Lecture 28: Into 3D

The Vertex Shader, the MVP Chain, the Depth Buffer

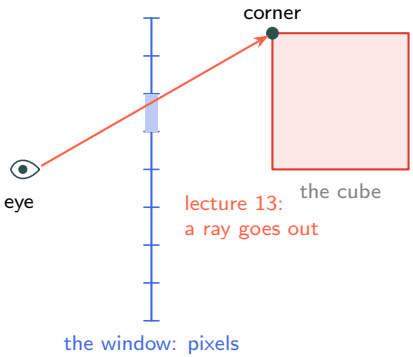
CS 453 — Computer Graphics

Fakhir Shaheen

Forman Christian University

Lecture 13's camera, run backwards

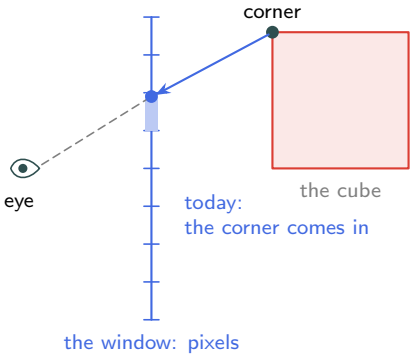
seen from the side



- Part A: each pixel sends a ray into the scene.

Lecture 13's camera, run backwards

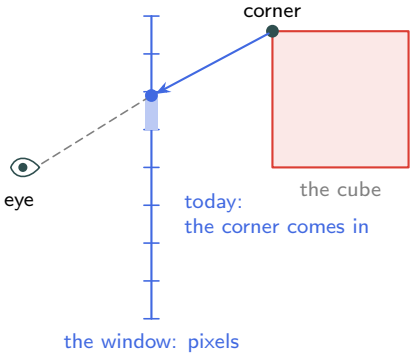
seen from the side

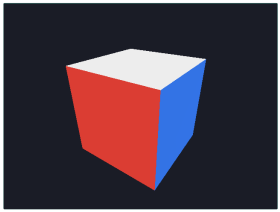


- Part A: each pixel sends a ray into the scene.
- Today: each corner finds its pixel.

Lecture 13's camera, run backwards

seen from the side





36 corners, then the rasterizer

- Part A: each pixel sends a ray into the scene.
- Today: each corner finds its pixel.
- The rasterizer fills in between (lecture 24).

Today: three rounds of slides, then you try

1 **Turn the cube**



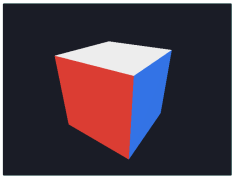
then You Try 1, 2

2 **What is in front?**



then You Try 3, 4

3 **Far things look smaller**



then You Try 5, 6

Open the Cube Lab now. It is on the course calendar, in lecture 28's row.

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

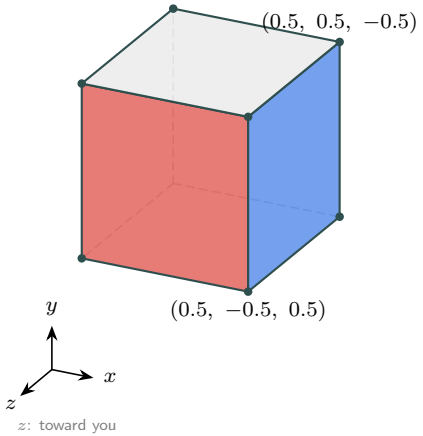
A slide with a blue **Run it in the Cube Lab** box has its code in the lab too.

Round 1

Turn the cube

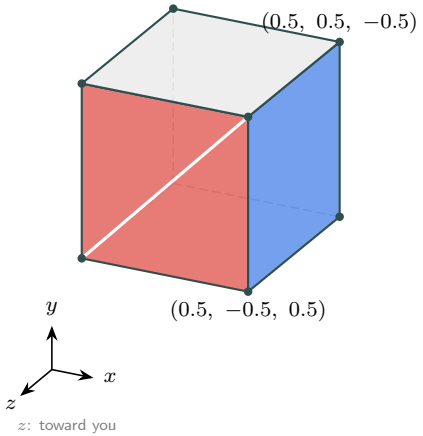
vs_main moves every corner

A cube: 6 faces, 12 triangles, 36 corners



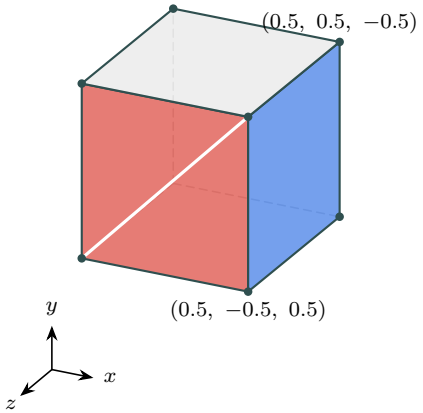
- Corners at ± 0.5; the middle is $(0, 0, 0)$.

A cube: 6 faces, 12 triangles, 36 corners



- Corners at ± 0.5; the middle is $(0, 0, 0)$.
- A face is 2 triangles: 6 corners.

A cube: 6 faces, 12 triangles, 36 corners



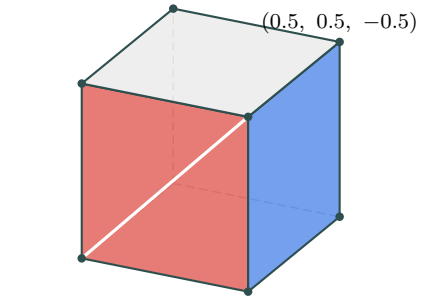
the vertex buffer, in drawing order

1 front	6 corners	
2 back	6 corners	
3 right	6 corners	
4 left	6 corners	36
5 top	6 corners	
6 bottom	6 corners	

a corner: x y z, red green blue

- Corners at ± 0.5; the middle is $(0, 0, 0)$.
- A face is 2 triangles: 6 corners.
- Lecture 25's buffer, with z added.

A cube: 6 faces, 12 triangles, 36 corners



z: toward you

the vertex buffer, in drawing order

1 front	6 corners	36
2 back	6 corners	
3 right	6 corners	
4 left	6 corners	
5 top	6 corners	
6 bottom	6 corners	

a corner: x y z, red green blue

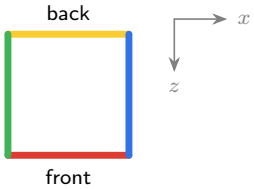
```
@location(0) xyz:  vec3f
@location(1) colour:  vec3f
```

vs_main's two inputs

- Corners at ± 0.5; the middle is $(0, 0, 0)$.
- A face is 2 triangles: 6 corners.
- Lecture 25's buffer, with z added.
- Each corner carries its face's colour.

Straight on: vs_main hands on x and y; z waits

seen from above



- Each corner keeps x and y; z is dropped.

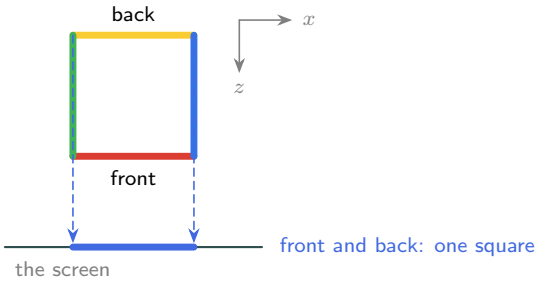
```
out.pos = vec4f(xyz.x / aspect, xyz.y, 0.5, 1.0);
```

aspect: lecture 26, slide 17. Clip space: lecture 25, slide 25.

▶ **Run it in the Cube Lab:**
The cube straight on

Straight on: vs_main hands on x and y; z waits

seen from above



- Each corner keeps x and y; z is dropped.
- Front and back land on one square.

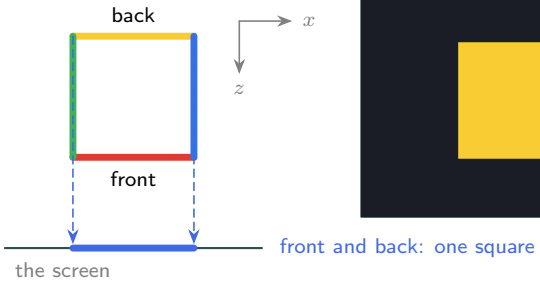
```
out.pos = vec4f(xyz.x / aspect, xyz.y, 0.5, 1.0);
```

aspect: lecture 26, slide 17. Clip space: lecture 25, slide 25.

▶ **Run it in the Cube Lab:**
The cube straight on

Straight on: vs_main hands on x and y; z waits

seen from above



the real render

- Each corner keeps x and y; z is dropped.
- Front and back land on one square.
- We see yellow: the back! Why? Round 2.

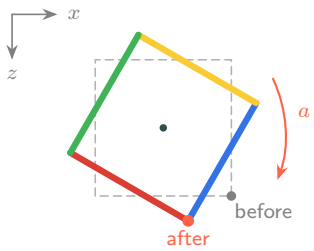
```
out.pos = vec4f(xyz.x / aspect, xyz.y, 0.5, 1.0);
```

aspect: lecture 26, slide 17. Clip space: lecture 25, slide 25.

▶ **Run it in the Cube Lab:**
The cube straight on

Turn about y: lecture 7's turn, with z in y's place

seen from above

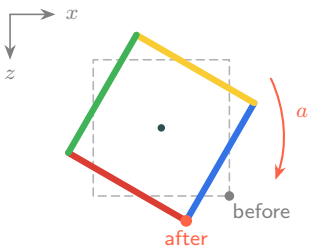


- From above, the cube is a square that turns.

▶ **Run it in the Cube Lab:**
Turn about y

Turn about y: lecture 7's turn, with z in y's place

seen from above



- From above, the cube is a square that turns.
- x and z change; y stays.

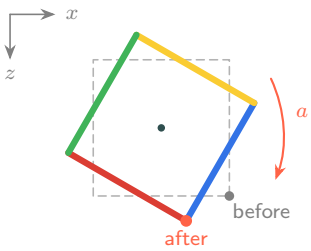
Lecture 7, slide 42: $x' = x\cos a - y\sin a, \quad y' = x\sin a + y\cos a$

z in y's place: $x' = x\cos a - z\sin a, \quad z' = x\sin a + z\cos a$

▶ **Run it in the Cube Lab:**
Turn about y

Turn about y: lecture 7's turn, with z in y's place

seen from above



- From above, the cube is a square that turns.
- x and z change; y stays.

Lecture 7, slide 42: $x' = x\cos a - y\sin a, \quad y' = x\sin a + y\cos a$

z in y's place: $x' = x\cos a - z\sin a, \quad z' = x\sin a + z\cos a$

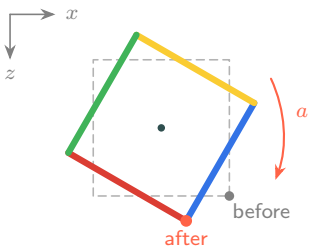
in our function turn_y:

```
let x = cos(angle) * p.x - sin(angle) * p.z;
let z = sin(angle) * p.x + cos(angle) * p.z;
```

▶ **Run it in the Cube Lab:**
Turn about y

Turn about y: lecture 7's turn, with z in y's place

seen from above





- From above, the cube is a square that turns.
- x and z change; y stays.
- The angle is u.time.

Lecture 7, slide 42: $x' = x\cos a - y\sin a, \quad y' = x\sin a + y\cos a$

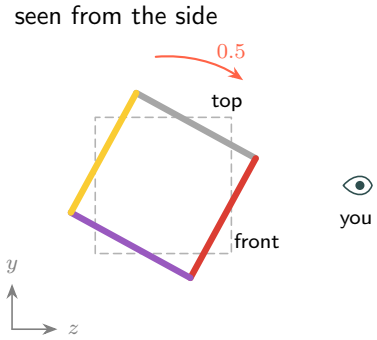
z in y's place: $x' = x\cos a - z\sin a, \quad z' = x\sin a + z\cos a$

in our function turn_y:

```
let x = cos(angle) * p.x - sin(angle) * p.z;
let z = sin(angle) * p.x + cos(angle) * p.z;
let p1 = turn_y(xyz, u.time); the angle grows by 1 every second
```

▶ **Run it in the Cube Lab:**
Turn about y

Tilt it toward you: a second turn, about x

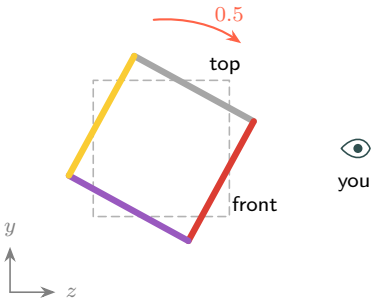


- Tilt 0.5 (about 29°): the top comes into view.

▶ **Run it in the Cube Lab:**
Turn, then tilt

Tilt it toward you: a second turn, about x

seen from the side



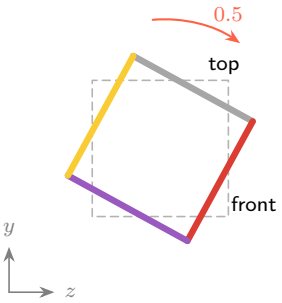
- Tilt 0.5 (about 29°): the top comes into view.
- Two turns, one after the other.

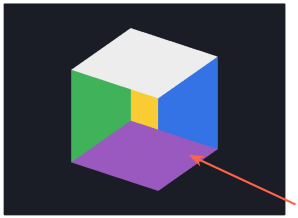
```
let p1 = turn_y(xyz, u.time);   1. turn
let p2 = turn_x(p1, 0.5);       2. tilt: lecture 7's turn in y and z

out.pos = vec4f(p2.x / aspect, p2.y, 0.5, 1.0);
```

▶ **Run it in the Cube Lab:**
Turn, then tilt

Tilt it toward you: a second turn, about x

seen from the side





the bottom shows through the top!

```
let p1 = turn_y(xyz, u.time);   1. turn
let p2 = turn_x(p1, 0.5);       2. tilt: lecture 7's turn in y and z

out.pos = vec4f(p2.x / aspect, p2.y, 0.5, 1.0);
```

- Tilt 0.5 (about 29°): the top comes into view.
- Two turns, one after the other.
- Wrong faces in front: round 2 fixes it.

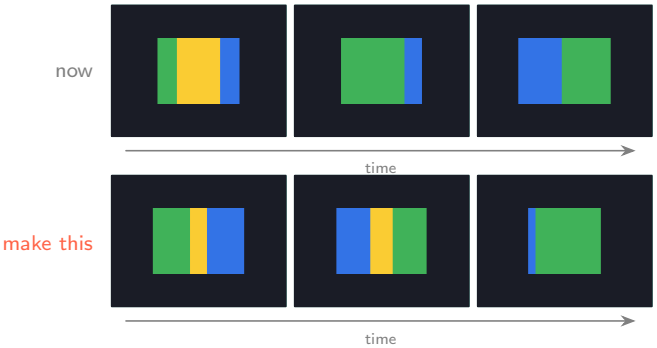
▶ **Run it in the Cube Lab:**
Turn, then tilt

You Try 1: the other way, twice as fast

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

```
// 1.  turn the cube about the y axis, a little more every second
let p1 = turn_y(xyz, u.time);
```

change this line



- **Turn the other way, twice as fast.**
- Guess first: which line, and how?

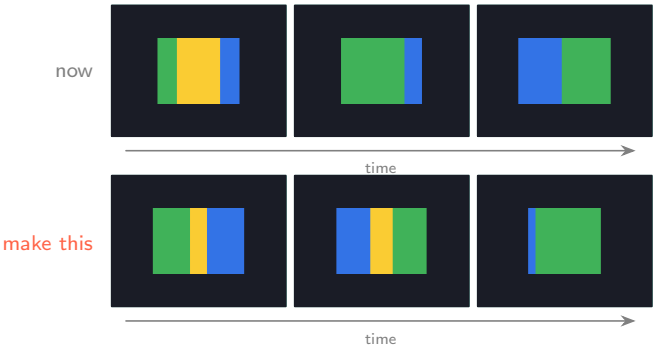
You Try 1: the other way, twice as fast

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

```
// 1.  turn the cube about the y axis, a little more every second
let p1 = turn_y(xyz, u.time);
```

change this line

answer: `let p1 = turn_y(xyz, -2.0 * u.time);`



- **Turn the other way, twice as fast.**
- Guess first: which line, and how?
- The angle: $-2.0\times$ u.time. The minus turns it back.

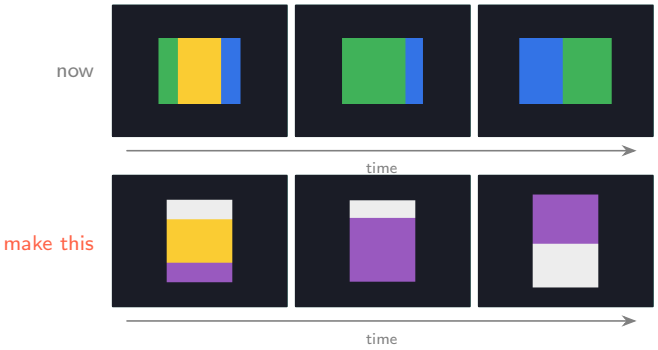
You Try 2: turn about the x axis

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

```
// 1.  turn the cube about the y axis, a little more every second
let p1 = turn_y(xyz, u.time);
```

change this line

The shader already has turn_x: it is just below turn_y.



- **Turn about x instead of y.**
- Guess first: will the bands stand up or lie down?

You Try 2: turn about the x axis

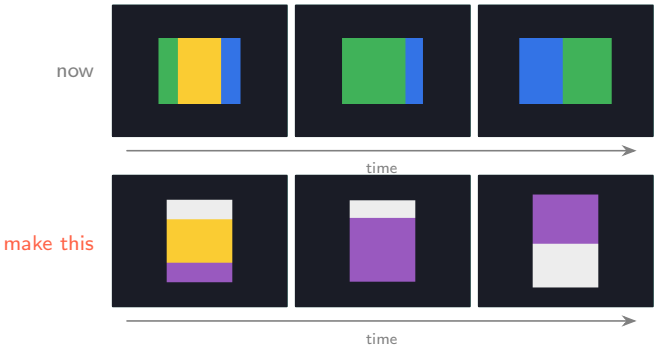
In the Cube Lab, choose **You Try 2**. Guess first, then change the code.

```
// 1.  turn the cube about the y axis, a little more every second
let p1 = turn_y(xyz, u.time);
```

change this line

answer: `let p1 = turn_x(xyz, u.time);`

The shader already has turn_x: it is just below turn_y.



- **Turn about x instead of y.**
- Guess first: will the bands stand up or lie down?
- They lie down: y and z change, x stays.

Round 2

What is in front?

the depth buffer

Without a depth test, the last triangle drawn wins

the order the GPU draws them (the buffer's order):



- Each triangle paints over what is there.

Without a depth test, the last triangle drawn wins

the order the GPU draws them (the buffer's order):





2 back is drawn after 1 front

- Each triangle paints over what is there.
- Straight on: the back covers the front.

Without a depth test, the last triangle drawn wins

the order the GPU draws them (the buffer's order):





2 back is drawn after 1 front



6 bottom is drawn after 5 top

- Each triangle paints over what is there.
- Straight on: the back covers the front.
- Tilted: the bottom covers the top.

Without a depth test, the last triangle drawn wins

the order the GPU draws them (the buffer's order):





2 back is drawn after 1 front

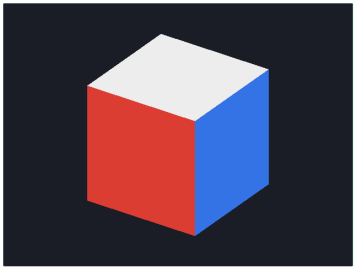


6 bottom is drawn after 5 top

- Each triangle paints over what is there.
- Straight on: the back covers the front.
- Tilted: the bottom covers the top.
- Lecture 24: keep the nearest instead.

The depth buffer: one number per pixel, the nearest wins

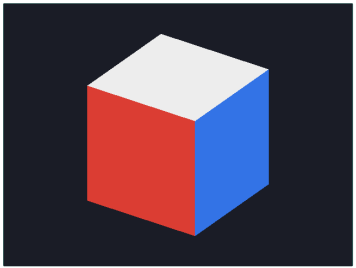
the colour page



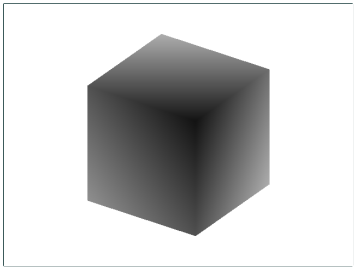
- Lecture 24, slide 46: a second page.

The depth buffer: one number per pixel, the nearest wins

the colour page



the depth page

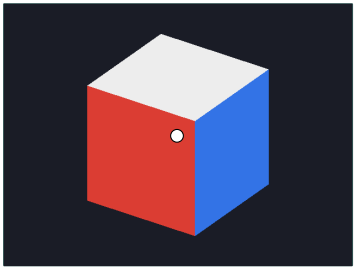


dark: near white: 1.0, nothing drawn

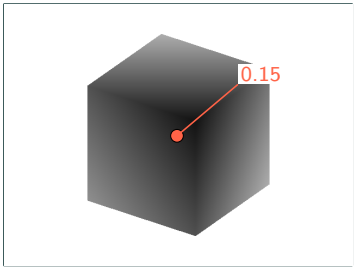
- Lecture 24, slide 46: a second page.
- Every frame starts at 1.0: as far as can be.

The depth buffer: one number per pixel, the nearest wins

the colour page



the depth page

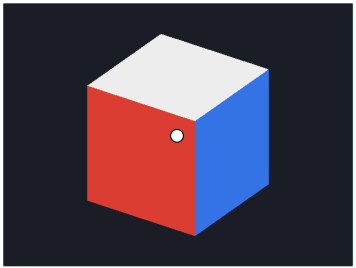


dark: near white: 1.0, nothing drawn

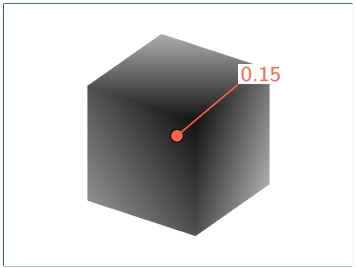
- Lecture 24, slide 46: a second page.
- Every frame starts at 1.0: as far as can be.

The depth buffer: one number per pixel, the nearest wins

the colour page



the depth page



dark: near white: 1.0, nothing drawn

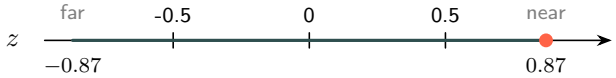
- Lecture 24, slide 46: a second page.
- Every frame starts at 1.0: as far as can be.
- Smaller depth wins.

A new pixel here with depth d:

$d < 0.15$: nearer. Keep its colour; store d.

$d \geq 0.15$: behind. Throw it away.

Depth runs from 0 (near) to 1 (far)

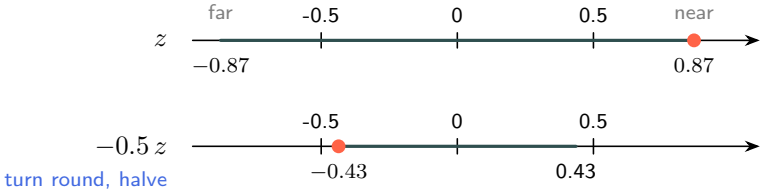


- The GPU wants depth 0 to 1.

A corner is $\sqrt{0.5^2 + 0.5^2 + 0.5^2} = 0.87$ from the middle. Turning keeps that.

▶ **Run it in the Cube Lab:**
The depth buffer on

Depth runs from 0 (near) to 1 (far)

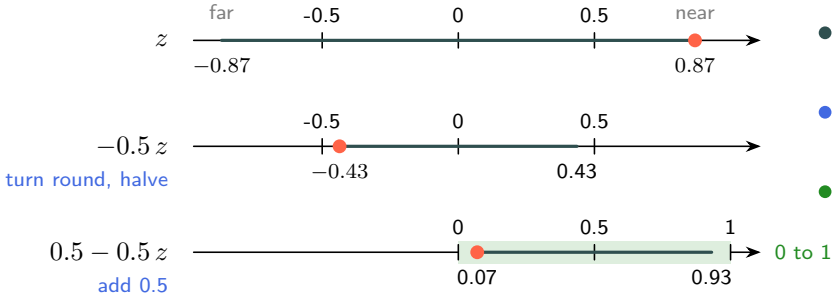


- The GPU wants depth 0 to 1.
- Nearer must be smaller.

A corner is $\sqrt{0.5^2 + 0.5^2 + 0.5^2} = 0.87$ from the middle. Turning keeps that.

▶ **Run it in the Cube Lab:**
The depth buffer on

Depth runs from 0 (near) to 1 (far)

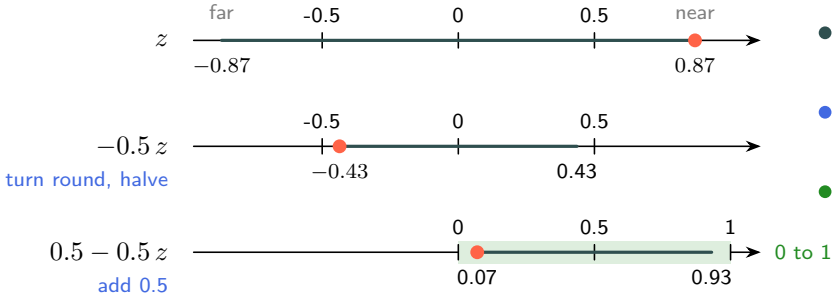


A corner is $\sqrt{0.5^2 + 0.5^2 + 0.5^2} = 0.87$ from the middle. Turning keeps that.

- The GPU wants depth 0 to 1.
- Nearer must be smaller.
- Outside 0 to 1: cut off.

▶ **Run it in the Cube Lab:**
The depth buffer on

Depth runs from 0 (near) to 1 (far)



A corner is $\sqrt{0.5^2 + 0.5^2 + 0.5^2} = 0.87$ from the middle. Turning keeps that.

- The GPU wants depth 0 to 1.
- Nearer must be smaller.
- Outside 0 to 1: cut off.

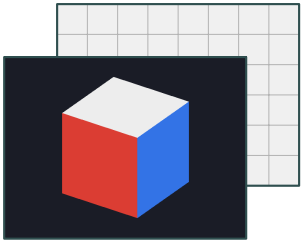
```
let depth = 0.5 - 0.5 * p2.z;
out.pos = vec4f(p2.x / aspect, p2.y, depth, 1.0);
```

the depth goes in out.pos's third slot

▶ **Run it in the Cube Lab:**
The depth buffer on

Our program: one texture, two settings

depth texture: 1.0 everywhere



colour page (lecture 25)

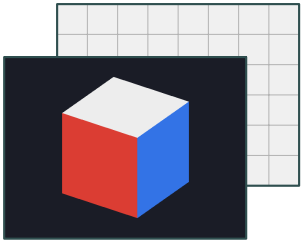
1. A depth texture
   `format=depth24plus`
   the window's size; made once

- Three additions to lecture 25's program.

cube.py: lecture 25's triangle_colours.py, plus these three, the cube, and u.
A shader whose first line says `// depth buffer: off` runs without them.

Our program: one texture, two settings

depth texture: 1.0 everywhere



colour page (lecture 25)

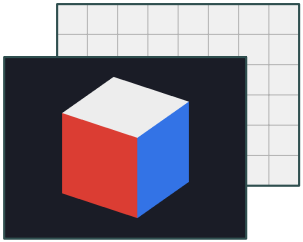
1. A depth texture
   `format=depth24plus`
   the window's size; made once
2. The pipeline: keep the smaller
   `"depth_compare": less`
   a setting in the pipeline

- Three additions to lecture 25's program.

cube.py: lecture 25's triangle_colours.py, plus these three, the cube, and u.
A shader whose first line says `// depth buffer: off` runs without them.

Our program: one texture, two settings

depth texture: 1.0 everywhere



colour page (lecture 25)

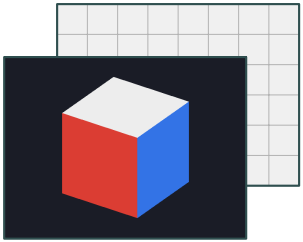
1. A depth texture
   `format=depth24plus`
   the window's size; made once
2. The pipeline: keep the smaller
   `"depth_compare": less`
   a setting in the pipeline
3. The pass: clear to 1.0
   `"depth_clear_value": 1.0`
   every frame, before drawing

- Three additions to lecture 25's program.

cube.py: lecture 25's triangle_colours.py, plus these three, the cube, and u.
A shader whose first line says `// depth buffer: off` runs without them.

Our program: one texture, two settings

depth texture: 1.0 everywhere



colour page (lecture 25)

1. A depth texture
   `format=depth24plus`
   the window's size; made once
2. The pipeline: keep the smaller
   `"depth_compare": less`
   a setting in the pipeline
3. The pass: clear to 1.0
   `"depth_clear_value": 1.0`
   every frame, before drawing

- Three additions to lecture 25's program.
- In the Cube Lab: the **Depth buffer** tick box.

cube.py: lecture 25's triangle_colours.py, plus these three, the cube, and u.
A shader whose first line says `// depth buffer: off` runs without them.

You Try 3: the cube is inside out

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

```
// the depth buffer wants 0 for the nearest and 1 for the farthest.
let depth = 0.5 + 0.5 * p2.z;
```

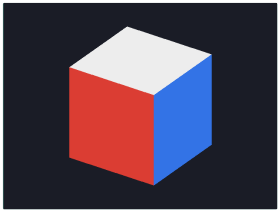
change this line

- **The far faces win. Fix the depth line.**
- Guess first: near faces or far ones?

now



make this



You Try 3: the cube is inside out

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

```
// the depth buffer wants 0 for the nearest and 1 for the farthest.
let depth = 0.5 + 0.5 * p2.z;
```

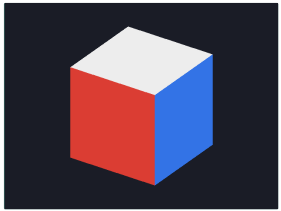
change this line

answer: `let depth = 0.5 - 0.5 * p2.z;`

now



make this



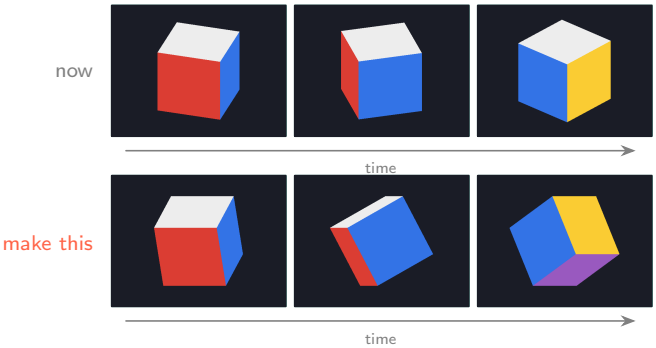
- **The far faces win. Fix the depth line.**
- Guess first: near faces or far ones?
- Far ones: with $+$, far gets the small depth.

You Try 4: tilt first, then turn

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

```
let p1 = turn_y(xyz, u.time);
let p2 = turn_x(p1, 0.5);
```

swap the turns



- **Tilt first, then turn.**
- Guess first: will it move the same way?

You Try 4: tilt first, then turn

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

```
let p1 = turn_y(xyz, u.time);
let p2 = turn_x(p1, 0.5);
```

swap the turns

answer:

```
let p1 = turn_x(xyz, 0.5);
let p2 = turn_y(p1, u.time);
```

now



time

make this



time

- **Tilt first, then turn.**
- Guess first: will it move the same way?
- No: the tilted cube swings round the upright y axis. Lecture 8, slide 10: order matters.

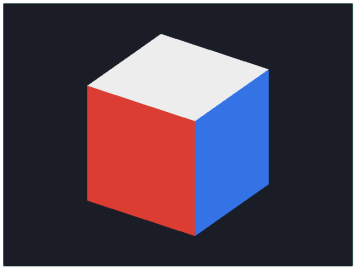
Round 3

Far things look smaller

a camera, and a divide

Our cube looks flat: the far side is as big as the near

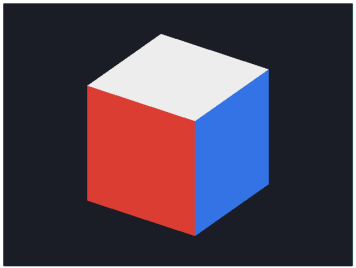
round 2: no camera



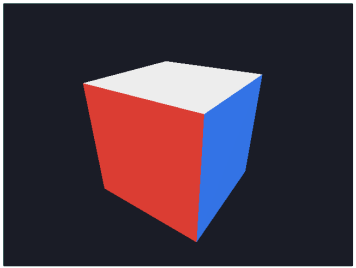
- Round 2: parallel edges stay parallel.

Our cube looks flat: the far side is as big as the near

round 2: no camera



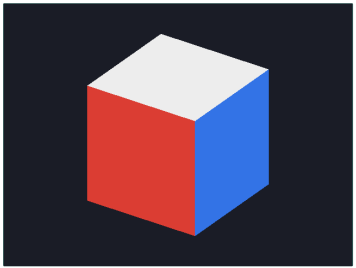
round 3: a camera at $z = 2$



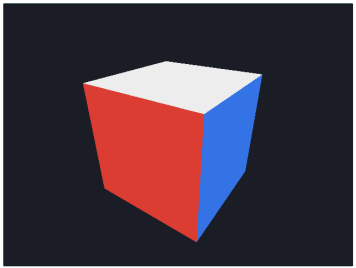
- Round 2: parallel edges stay parallel.
- An eye or a camera: farther looks smaller.

Our cube looks flat: the far side is as big as the near

round 2: no camera



round 3: a camera at $z = 2$



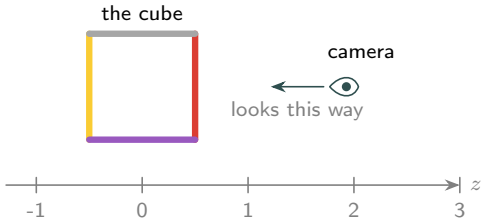
- Round 2: parallel edges stay parallel.
- An eye or a camera: farther looks smaller.
- Two more steps in vs_main.

Step 3: put a camera in front of the cube.

Step 4: divide by the distance: lecture 13's window, run backwards.

Step 3: the camera stands at $z = 2$, looking toward $-z$

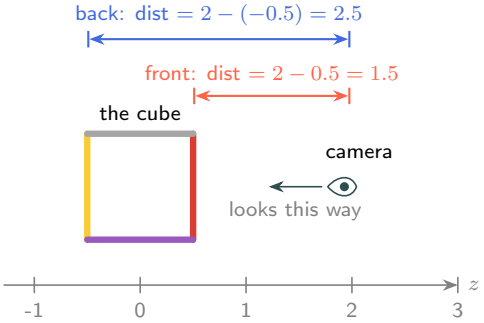
seen from the side (the cube not yet turned)



- The camera is a point on the z axis.

Step 3: the camera stands at $z = 2$, looking toward $-z$

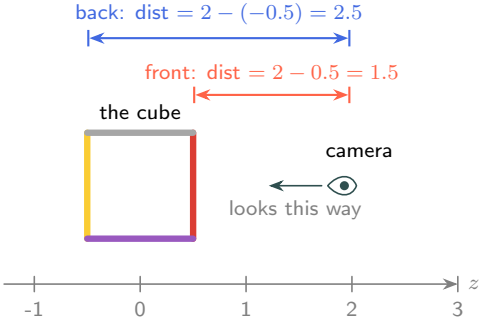
seen from the side (the cube not yet turned)



- The camera is a point on the z axis.
- dist: how far in front of it.

Step 3: the camera stands at $z = 2$, looking toward $-z$

seen from the side (the cube not yet turned)

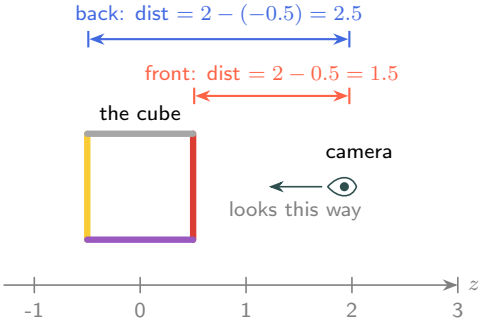



```
let dist = 2.0 - p2.z;
```
how far in front of the camera

- The camera is a point on the z axis.
- dist: how far in front of it.

Step 3: the camera stands at $z = 2$, looking toward $-z$

seen from the side (the cube not yet turned)



- The camera is a point on the z axis.
- dist: how far in front of it.
- Lecture 9: move to the camera's frame.

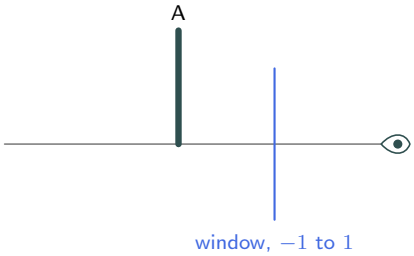
```
let dist = 2.0 - p2.z;
```
how far in front of the camera

Lecture 9, slide 19: world to frame, subtract the frame's origin first. The camera's frame sits at $z = 2$ and looks along $-z$, so in front of it, dist $= -(z - 2) = 2 - z$.

Step 4: similar triangles make far things small

The line from a point to the eye crosses the window at y_s.

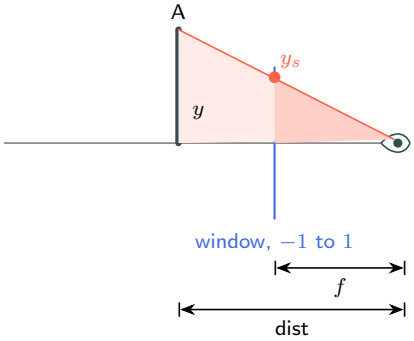
- Lecture 13's window, run backwards.



Step 4: similar triangles make far things small

The line from a point to the eye crosses the window at y_s.

$$\frac{y_s}{f} = \frac{y}{\text{dist}} \quad \Rightarrow y_s = \frac{f\,y}{\text{dist}}$$



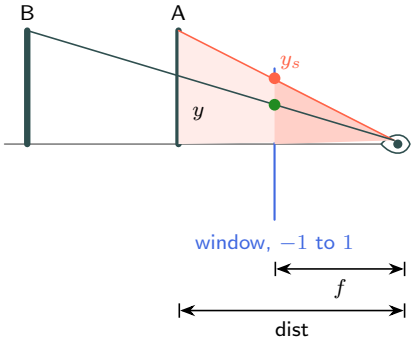
- Lecture 13's window, run backwards.
- Two triangles, one shape.

Step 4: similar triangles make far things small

The line from a point to the eye crosses the window at y_s.

$$\frac{y_s}{f} = \frac{y}{\text{dist}} \quad \Rightarrow y_s = \frac{f\,y}{\text{dist}}$$

Try first: $f = 1.73$, $y = 1.5$
A, dist 3: y_s =?
B, dist 5: y_s =?



- Lecture 13's window, run backwards.
- Two triangles, one shape.

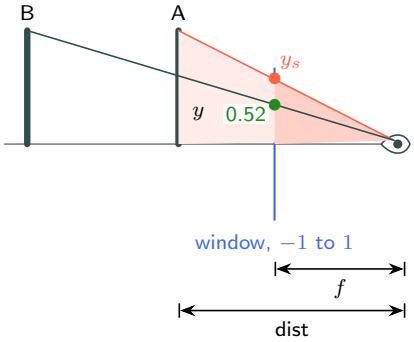
Step 4: similar triangles make far things small

The line from a point to the eye crosses the window at y_s.

$$\frac{y_s}{f} = \frac{y}{\text{dist}} \quad \Rightarrow y_s = \frac{f\,y}{\text{dist}}$$

Try first: $f = 1.73$, $y = 1.5$
A, dist 3: $y_s = ?$
B, dist 5: $y_s = ?$

A: $1.73 \times 1.5/3 = 0.87$
B: $1.73 \times 1.5/5 = 0.52$



- Lecture 13's window, run backwards.
- Two triangles, one shape.
- Same post, farther: smaller on screen.

Step 4 in code: put dist in w, and the GPU divides

out.pos $= (x,\ y,\ z,\ w)$, as vs_main writes it



- Until now w was 1.

▶ **Run it in the Cube Lab:**
A camera: far looks smaller

Step 4 in code: put dist in w, and the GPU divides

out.pos $= (x,\ y,\ z,\ w)$, as vs_main writes it

$f\,x$ / aspect	$f\,y$	depth $\times$ dist	dist

the GPU divides all four by w (lecture 25, slide 24)

$\dfrac{f\,x}{\text{aspect} \cdot \text{dist}}$	$\dfrac{f\,y}{\text{dist}}$	depth	1

- Until now w was 1.
- Now $w =$ dist: one divide does it all.

▶ **Run it in the Cube Lab:**
A camera: far looks smaller

Step 4 in code: put dist in w, and the GPU divides

out.pos $= (x,\ y,\ z,\ w)$, as vs_main writes it

$f\,x\ /$ aspect	$f\,y$	depth $\times$ dist	dist

the GPU divides all four by w (lecture 25, slide 24)

$\dfrac{f\,x}{\text{aspect} \cdot \text{dist}}$	$\dfrac{f\,y}{\text{dist}}$	depth	1

slide 21's $f\,y$ / dist: done for us

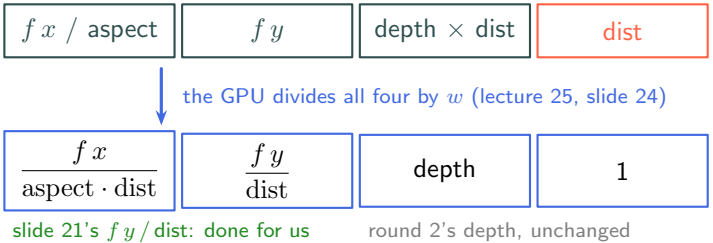
round 2's depth, unchanged

- Until now w was 1.
- Now $w =$ dist: one divide does it all.

▶ **Run it in the Cube Lab:**
A camera: far looks smaller

Step 4 in code: put dist in w, and the GPU divides

out.pos $= (x,\ y,\ z,\ w)$, as vs_main writes it




```
let f = 1.73;                          // 60 degrees:  slide 23
out.pos = vec4f(f * p2.x / aspect, f * p2.y, depth * dist, dist);
```

- Until now w was 1.
- Now w = dist: one divide does it all.
- Lecture 8: a 4th number that is not 1.

▶ **Run it in the Cube Lab:**
A camera: far looks smaller

Step 4 in code: put dist in w, and the GPU divides

out.pos $= (x,\ y,\ z,\ w)$, as vs_main writes it

$f\,x$ / aspect	$f\,y$	depth $\times$ dist	dist

the GPU divides all four by w (lecture 25, slide 24)

$\dfrac{f\,x}{\text{aspect} \cdot \text{dist}}$	$\dfrac{f\,y}{\text{dist}}$	depth	1

slide 21's $f\,y$ / dist: done for us round 2's depth, unchanged

```
let f = 1.73;                              // 60 degrees:  slide 23
out.pos = vec4f(f * p2.x / aspect, f * p2.y, depth * dist, dist);
```

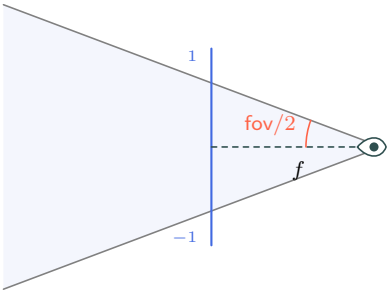


Why not divide ourselves? Because the GPU cuts away what is behind the eye *before* it divides (lecture 24, slide 13). Then it divides.

- Until now w was 1.
- Now $w =$ dist: one divide does it all.
- Lecture 8: a 4th number that is not 1.

▶ **Run it in the Cube Lab:**
A camera: far looks smaller

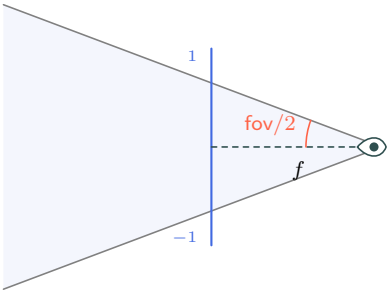
f sets the field of view



$$\tan(\text{fov}/2) = \frac{1}{f}$$

- The window is 2 tall, f from the eye.

f sets the field of view

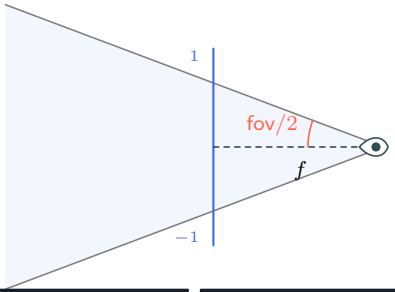


$$\tan(\text{fov}/2) = \frac{1}{f}$$

60°: $f = \dfrac{1}{\tan 30^\circ} = 1.73$

- The window is 2 tall, f from the eye.
- A 60° view: $f = 1.73$.

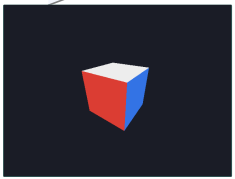
f sets the field of view



$$\tan(\text{fov}/2) = \frac{1}{f}$$

60°: $f = \dfrac{1}{\tan 30^\circ} = 1.73$

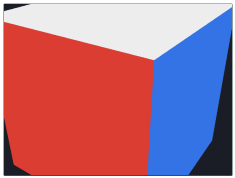
- The window is 2 tall, f from the eye.
- A 60° view: $f = 1.73$.
- Bigger f: narrower view, bigger cube. A zoom.



90°: $f = 1$



60°: $f = 1.73$

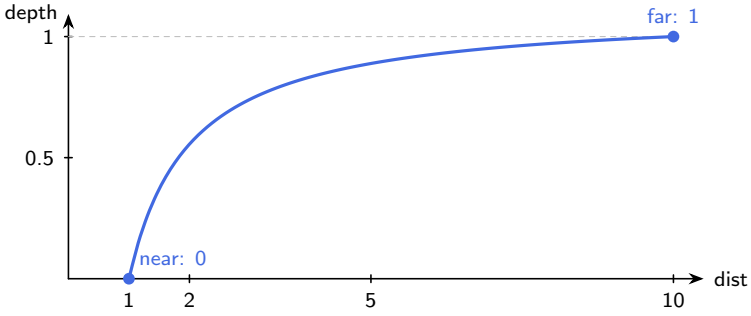


30°: $f = 3.73$

near and far: depth from distance

```
let near = 1.0;   let far = 10.0;
let a = far / (far - near);
let b = -near * far / (far - near);
out.pos = vec4f(..., a * dist + b, dist);
```

- Two new numbers: near and far.



▶ **Run it in the Cube Lab:**
Depth from near to far

near and far: depth from distance

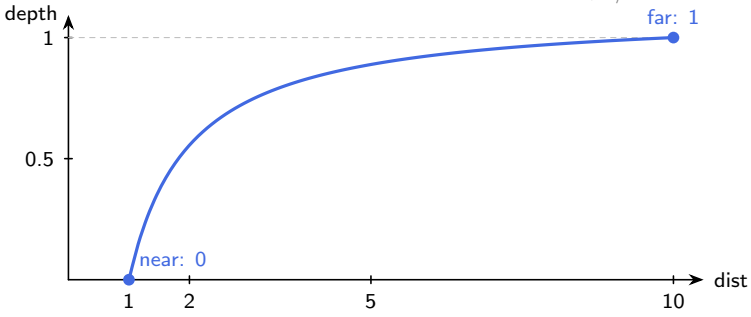
```
let near = 1.0;   let far = 10.0;
let a = far / (far - near);
let b = -near * far / (far - near);
out.pos = vec4f(..., a * dist + b, dist);
```

after the divide:

$$\text{depth} = a + \frac{b}{\text{dist}}$$

near: $a + b/\text{near} = 0$

far: $a + b/\text{far} = 1$



- Two new numbers: near and far.
- a, b make near 0 and far 1.

▶ **Run it in the Cube Lab:**
Depth from near to far

near and far: depth from distance

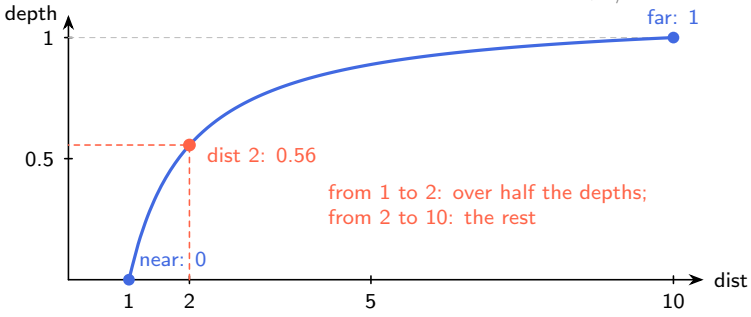
```
let near = 1.0;   let far = 10.0;
let a = far / (far - near);
let b = -near * far / (far - near);
out.pos = vec4f(..., a * dist + b, dist);
```

after the divide:

$$\text{depth} = a + \frac{b}{\text{dist}}$$

near: $a + b/\text{near} = 0$

far: $a + b/\text{far} = 1$



- Two new numbers: near and far.
- a, b make near 0 and far 1.
- Depths crowd far away: z-fighting (lecture 24, slide 54).

▶ **Run it in the Cube Lab:**
Depth from near to far

Four steps, three matrices: M, V, P



- Each step is one 4×4 matrix.

Four steps, three matrices: M, V, P



$\text{turn}_x(0.5)$
$\text{turn}_y(a)$

lecture 7's turns, one size up

- Each step is one 4×4 matrix.

Four steps, three matrices: M, V, P



$\text{turn}_x(0.5)$
$\text{turn}_y(a)$

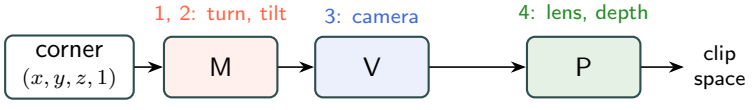
lecture 7's turns, one size up

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

lecture 8, slide 20: move by -2 in z

- Each step is one 4×4 matrix.

Four steps, three matrices: M, V, P



$\text{turn}_x(0.5)$
$\text{turn}_y(a)$

lecture 7's turns, one size up

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

lecture 8, slide 20: move by -2 in z

$$\begin{bmatrix} f/\text{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & -a & b \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

last row: $w = -z = \text{dist}$
row 3: $z_{\text{clip}} = a\ \text{dist} + b$

- Each step is one 4×4 matrix.
- P puts dist in w.

Four steps, three matrices: M, V, P



$\text{turn}_x(0.5)$
$\text{turn}_y(a)$

lecture 7's turns, one size up

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

lecture 8, slide 20: move by -2 in z

$$\begin{bmatrix} f/\text{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & -a & b \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

last row: $w = -z = \text{dist}$
row 3: $z_{\text{clip}} = a\ \text{dist} + b$

- Each step is one 4×4 matrix.
- P puts dist in w.
- Multiply them: one matrix, P V M.

$$\mathbf{p}_{\text{clip}} = \text{P V M } \mathbf{p}$$

read right to left: M acts first

Lecture 8, slide 18: a 4th number, always 1, lets a matrix move a point.
Lecture 9's names: M $= \mathbf{M}_{WO}$ (object to world), V $= \mathbf{M}_{CW}$ (world to camera).

One matrix a frame, one multiply a corner

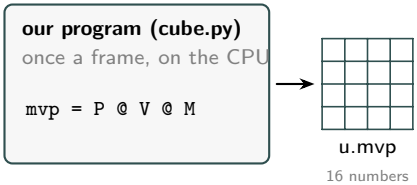
our program (cube.py)
once a frame, on the CPU

```
mvp = P @ V @ M
```

- Our program multiplies P V M once.

▶ **Run it in the Cube Lab:**
One matrix, u.mvp: use the sliders

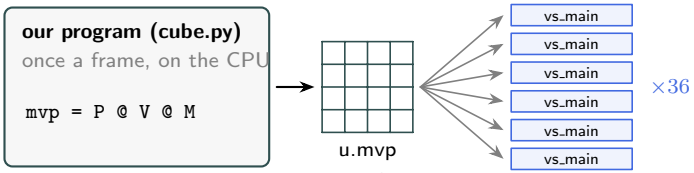
One matrix a frame, one multiply a corner



- Our program multiplies P V M once.
- It goes to the GPU in u.

▶ **Run it in the Cube Lab:**
One matrix, u.mvp: use the sliders

One matrix a frame, one multiply a corner



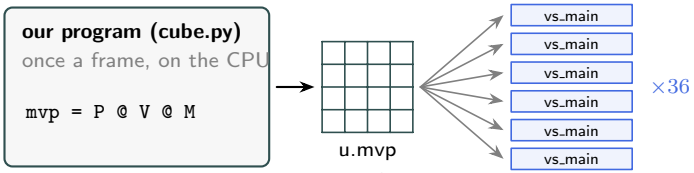

```
out.pos = u.mvp * vec4f(xyz.x, xyz.y, xyz.z, 1.0);
```

u gets one more field, mvp: mat4x4f, a 4×4 matrix. Our program fills it.

- Our program multiplies P V M once.
- It goes to the GPU in u.
- vs_main: one multiply per corner.

▶ Run it in the Cube Lab:
One matrix, u.mvp: use the sliders

One matrix a frame, one multiply a corner




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

u gets one more field, mvp: mat4x4f, a 4×4 matrix. Our program fills it.



The same picture as near_far.wgsl's four steps. In the Cube Lab, six sliders change M, V and P while it runs.

- Our program multiplies P V M once.
- It goes to the GPU in u.
- vs_main: one multiply per corner.

▶ **Run it in the Cube Lab:**
One matrix, u.mvp: use the sliders

You Try 5: back away and zoom in

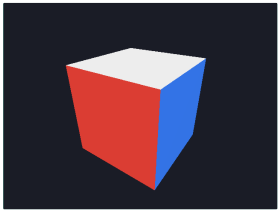
In the Cube Lab, choose **You Try 5**. Guess first, then change the code.

```
let dist = 2.0 - p2.z;
let f = 1.73;
```

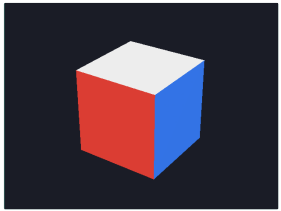
change two numbers

- **Camera to $z = 6$; then $f = 5.2$, the old size.**
- Guess first: same size, so what changes?

now



make this



You Try 5: back away and zoom in

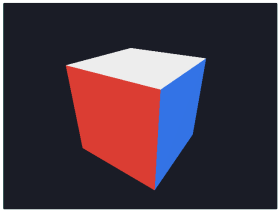
In the Cube Lab, choose **You Try 5**. Guess first, then change the code.

```
let dist = 2.0 - p2.z;
let f = 1.73;
```

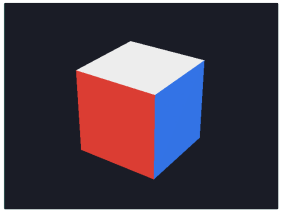
change two numbers

answer: `let dist = 6.0 - p2.z;` `let f = 5.2;`

now



make this



- **Camera to $z = 6$; then $f = 5.2$, the old size.**
- Guess first: same size, so what changes?
- It looks flat, as in round 2. Distance makes perspective; f only zooms.

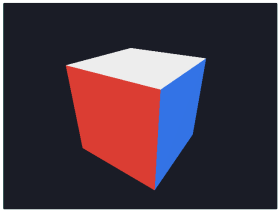
You Try 6: the near cut

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

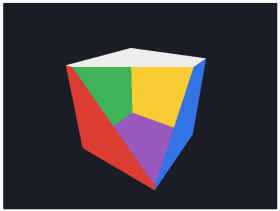
```
let near = 1.0;                    // nearer than this:  cut off
let far = 10.0;    change near
```

- **Make near 1.6.**
- Guess first: which part disappears?

now



make this



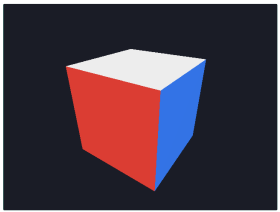
You Try 6: the near cut

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

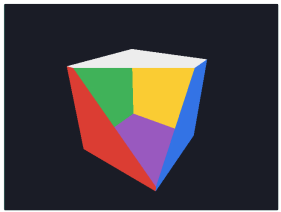
```
let near = 1.0;                    // nearer than this:  cut off
let far = 10.0;    change near
```

answer: `let near = 1.6;`

now



make this



- **Make near 1.6.**
- Guess first: which part disappears?
- The nearest corner: closer than 1.6, its depth is below 0. Cut off (lecture 24, slide 13).

Now make your own



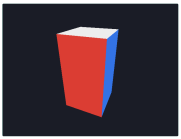
The mouse turns it

`turn_y(xyz, 3.0 * mx)`



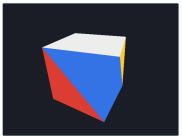
A hopping cube

`+ vec3f(0.0, hop, 0.0)`



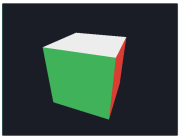
Squash and stretch

`xyz.y * (1.0 + s)`



A twisted cube

`the angle grows with xyz.y`



The dolly zoom

`back away, f grows`



The colour cube

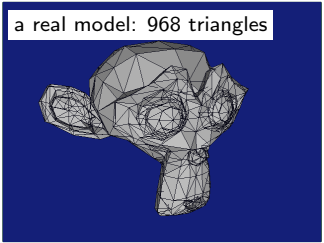
`out.colour = xyz + 0.5`

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

Summary

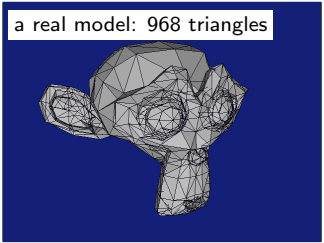
1	vs_main moves corners	turn_y, turn_x: lecture 7's turn 36 corners, once each; the order of the turns matters
2	The depth buffer	depth 0 (near) to 1 (far); smaller wins one texture, cleared to 1.0; outside 0 to 1 is cut off
3	Perspective	$y_s = f\,y\,/\,\text{dist}$, by putting dist in w the GPU divides by w; $f = 1/\tan(\text{fov}/2)$
4	near and far	depth $= a + b\,/\,\text{dist}$ near gives 0, far gives 1; depths crowd far away
5	MVP	u.mvp = P V M, made once a frame vs_main: `u.mvp * vec4f(x, y, z, 1.0)`

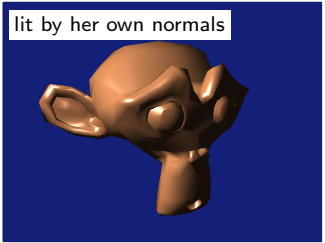
Next lecture: Suzanne, lit



- Suzanne: a model loaded from a file.

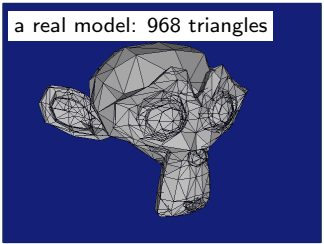
Next lecture: Suzanne, lit

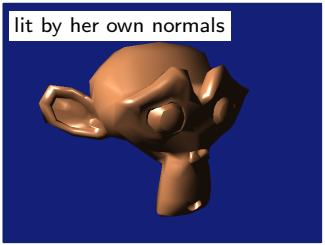




- Suzanne: a model loaded from a file.
- Each corner brings a normal; lecture 27's light uses it.

Next lecture: Suzanne, lit

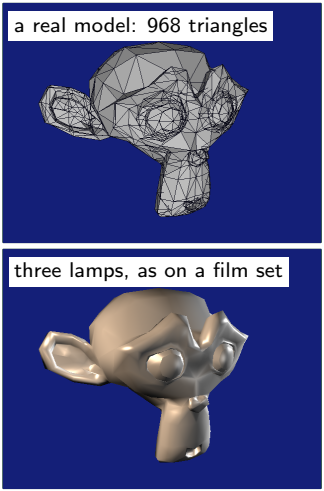






- Suzanne: a model loaded from a file.
- Each corner brings a normal; lecture 27's light uses it.
- Coloured lamps and a spotlight.

Next lecture: Suzanne, lit



- Suzanne: a model loaded from a file.
- Each corner brings a normal; lecture 27's light uses it.
- Coloured lamps and a spotlight.

Exit Ticket

Today's question *(2 minutes)*

(a) A corner is $y = 0.6$ above the middle and dist $= 3$ in front of the camera, with $f = 2$. How high is it on the screen: $y_s = f\, y \,/\, \mathrm{dist}$?

(b) With near $= 1$ and far $= 10$, the depth is $1.11 - 1.11 \,/\, \mathrm{dist}$. Find the depth at dist $= 2$. Is it nearer to 0 or to 1?

CS 453 — Exit Ticket #28

Name: _____

Roll #: _____ Date: _____

Answer:

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