest surface?

The ball's distance: lecture 26's circle in 3D

lecture 26, slide 18: a circle

```
let d = length(vec2f(x, y)) - r;
```



- Lecture 26: the distance to a circle, negative inside.

The ball's distance: lecture 26's circle in 3D

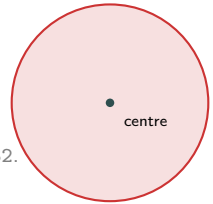
lecture 26, slide 18: a circle

```
let d = length(vec2f(x, y)) - r;
```

today: a ball, in our own function

```
fn ball(p:  vec3f) -> f32 {
    let centre = vec3f(0.0, 0.0, 0.0);
    let r = 0.5;
    return length(p - centre) - r;
}
```

fn: a function we write, as fs_main is.
It gets a point p, returns one number: `-> f32`.



length: built into WGSL, now of three numbers

- Lecture 26: the distance to a circle, negative inside.
- A ball is the same with one more coordinate.

The ball's distance: lecture 26's circle in 3D

lecture 26, slide 18: a circle

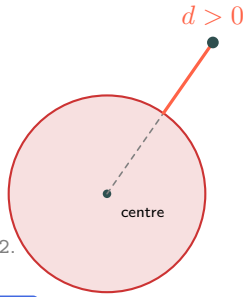
```
let d = length(vec2f(x, y)) - r;
```

today: a ball, in our own function

```
fn ball(p:  vec3f) -> f32 {
    let centre = vec3f(0.0, 0.0, 0.0);
    let r = 0.5;
    return length(p - centre) - r;
}
```

fn: a function we write, as fs_main is.
It gets a point p, returns one number: `-> f32`.

length: built into WGSL, now of three numbers



- Lecture 26: the distance to a circle, negative inside.
- A ball is the same with one more coordinate.
- Outside: how far to the surface.

The ball's distance: lecture 26's circle in 3D

lecture 26, slide 18: a circle

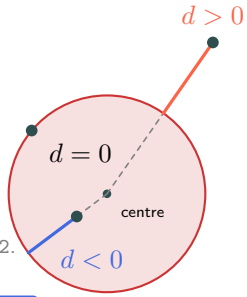
```
let d = length(vec2f(x, y)) - r;
```

today: a ball, in our own function

```
fn ball(p:  vec3f) -> f32 {
    let centre = vec3f(0.0, 0.0, 0.0);
    let r = 0.5;
    return length(p - centre) - r;
}
```

fn: a function we write, as fs_main is.
It gets a point p, returns one number: `-> f32`.

length: built into WGSL, now of three numbers

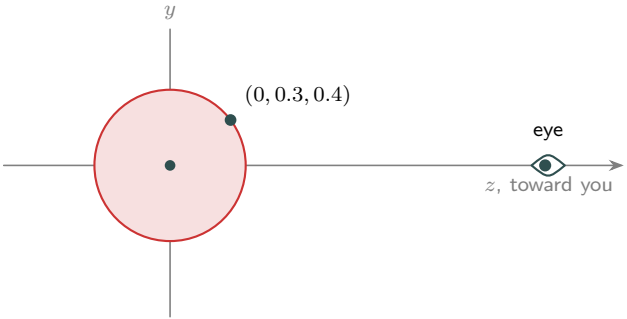


- Lecture 26: the distance to a circle, negative inside.
- A ball is the same with one more coordinate.
- Outside: how far to the surface.
- On the surface 0; inside, negative.

Worked: how far is the ball?

$d = \text{length}(\mathbf{p} - \text{centre}) - r$, centre $(0, 0, 0)$, $r = 0.5$

Try first: d at the eye $(0, 0, 2.5)$, at $(0, 0.3, 0.4)$, at $(0, 0, 0)$

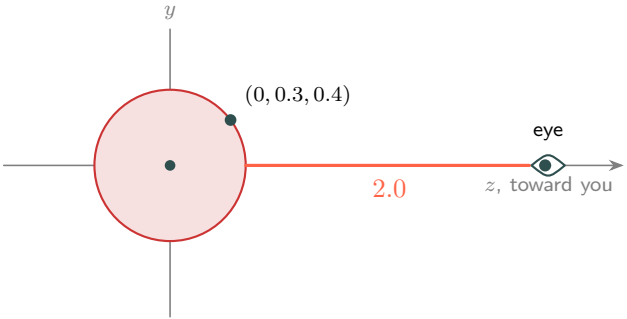


seen from the side

Worked: how far is the ball?

$d = \text{length}(\mathbf{p} - \text{centre}) - r$, centre $(0, 0, 0)$, $r = 0.5$

Try first: d at the eye $(0, 0, 2.5)$, at $(0, 0.3, 0.4)$, at $(0, 0, 0)$



seen from the side

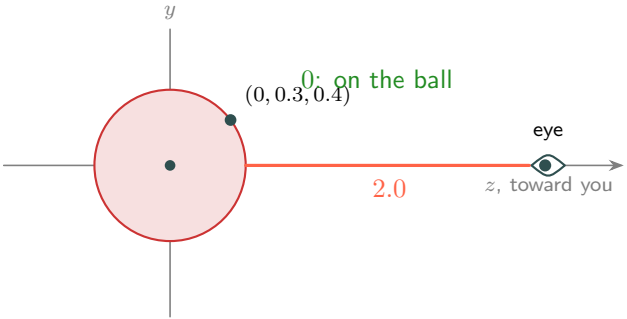
eye: $2.5 - 0.5 = 2.0$

- From the eye, the ball's front is 2.0 away.

Worked: how far is the ball?

$d = \text{length}(\mathbf{p} - \text{centre}) - r$, centre $(0, 0, 0)$, $r = 0.5$

Try first: d at the eye $(0, 0, 2.5)$, at $(0, 0.3, 0.4)$, at $(0, 0, 0)$



seen from the side

eye: $2.5 - 0.5 = 2.0$

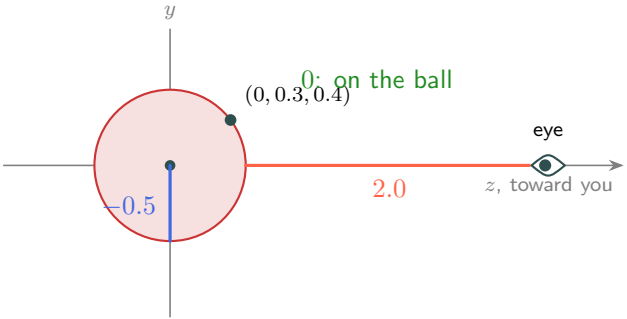
$\sqrt{0.3^2 + 0.4^2} - 0.5 = 0$

- From the eye, the ball's front is 2.0 away.
- $(0, 0.3, 0.4)$ is lecture 27's point: on the ball, so $d = 0$.

Worked: how far is the ball?

$d = \text{length}(\mathbf{p} - \text{centre}) - r$, centre $(0, 0, 0)$, $r = 0.5$

Try first: d at the eye $(0, 0, 2.5)$, at $(0, 0.3, 0.4)$, at $(0, 0, 0)$



- From the eye, the ball's front is 2.0 away.
- $(0, 0.3, 0.4)$ is lecture 27's point: on the ball, so $d = 0$.
- The centre is r inside: -0.5.

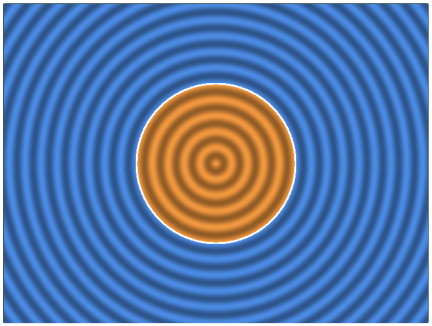
seen from the side

eye: $2.5 - 0.5 = 2.0$ $\sqrt{0.3^2 + 0.4^2} - 0.5 = 0$ centre: $0 - 0.5 = -0.5$

See the distance on a slice

```
let P = vec3f(x, y, 0.0);
let d = ball(P);
c = c * (0.8 + 0.2 * cos(62.8 * d));
```

Each pixel (x, y) is the point $(x, y, 0)$: a flat slice through the ball's middle.



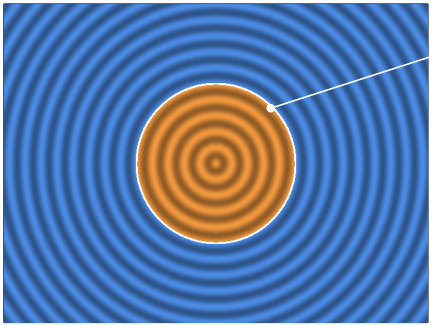
- Colour every pixel of a slice by its distance.

▶ **Run it in the March Lab:**
A slice through the ball

See the distance on a slice

```
let P = vec3f(x, y, 0.0);
let d = ball(P);
c = c * (0.8 + 0.2 * cos(62.8 * d));
```

Each pixel (x, y) is the point $(x, y, 0)$: a flat slice through the ball's middle.



white: $d = 0$
the surface

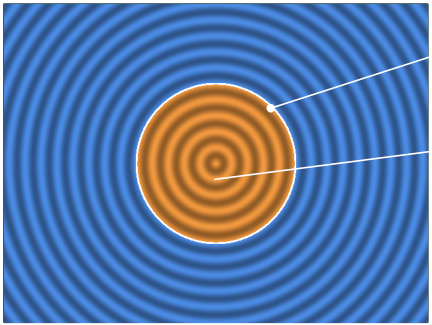
- Colour every pixel of a slice by its distance.
- The white line is where the distance is 0.

▶ **Run it in the March Lab:**
A slice through the ball

See the distance on a slice

```
let P = vec3f(x, y, 0.0);
let d = ball(P);
c = c * (0.8 + 0.2 * cos(62.8 * d));
```

Each pixel (x, y) is the point $(x, y, 0)$: a flat slice through the ball's middle.



white: $d = 0$
the surface

orange: $d < 0$
inside

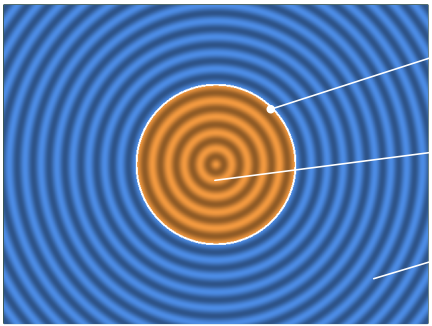
- Colour every pixel of a slice by its distance.
- The white line is where the distance is 0.

▶ **Run it in the March Lab:**
A slice through the ball

See the distance on a slice

```
let P = vec3f(x, y, 0.0);
let d = ball(P);
c = c * (0.8 + 0.2 * cos(62.8 * d));
```

Each pixel (x, y) is the point $(x, y, 0)$: a flat slice through the ball's middle.



white: $d = 0$
the surface

orange: $d < 0$
inside

blue: $d > 0$
a ring for every 0.1

- Colour every pixel of a slice by its distance.
- The white line is where the distance is 0.
- Every point in space has a distance: the scene is a field of numbers.

▶ **Run it in the March Lab:**
A slice through the ball

You Try 1: a bigger ball

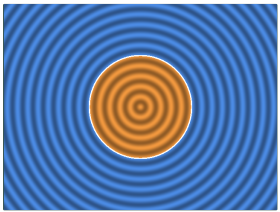
In the March Lab, choose **You Try 1**. Guess first, then change the code.

```
let centre = vec3f(0.0, 0.0, 0.0);
let r = 0.5;
return length(p - centre) - r;
```

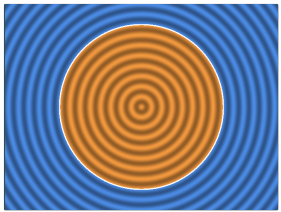
change this line

- **Make the ball bigger: radius 0.8.**
- Guess first: where does the white line go? And the rings?

now



make this



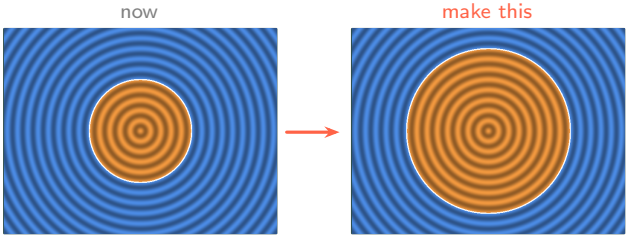
You Try 1: a bigger ball

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

```
let centre = vec3f(0.0, 0.0, 0.0);
let r = 0.5;
return length(p - centre) - r;
```

change this line

answer: `let r = 0.8;`



- **Make the ball bigger: radius 0.8.**
- Guess first: where does the white line go? And the rings?
- Out to 0.8. The rings move with it, still one every 0.1.

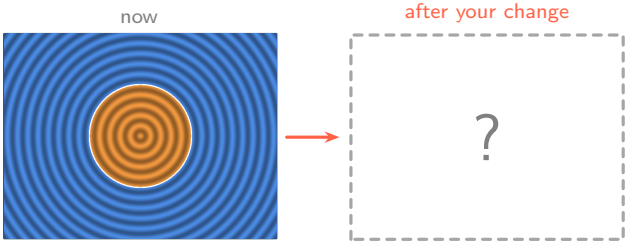
You Try 2: the ball moves toward you

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

```
let centre = vec3f(0.0, 0.0, 0.0);
let r = 0.5;
return length(p - centre) - r;
```

change this line

The slice stays where it is: every pixel is still the point $(x, y, 0)$.



- **Move the ball 0.3 toward you.**
- Before you run it: does the white circle grow, shrink or stay?

You Try 2: the ball moves toward you

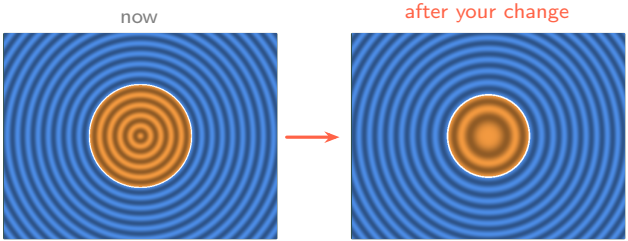
In the March Lab, choose **You Try 2**. Guess first, then change the code.

```
let centre = vec3f(0.0, 0.0, 0.0);
let r = 0.5;
return length(p - centre) - r;
```

change this line

The slice stays where it is: every pixel is still the point $(x, y, 0)$.

answer: `let centre = vec3f(0.0, 0.0, 0.3);`



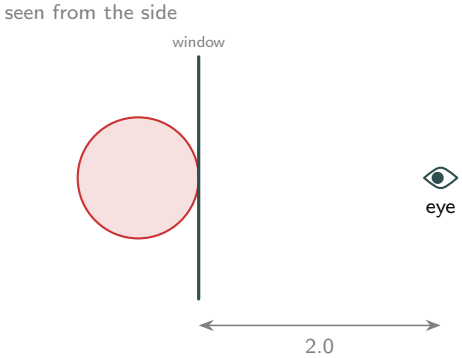
- **Move the ball 0.3 toward you.**
- Before you run it: does the white circle grow, shrink or stay?
- It shrinks to 0.4: the slice now cuts the ball off its middle. $\sqrt{0.5^2 - 0.3^2} = 0.4$.

Round 2

The march

Ask the distance, step that far, repeat

A ray for every pixel

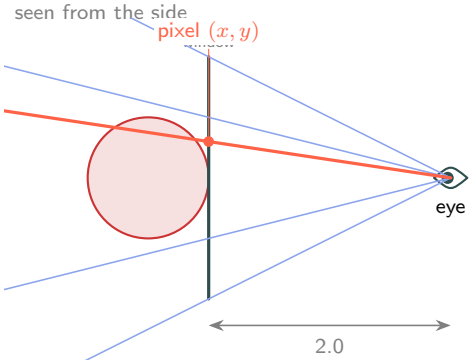



```
let eye = vec3f(0.0, 0.0, 2.5);
```

- Lecture 13's camera: an eye, and a window 2.0 in front.

▶ **Run it in the March Lab:**
Every pixel's ray

A ray for every pixel




```
let eye = vec3f(0.0, 0.0, 2.5);
let dir = normalize(vec3f(x, y, -2.0));
```

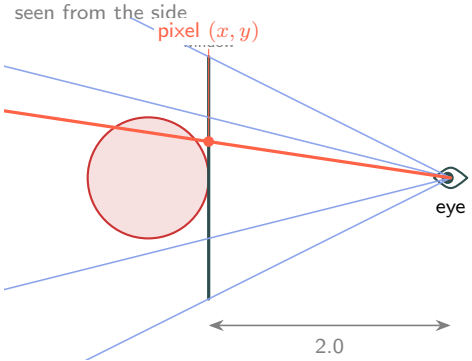
x right, y up, 2.0 into the screen: lecture 13, slide 42.

Every ray starts at the eye: $\mathbf{p} = \text{eye} + t\,\text{dir}$.

- Lecture 13's camera: an eye, and a window 2.0 in front.
- Each pixel's ray leaves the eye through its spot on the window.

▶ **Run it in the March Lab:**
Every pixel's ray

A ray for every pixel





dir as a colour:
red right, green up

- Lecture 13's camera: an eye, and a window 2.0 in front.
- Each pixel's ray leaves the eye through its spot on the window.
- Lecture 27's rays all went straight in. These fan out: perspective.

```
let eye = vec3f(0.0, 0.0, 2.5);
let dir = normalize(vec3f(x, y, -2.0));
```

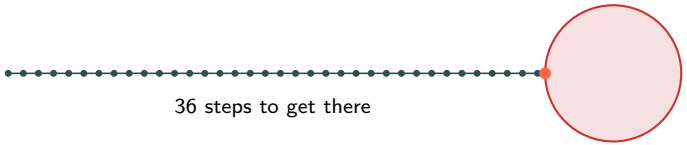
x right, y up, 2.0 into the screen: lecture 13, slide 42.

Every ray starts at the eye: $\mathbf{p} = \text{eye} + t\,\text{dir}$.

▶ **Run it in the March Lab:**
Every pixel's ray

Equal steps: too small is slow, too big jumps through

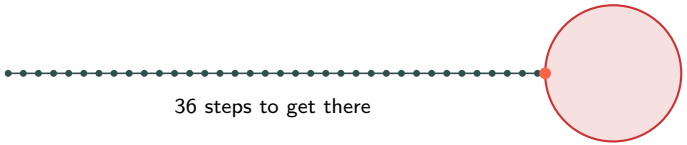
small equal steps



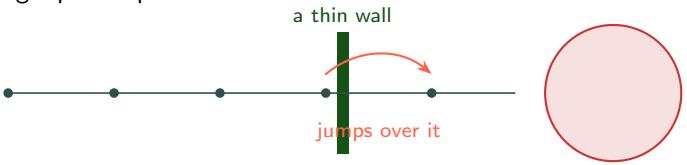
- A ray could walk in small equal steps and stop inside a surface.
- Small steps: right, but slow.

Equal steps: too small is slow, too big jumps through

small equal steps



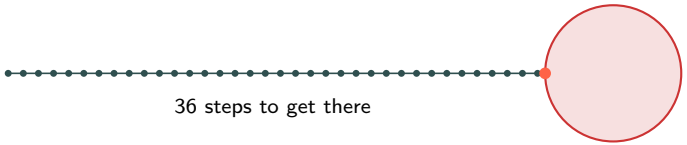
big equal steps



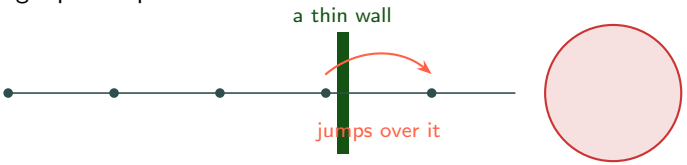
- A ray could walk in small equal steps and stop inside a surface.
- Small steps: right, but slow.
- Big steps: fast, but they jump through a thin shape.

Equal steps: too small is slow, too big jumps through

small equal steps

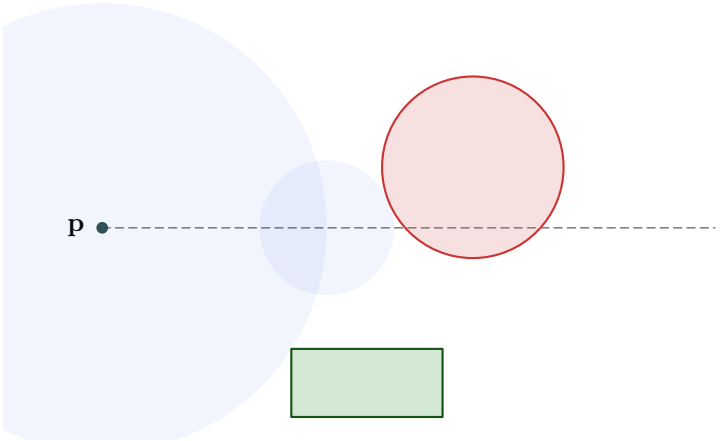


big equal steps



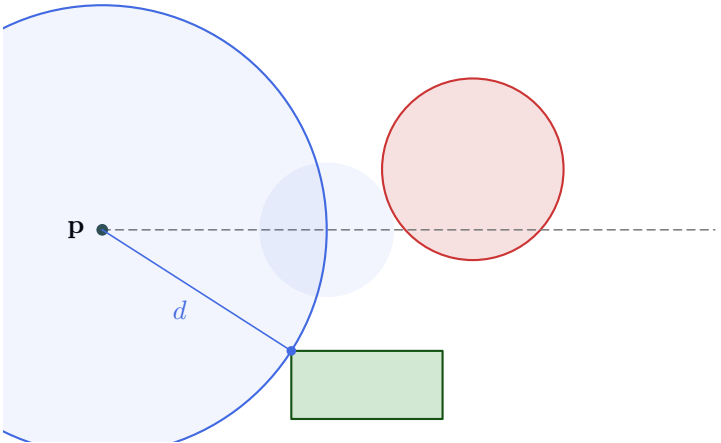
- A ray could walk in small equal steps and stop inside a surface.
- Small steps: right, but slow.
- Big steps: fast, but they jump through a thin shape.
- We want big steps in open space and small ones near a surface.

Step by the distance: nothing is nearer



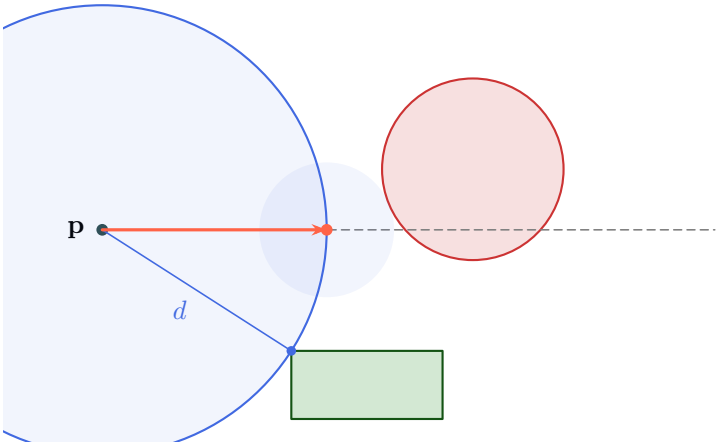
- At $\mathbf{p}$, ask the scene: how far is the nearest surface?

Step by the distance: nothing is nearer



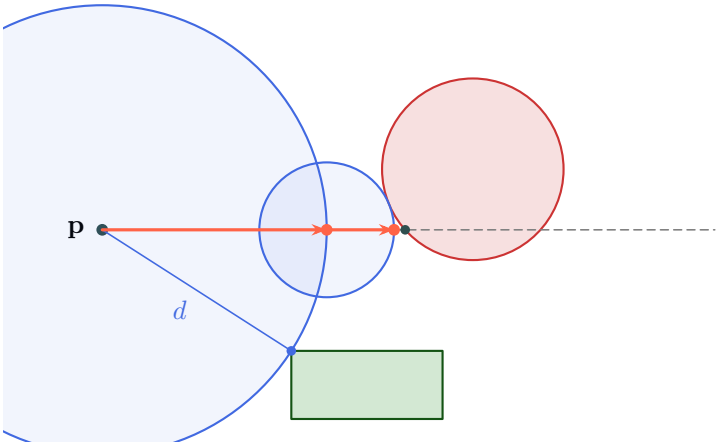
- At $\mathbf{p}$, ask the scene: how far is the nearest surface?
- No surface lies inside the circle of radius d, in any direction.

Step by the distance: nothing is nearer



- At $\mathbf{p}$, ask the scene: how far is the nearest surface?
- No surface lies inside the circle of radius d, in any direction.
- So a step of d along the ray cannot pass through anything.

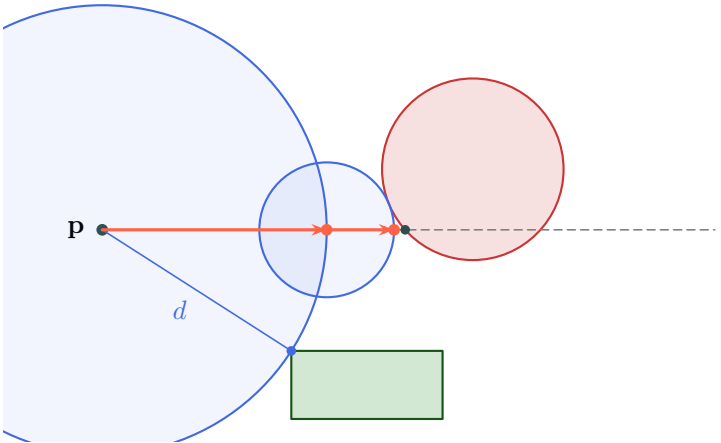
Step by the distance: nothing is nearer



- At $\mathbf{p}$, ask the scene: how far is the nearest surface?
- No surface lies inside the circle of radius d, in any direction.
- So a step of d along the ray cannot pass through anything.
- Near a surface d is small, so the steps get small.

Step by the distance: nothing is nearer

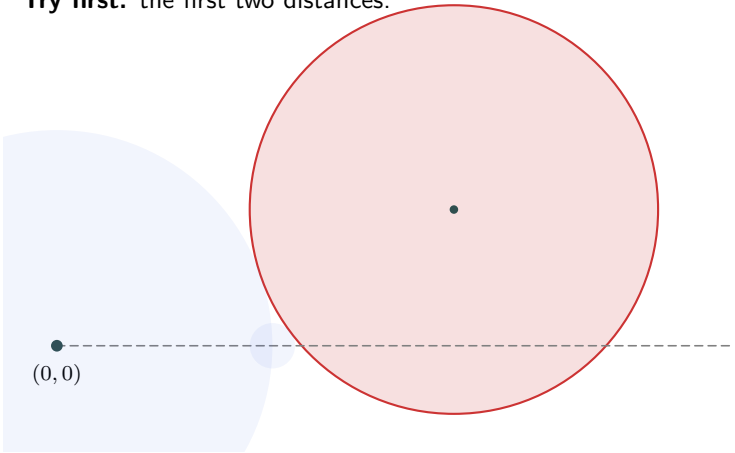
John Hart named it sphere tracing (1996). Most people say ray marching.



- At $\mathbf{p}$, ask the scene: how far is the nearest surface?
- No surface lies inside the circle of radius d, in any direction.
- So a step of d along the ray cannot pass through anything.
- Near a surface d is small, so the steps get small.

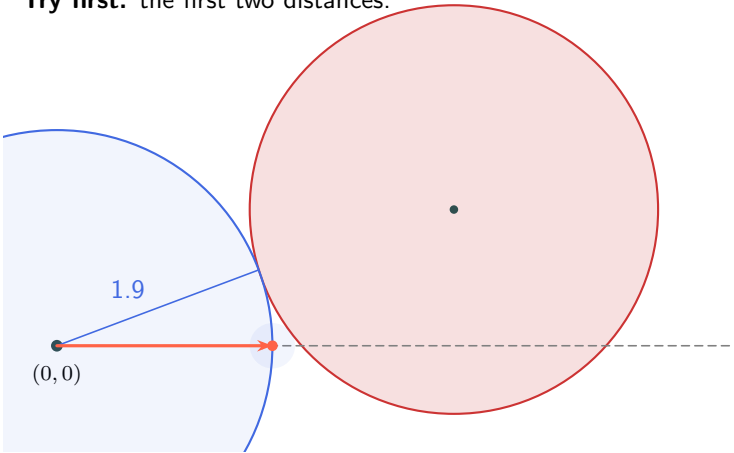
Worked: march toward a ball

Start at $(0, 0)$ and move right. Ball: centre $(3.5, 1.2)$, radius 1.8.
Try first: the first two distances.



Worked: march toward a ball

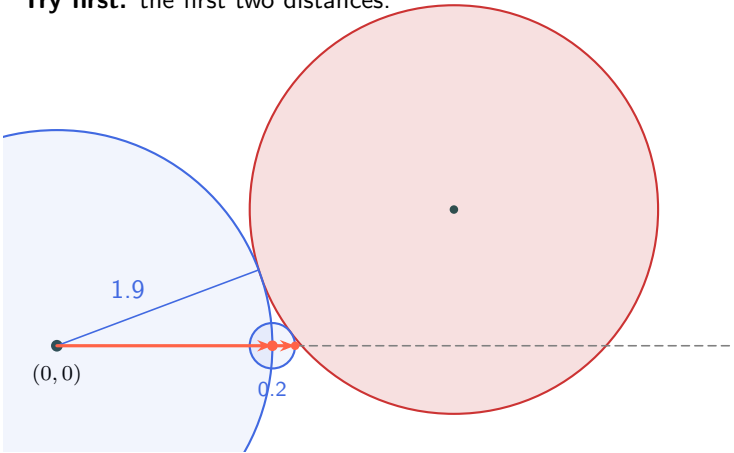
Start at $(0, 0)$ and move right. Ball: centre $(3.5, 1.2)$, radius 1.8.
Try first: the first two distances.



- $\sqrt{3.5^2 + 1.2^2} - 1.8 = 1.9$.
  Step to $(1.9, 0)$.

Worked: march toward a ball

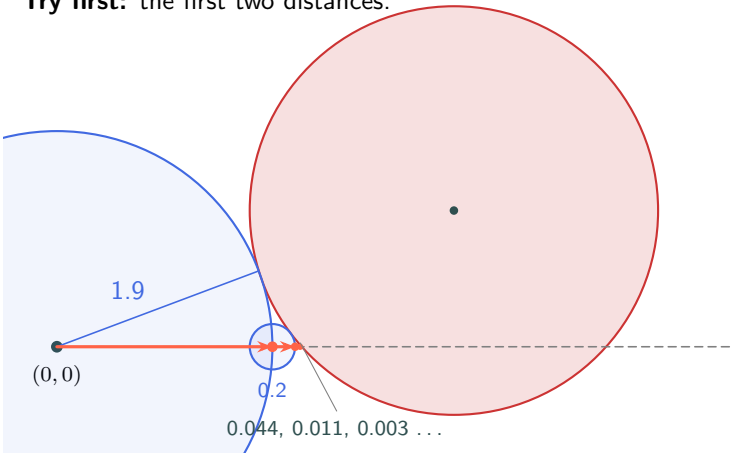
Start at $(0, 0)$ and move right. Ball: centre $(3.5, 1.2)$, radius 1.8.
Try first: the first two distances.



- $\sqrt{3.5^2 + 1.2^2} - 1.8 = 1.9$.
  Step to $(1.9, 0)$.
- $\sqrt{1.6^2 + 1.2^2} - 1.8 = 0.2$.
  Step to $(2.1, 0)$.

Worked: march toward a ball

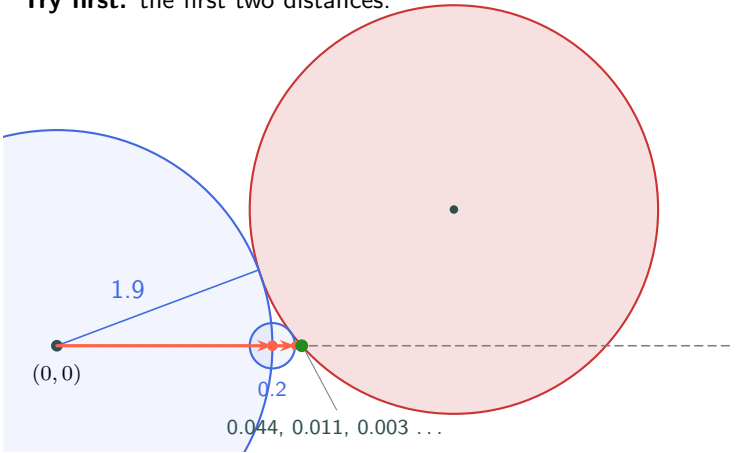
Start at $(0, 0)$ and move right. Ball: centre $(3.5, 1.2)$, radius 1.8.
Try first: the first two distances.



- $\sqrt{3.5^2 + 1.2^2} - 1.8 = 1.9$. Step to $(1.9, 0)$.
- $\sqrt{1.6^2 + 1.2^2} - 1.8 = 0.2$. Step to $(2.1, 0)$.
- The ray slants into the ball, so the distances shrink fast.

Worked: march toward a ball

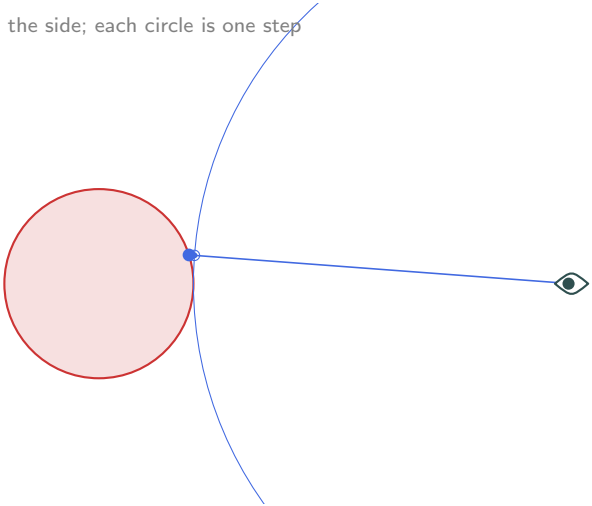
Start at $(0, 0)$ and move right. Ball: centre $(3.5, 1.2)$, radius 1.8.
Try first: the first two distances.



- $\sqrt{3.5^2 + 1.2^2} - 1.8 = 1.9$. Step to $(1.9, 0)$.
- $\sqrt{1.6^2 + 1.2^2} - 1.8 = 0.2$. Step to $(2.1, 0)$.
- The ray slants into the ball, so the distances shrink fast.
- Step 6: $d = 0.0007$, below 0.001. A hit at $x = 2.158$.

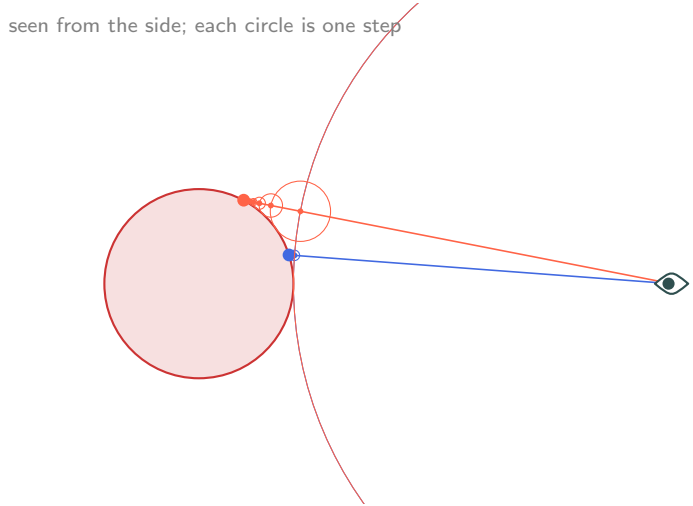
Three rays through our scene

seen from the side; each circle is one step



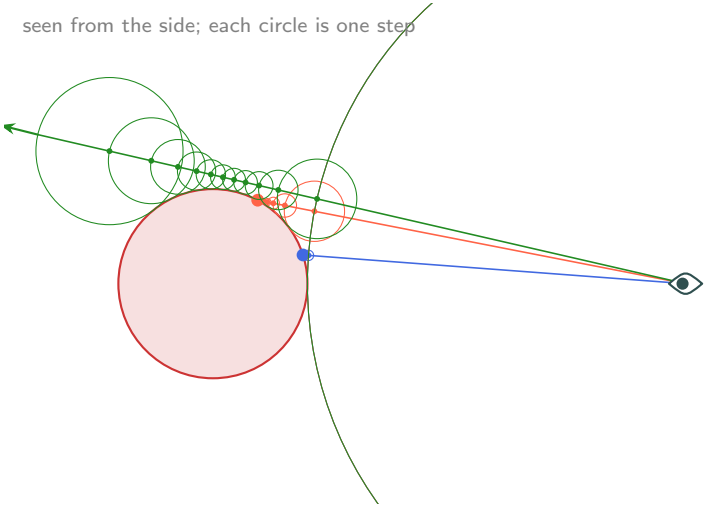
- Straight at the ball: 4 steps. The first is 2.0 long.

Three rays through our scene



- Straight at the ball: 4 steps. The first is 2.0 long.
- Close past the edge: 13 steps, each smaller than the last.

Three rays through our scene

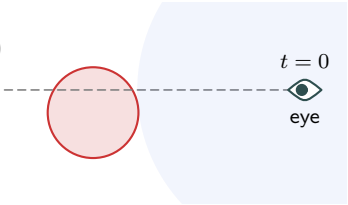


- Straight at the ball: 4 steps. The first is 2.0 long.
- Close past the edge: 13 steps, each smaller than the last.
- Just over the top: the circles shrink, then grow again. The ray is getting away: a miss.

The march in code

```
var t = 0.0;
var hit = false;
```

var: a value that may change (a let may not)



- t: how far along the ray we are. It starts at 0.

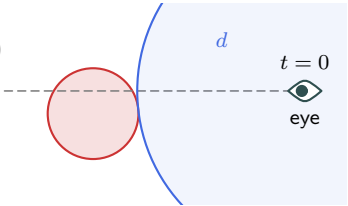
The march in code

```
var t = 0.0;
var hit = false;
for (var i = 0; i < 64; i++) {
    let d = scene(eye + t * dir);


}
```

var: a value that may change (a let may not)

repeat, at most 64 times

scene: see slide 25



- t: how far along the ray we are. It starts at 0.

The march in code

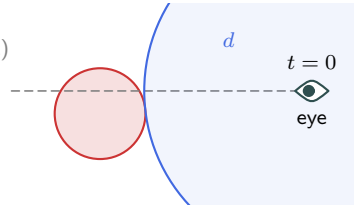
```
var t = 0.0;
var hit = false;
for (var i = 0; i < 64; i++) {
    let d = scene(eye + t * dir);
    if (d < 0.001) {
        hit = true;
        break;
    }



}
```

var: a value that may change (a let may not)

repeat, at most 64 times

scene: see slide 25

break: leave the loop now



- t: how far along the ray we are. It starts at 0.

The march in code

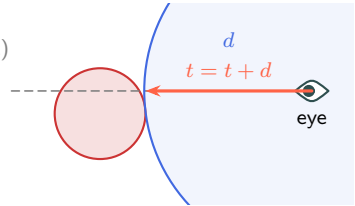
```
var t = 0.0;
var hit = false;
for (var i = 0; i < 64; i++) {
    let d = scene(eye + t * dir);
    if (d < 0.001) {
        hit = true;
        break;
    }
    t = t + d;


}
```

var: a value that may change (a let may not)

repeat, at most 64 times

scene: see slide 25

break: leave the loop now



- t: how far along the ray we are. It starts at 0.
- Each turn: ask d, then step d.

The march in code

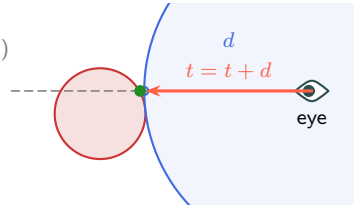
```
var t = 0.0;
var hit = false;
for (var i = 0; i < 64; i++) {
    let d = scene(eye + t * dir);
    if (d < 0.001) {
        hit = true;
        break;
    }
    t = t + d;
    if (t > 20.0) {
        break;
    }
}
```

var: a value that may change (a let may not)

repeat, at most 64 times

scene: see slide 25

break: leave the loop now



- t: how far along the ray we are. It starts at 0.
- Each turn: ask d, then step d.
- Stop on a hit, or when t passes 20: a miss.

The march in code

```
var t = 0.0;
var hit = false;
for (var i = 0; i < 64; i++) {
    let d = scene(eye + t * dir);
    if (d < 0.001) {
        hit = true;
        break;
    }
    t = t + d;
    if (t > 20.0) {
        break;
    }
}
```

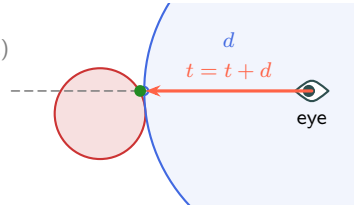
var: a value that may change (a let may not)

repeat, at most 64 times

scene: see slide 25

break: leave the loop now

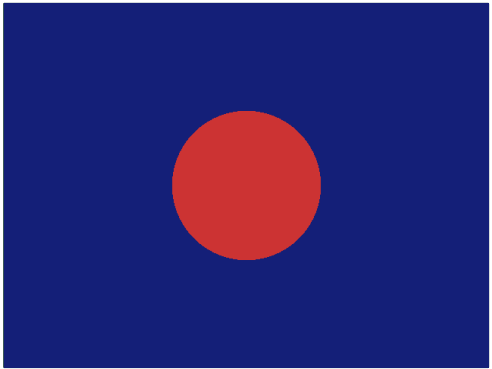
After the loop: `if (!hit) { return sky }`. !hit means "not hit".



- t: how far along the ray we are. It starts at 0.
- Each turn: ask d, then step d.
- Stop on a hit, or when t passes 20: a miss.

The first picture

```
if (!hit) { return vec4f(sky.r, sky.g, sky.b, 1.0); }
let c = base;       // every hit:  the same red
```

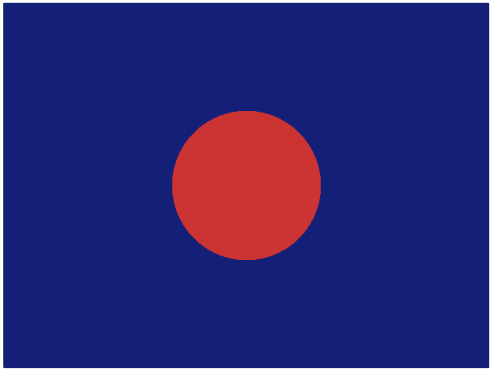


- Red where the march reached the ball; sky where it gave up.

▶ **Run it in the March Lab:**
The march: hit or miss

The first picture

```
if (!hit) { return vec4f(sky.r, sky.g, sky.b, 1.0); }
let c = base;       // every hit:  the same red
```

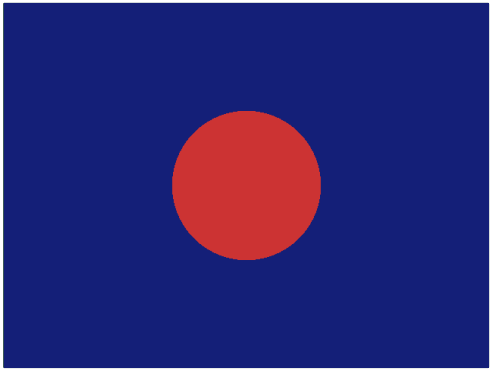


- Red where the march reached the ball; sky where it gave up.
- The same disc as lecture 14's first picture (slide 18) and lecture 27's flat disc.

▶ **Run it in the March Lab:**
The march: hit or miss

The first picture

```
if (!hit) { return vec4f(sky.r, sky.g, sky.b, 1.0); }
let c = base;       // every hit:  the same red
```

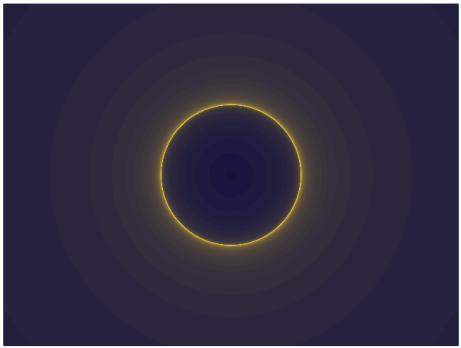


- Red where the march reached the ball; sky where it gave up.
- The same disc as lecture 14's first picture (slide 18) and lecture 27's flat disc.
- No equation was solved: only distances were asked.

▶ **Run it in the March Lab:**
The march: hit or miss

Colour by the number of steps

```
steps = steps + 1.0;                inside the loop: once each turn
let k = steps / 64.0;
let c = mix(vec3f(0.05, 0.05, 0.25), vec3f(1.0, 0.85, 0.2), k);
```

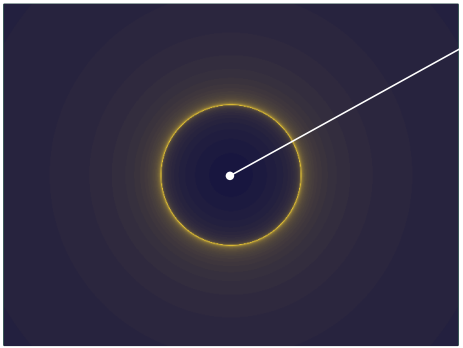


- Yellow pixels took many steps; dark ones took few.

▶ **Run it in the March Lab:**
The number of steps

Colour by the number of steps

```
steps = steps + 1.0;          inside the loop: once each turn
let k = steps / 64.0;
let c = mix(vec3f(0.05, 0.05, 0.25), vec3f(1.0, 0.85, 0.2), k);
```



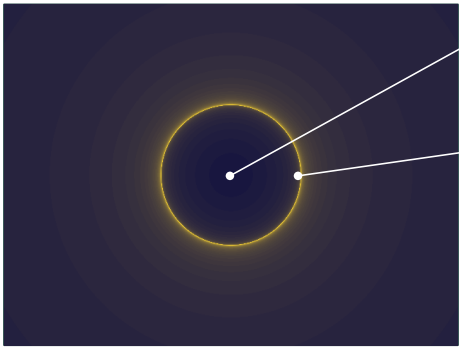
the middle: 2 steps

- Yellow pixels took many steps; dark ones took few.

▶ **Run it in the March Lab:**
The number of steps

Colour by the number of steps

```
steps = steps + 1.0;          inside the loop: once each turn
let k = steps / 64.0;
let c = mix(vec3f(0.05, 0.05, 0.25), vec3f(1.0, 0.85, 0.2), k);
```



the middle: 2 steps

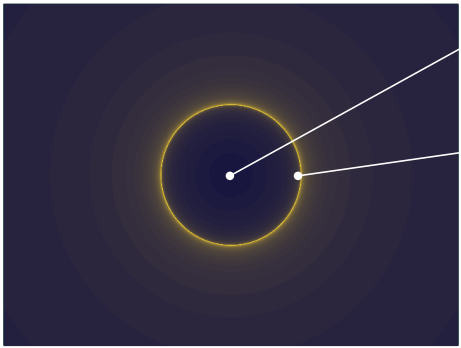
the edge: 17 steps;
just outside: 24

- Yellow pixels took many steps; dark ones took few.
- Rays that pass close to the edge take many tiny steps.

▶ **Run it in the March Lab:**
The number of steps

Colour by the number of steps

```
steps = steps + 1.0;          inside the loop: once each turn
let k = steps / 64.0;
let c = mix(vec3f(0.05, 0.05, 0.25), vec3f(1.0, 0.85, 0.2), k);
```



the middle: 2 steps

the edge: 17 steps;
just outside: 24

the whole picture:
8.8 steps a pixel

- Yellow pixels took many steps; dark ones took few.
- Rays that pass close to the edge take many tiny steps.
- Measured: 8.8 steps a pixel on average, 64 at most.

▶ **Run it in the March Lab:**
The number of steps

You Try 3: only 8 steps

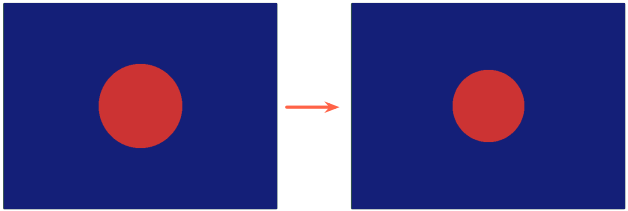
In the March Lab, choose **You Try 3**. Guess first, then change the code.

```
var hit = false;
for (var i = 0; i < 64; i++) {
    let d = scene(eye + t * dir);
```

change this line

- **Allow only 8 steps.**
- Guess first: which part of the ball goes first, the middle or the edge?

now



make this

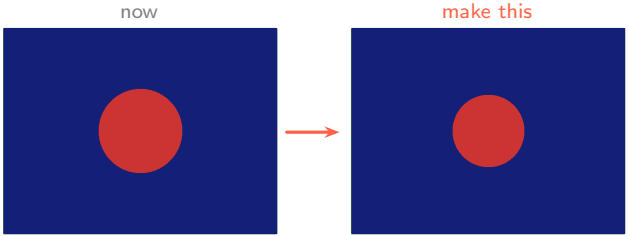
You Try 3: only 8 steps

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

```
var hit = false;
for (var i = 0; i < 64; i++) {
    let d = scene(eye + t * dir);
```

change this line

answer: `for (var i = 0; i < 8; i++) {`



- **Allow only 8 steps.**
- Guess first: which part of the ball goes first, the middle or the edge?
- The edge: its rays need 17 steps or more. The middle needs 2.

You Try 4: "very close" is 0.1

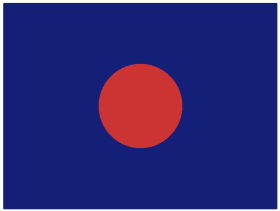
In the March Lab, choose **You Try 4**. Guess first, then change the code.

```
let d = scene(eye + t * dir);
if (d < 0.001) {
    hit = true;
```

change this line

- **Let "very close" mean 0.1 instead of 0.001.**
- Before you run it: is the ball bigger, smaller, or the same?

now



after your change



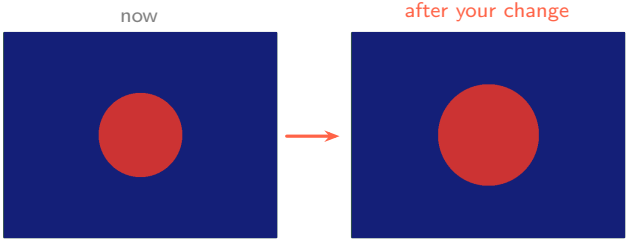
You Try 4: "very close" is 0.1

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

```
let d = scene(eye + t * dir);
if (d < 0.001) {
    hit = true;
```

change this line

answer: `if (d < 0.1) {`



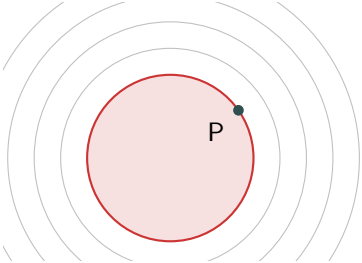
- **Let "very close" mean 0.1 instead of 0.001.**
- Before you run it: is the ball bigger, smaller, or the same?
- Bigger: a ray that passes within 0.1 of the ball now counts as a hit.

Round 3

Light and a floor

the normal from the distance; two shapes with min

The normal from the distance



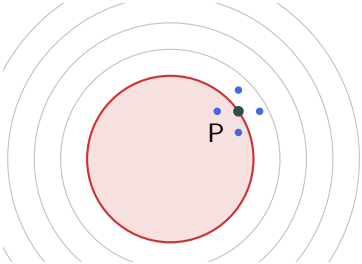
gray rings: equal distance

```
fn normal(p:  vec3f) -> vec3f {
    let e = 0.001;



}
```

- Lecture 14 knew the sphere's normal. A march knows only distances.

▶ **Run it in the March Lab:**
The normal as a colour

The normal from the distance



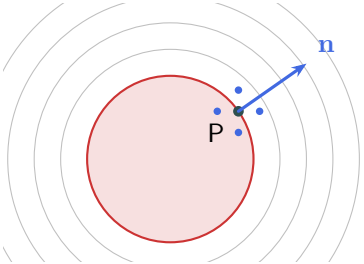
gray rings: equal distance

```
fn normal(p:  vec3f) -> vec3f {
    let e = 0.001;
    let dx = scene(p + vec3f(e, 0.0, 0.0)) - scene(p - vec3f(e, 0.0, 0.0));
    let dy = scene(p + vec3f(0.0, e, 0.0)) - scene(p - vec3f(0.0, e, 0.0));
    let dz = scene(p + vec3f(0.0, 0.0, e)) - scene(p - vec3f(0.0, 0.0, e));

}
```

- Lecture 14 knew the sphere's normal. A march knows only distances.
- Step a little each way and ask the distance: dx, dy, dz.

▶ **Run it in the March Lab:**
The normal as a colour

The normal from the distance



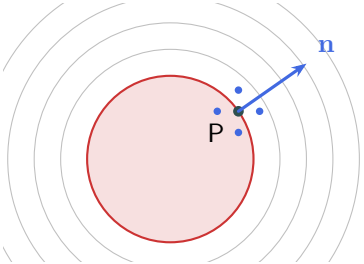
gray rings: equal distance

```
fn normal(p:  vec3f) -> vec3f {
    let e = 0.001;
    let dx = scene(p + vec3f(e, 0.0, 0.0)) - scene(p - vec3f(e, 0.0, 0.0));
    let dy = scene(p + vec3f(0.0, e, 0.0)) - scene(p - vec3f(0.0, e, 0.0));
    let dz = scene(p + vec3f(0.0, 0.0, e)) - scene(p - vec3f(0.0, 0.0, e));
    return normalize(vec3f(dx, dy, dz));
}
```

- Lecture 14 knew the sphere's normal. A march knows only distances.
- Step a little each way and ask the distance: dx, dy, dz.
- The distance grows fastest straight out from the surface: that way is $\mathbf{n}$.

▶ **Run it in the March Lab:**
The normal as a colour

The normal from the distance



gray rings: equal distance

as a colour: lecture 27, slide 8's colours



```
fn normal(p:  vec3f) -> vec3f {
    let e = 0.001;
    let dx = scene(p + vec3f(e, 0.0, 0.0)) - scene(p - vec3f(e, 0.0, 0.0));
    let dy = scene(p + vec3f(0.0, e, 0.0)) - scene(p - vec3f(0.0, e, 0.0));
    let dz = scene(p + vec3f(0.0, 0.0, e)) - scene(p - vec3f(0.0, 0.0, e));
    return normalize(vec3f(dx, dy, dz));
}
```

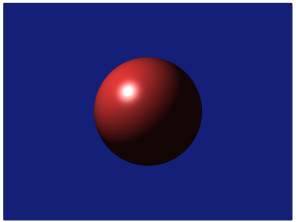
- Lecture 14 knew the sphere's normal. A march knows only distances.
- Step a little each way and ask the distance: dx, dy, dz.
- The distance grows fastest straight out from the surface: that way is $\mathbf{n}$.

▶ **Run it in the March Lab:**
The normal as a colour

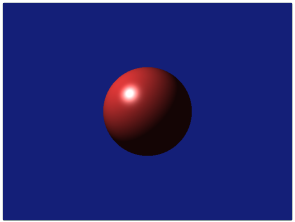
Lecture 27's light, line for line

```
let n = normal(P);
let l = normalize(lamp - P);
let diffuse = max(dot(n, l), 0.0);
let v = -dir;
let h = normalize(l + v);
let shine = pow(max(dot(n, h), 0.0), 50.0);
let c = base * (ambient + diffuse) + shine;
```

lecture 27: a fake ball



today: marched



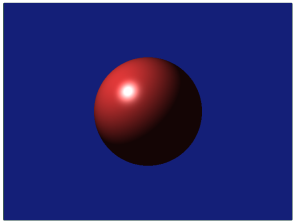
- From $\mathbf{n}$ on, nothing changes: lecture 27's light.

▶ **Run it in the March Lab:**
Lecture 27's light

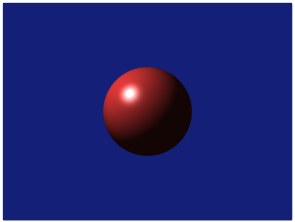
Lecture 27's light, line for line

```
let n = normal(P);
let l = normalize(lamp - P);
let diffuse = max(dot(n, l), 0.0);
let v = -dir;      lecture 27: vec3f(0.0, 0.0, 1.0)
let h = normalize(l + v);
let shine = pow(max(dot(n, h), 0.0), 50.0);
let c = base * (ambient + diffuse) + shine;
```

lecture 27: a fake ball



today: marched



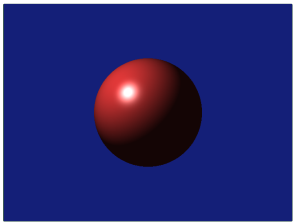
- From $\mathbf{n}$ on, nothing changes: lecture 27's light.
- One line differs: $\mathbf{v}$ points back along this pixel's own ray.

▶ **Run it in the March Lab:**
Lecture 27's light

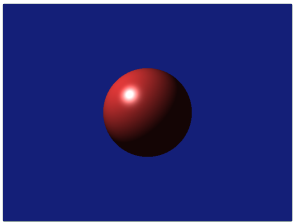
Lecture 27's light, line for line

```
let n = normal(P);
let l = normalize(lamp - P);
let diffuse = max(dot(n, l), 0.0);
let v = -dir;      lecture 27: vec3f(0.0, 0.0, 1.0)
let h = normalize(l + v);
let shine = pow(max(dot(n, h), 0.0), 50.0);
let c = base * (ambient + diffuse) + shine;
```

lecture 27: a fake ball



today: marched

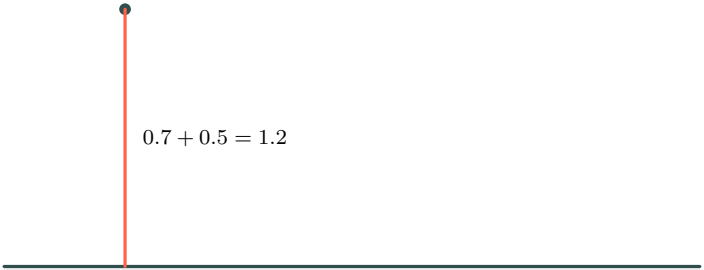


- From $\mathbf{n}$ on, nothing changes: lecture 27's light.
- One line differs: $\mathbf{v}$ points back along this pixel's own ray.
- The two balls look alike. Only today's ball sits in a real 3D scene.

▶ **Run it in the March Lab:**
Lecture 27's light

A floor's distance is a height

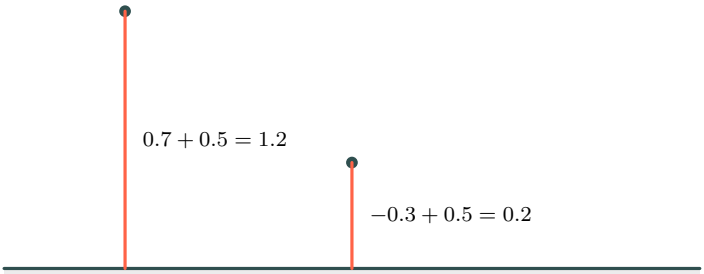
```
fn ground(p:  vec3f) -> f32 {
    return p.y + 0.5;
}
```



- The nearest point of a flat floor is straight below.

A floor's distance is a height

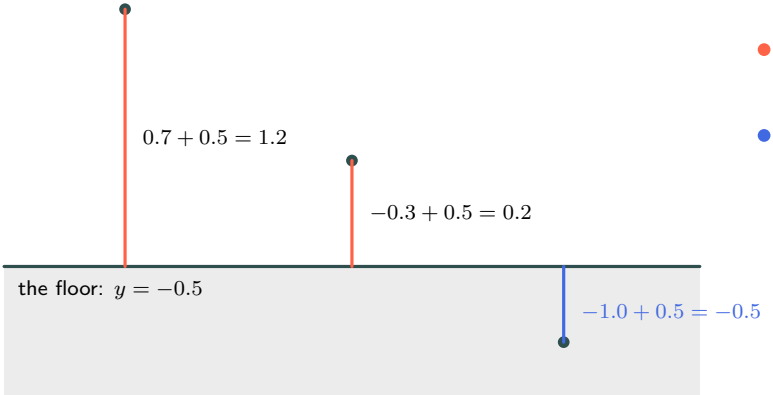
```
fn ground(p:  vec3f) -> f32 {
    return p.y + 0.5;
}
```



- The nearest point of a flat floor is straight below.
- So the distance is the height above it: $y + 0.5$.

A floor's distance is a height

```
fn ground(p:  vec3f) -> f32 {
    return p.y + 0.5;
}
```

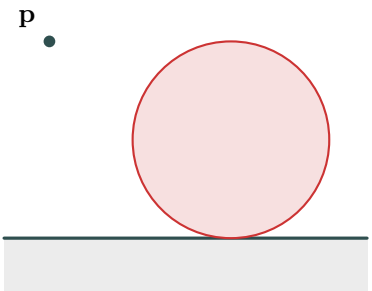


- The nearest point of a flat floor is straight below.
- So the distance is the height above it: $y + 0.5$.
- Under the floor it is negative, as inside the ball.

Two shapes: the smaller distance wins

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

min: built into WGSL, the smaller of two



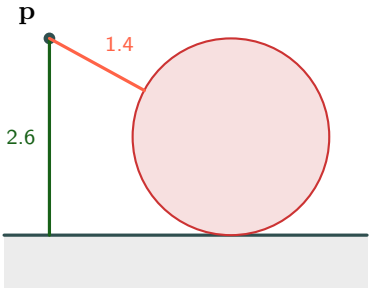
- Two shapes: each has its own distance.

▶ **Run it in the March Lab:**
A ball on a floor

Two shapes: the smaller distance wins

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

min: built into WGSL, the smaller of two



- Two shapes: each has its own distance.

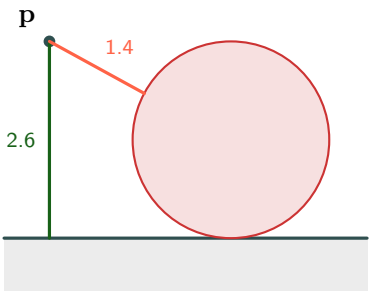
▶ **Run it in the March Lab:**
A ball on a floor

Two shapes: the smaller distance wins

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

min: built into WGSL, the smaller of two

$\min(1.4,\ 2.6) = 1.4$: safe from both



- Two shapes: each has its own distance.
- The scene's distance is the smaller one: the nearer surface.

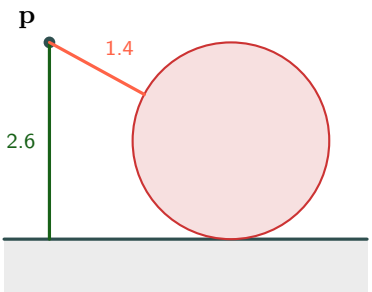
▶ **Run it in the March Lab:**
A ball on a floor

Two shapes: the smaller distance wins

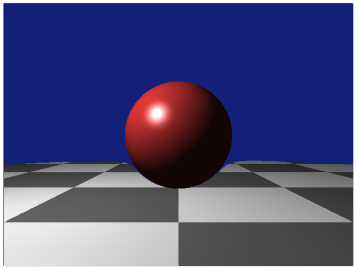
```
fn scene(p:  vec3f) -> f32 {
    return min(ball(p), ground(p));
}
```

min: built into WGSL, the smaller of two

$\min(1.4,\ 2.6) = 1.4$: safe from both




```
if (ground(P) < 0.01) { base = checks }
```



- Two shapes: each has its own distance.
- The scene's distance is the smaller one: the nearer surface.
- Where the floor's distance is 0, the hit is on the floor: paint it in checks.

▶ **Run it in the March Lab:**
A ball on a floor

You Try 5: put the floor in

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

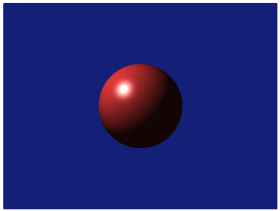
```
fn scene(p:  vec3f) -> f32 {
    return ball(p);
}
```

change this line

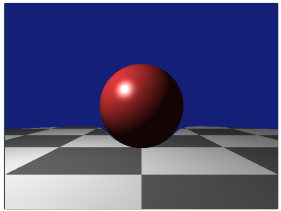
ground(p) is already in the file, under ball(p).

- **Put the floor into the scene.**
- Guess first: which of the two distances must scene return?

now



make this



You Try 5: put the floor in

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

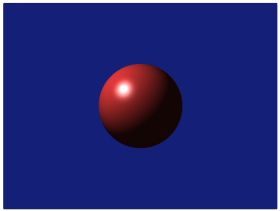
```
fn scene(p:  vec3f) -> f32 {
    return ball(p);
}
```

change this line

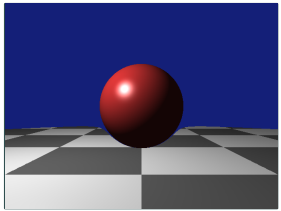
ground(p) is already in the file, under ball(p).

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

now



make this



- **Put the floor into the scene.**
- Guess first: which of the two distances must scene return?
- The smaller. The checks come by themselves: their lines were already there.

You Try 6: a second ball

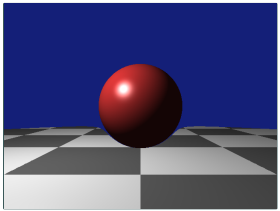
In the March Lab, choose **You Try 6**. Guess first, then change the code.

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

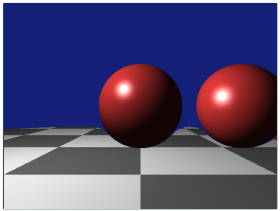
change this line

- **Add a second ball, radius 0.5, 1.2 to the right.**
- Guess first: what colour will the new ball be?

now



make this



You Try 6: a second ball

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

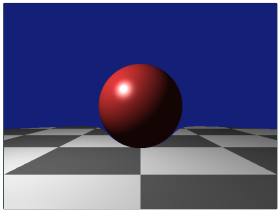
```
fn scene(p:  vec3f) -> f32 {
    return min(ball(p), ground(p));
}
```

change this line

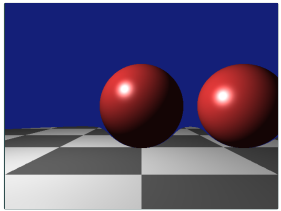
answer:

```
let other = length(p - vec3f(1.2, 0.0, 0.0)) - 0.5;
return min(min(ball(p), ground(p)), other);
```

now



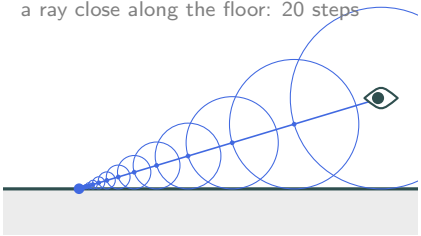
make this



- **Add a second ball, radius 0.5, 1.2 to the right.**
- Guess first: what colour will the new ball be?
- Red: only the floor test changes the colour, and the new ball is not the floor.

What the march gets wrong: rays that graze a surface

a ray close along the floor: 20 steps

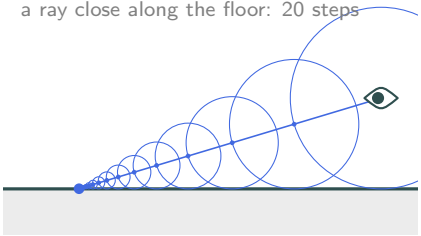


64 steps, coloured by count

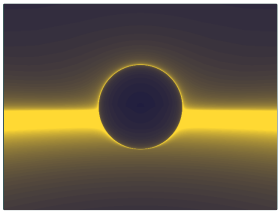
- A ray close along the floor stays near it: every step is tiny.

What the march gets wrong: rays that graze a surface

a ray close along the floor: 20 steps



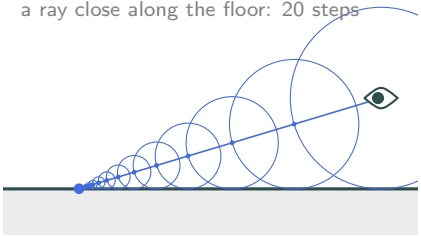
64 steps, coloured by count



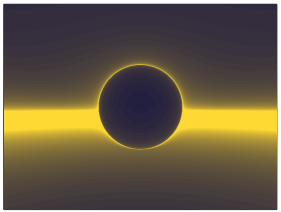
- A ray close along the floor stays near it: every step is tiny.
- Near the horizon, 6% of the pixels use all 64 steps.

What the march gets wrong: rays that graze a surface

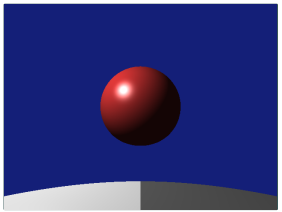
a ray close along the floor: 20 steps



64 steps, coloured by count



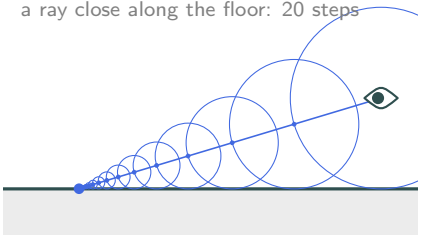
only 16 steps



- A ray close along the floor stays near it: every step is tiny.
- Near the horizon, 6% of the pixels use all 64 steps.
- With 16 steps, the far floor is lost: those rays show the sky.

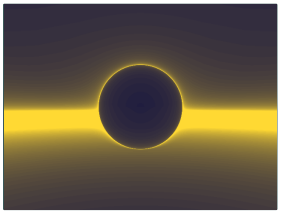
What the march gets wrong: rays that graze a surface

a ray close along the floor: 20 steps

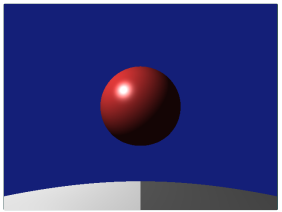


steps a pixel, measured:
ball alone 8.8
ball and floor 21.9

64 steps, coloured by count

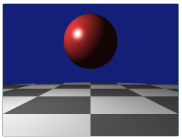


only 16 steps



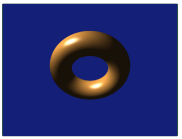
- A ray close along the floor stays near it: every step is tiny.
- Near the horizon, 6% of the pixels use all 64 steps.
- With 16 steps, the far floor is lost: those rays show the sky.
- Lecture 14's formula had no step limit.

Now make your own picture



A bouncing ball

`abs(sin(3.0 * u.time))`



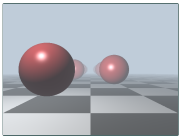
A spinning doughnut

`length(ring) - 0.2`



A chrome ball

`march again: reflect`



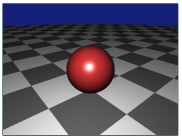
Balls in the fog

`mix(lit, sky, haze)`



A glowing halo

`the step count as light`



Fly round with the mouse

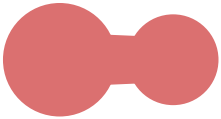
`lecture 13's camera frame`

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

Summary

1	A distance	`length(p - centre) - r` every point in space has one; negative inside
2	A ray for every pixel	`normalize(vec3f(x, y, -2.0))` lecture 13's camera: rays fan out from the eye
3	The march	ask d, step d, repeat a hit below 0.001, a miss past 20; nothing is nearer than d
4	Normal, light, more shapes	ask a little each way; `min` of two distances from $\mathbf{n}$ on: lecture 27's light, with $\mathbf{v} = -\text{dir}$
5	What it gets wrong	grazing rays run out of steps the far floor is lost when the step limit is too low

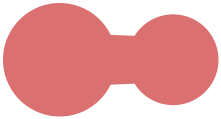
Next lecture: shapes that melt, repeat and cast shadows



melt

- A smooth min: two shapes melt into one.

Next lecture: shapes that melt, repeat and cast shadows



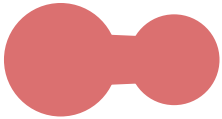
melt



repeat

- A smooth min: two shapes melt into one.
- Bend the space, not the shape: lecture 26's fract repeats one ball forever.

Next lecture: shapes that melt, repeat and cast shadows



melt



repeat



a soft shadow

- A smooth min: two shapes melt into one.
- Bend the space, not the shape: lecture 26's fract repeats one ball forever.
- March toward the lamp: lecture 32's soft shadow, from one ray.

Exit Ticket

Today's question *(2 minutes)*

(a) A ray starts at $(0, 0)$ and moves right. A ball has centre $(4, 3)$ and radius 1. March two steps: write each distance.

(b) Find the third distance. Is the ray closing in on the ball, or getting away?

CS 453 — Exit Ticket #33

Name: _____

Roll #: _____ Date: _____

Answer:

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