Lecture 36: Splatting 2

A Blob in 3D

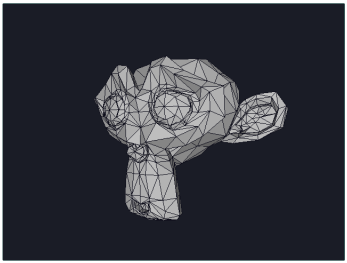
CS 453 — Computer Graphics

Fakhir Shaheen

Forman Christian University

Lecture 29's triangles. Today: a blob on each point

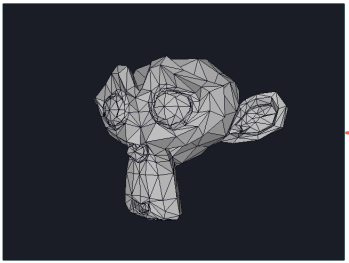
lecture 29: 968 triangles



hard edges

Lecture 29's triangles. Today: a blob on each point

lecture 29: 968 triangles



hard edges

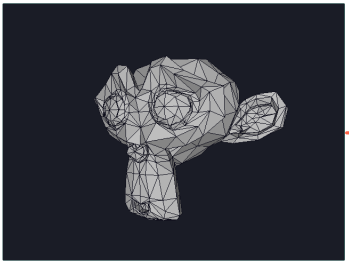
the same 507 points



lecture 29's v lines

Lecture 29's triangles. Today: a blob on each point

lecture 29: 968 triangles



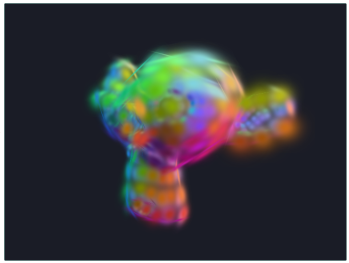
hard edges

the same 507 points



lecture 29's v lines

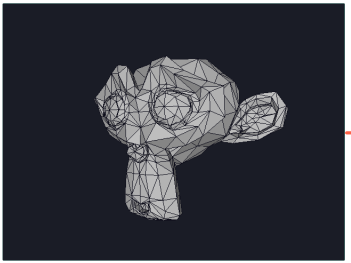
a blob on each point



no triangles

Lecture 29's triangles. Today: a blob on each point

lecture 29: 968 triangles



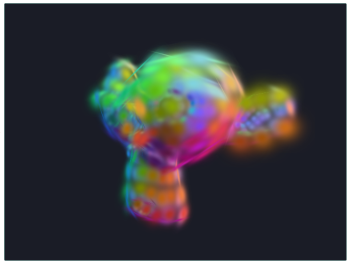
hard edges

the same 507 points



lecture 29's v lines

a blob on each point

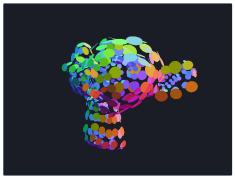


no triangles

Lecture 35's blob, now in 3D

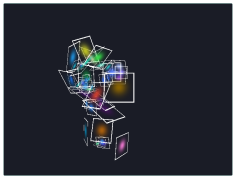
Today: three rounds of slides, then you try

1 **A blob in 3D**



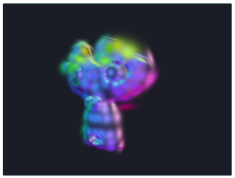
then You Try 1, 2

2 **Onto the screen**



then You Try 3, 4

3 **Drawing 507 blobs**



then You Try 5, 6

Open the Splat Lab now: the course calendar, lecture 36's row.

Pairs: one screen for two.

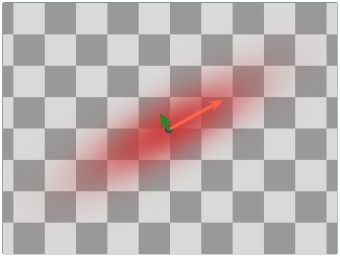
Round 1

A blob in 3D

three sizes, and the turn of three arrows

Lecture 35's blob, with one more size

lecture 35 (slide 16): flat

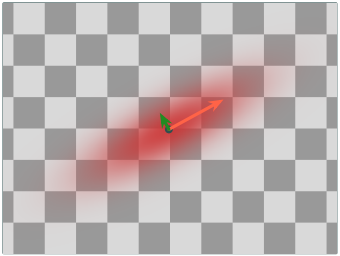


2 sizes, an angle

- 2D: two arrows.

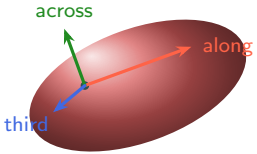
Lecture 35's blob, with one more size

lecture 35 (slide 16): flat



2 sizes, an angle

today: in 3D

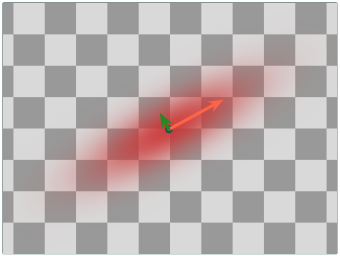


3 sizes, a turn

- 2D: two arrows.
- 3D: three arrows.

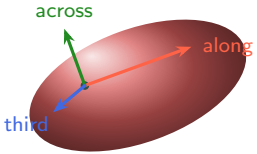
Lecture 35's blob, with one more size

lecture 35 (slide 16): flat



2 sizes, an angle

today: in 3D

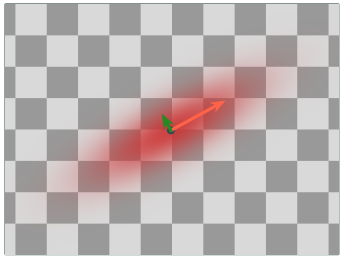


3 sizes, a turn

- 2D: two arrows.
- 3D: three arrows.
- Length = size.

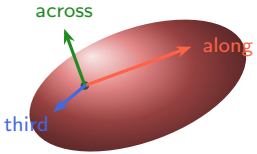
Lecture 35's blob, with one more size

lecture 35 (slide 16): flat



2 sizes, an angle

today: in 3D

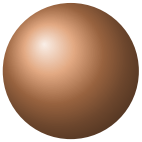


3 sizes, a turn

- 2D: two arrows.
- 3D: three arrows.
- Length = size.

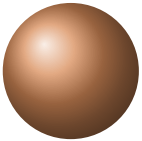
$\text{weight} = \exp\left(-\frac{1}{2}\, q \cdot q\right)$ q now has 3 parts

Flatten one size: a disc. A surface is made of flat blobs

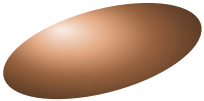


a ball

Flatten one size: a disc. A surface is made of flat blobs

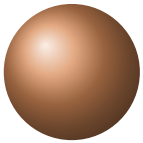


a ball

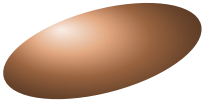


an egg

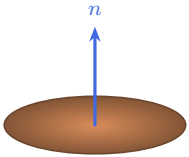
Flatten one size: a disc. A surface is made of flat blobs



a ball



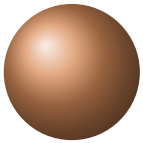
an egg



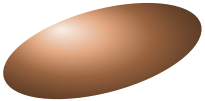
one size 0: a disc

- Flat: a disc facing n.

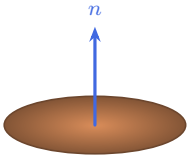
Flatten one size: a disc. A surface is made of flat blobs



a ball

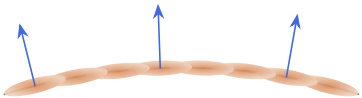


an egg



one size 0: a disc

- Flat: a disc facing n.
- Discs make a skin.



a surface, from the side

The paper's words: a Gaussian, and its covariance



R: the turn

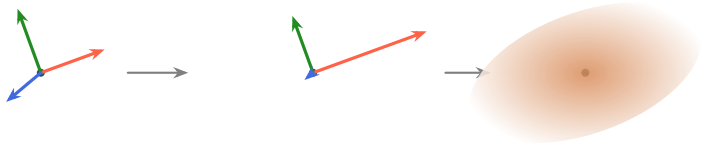
The paper's words: a Gaussian, and its covariance



R: the turn

RS: turn and sizes

The paper's words: a Gaussian, and its covariance



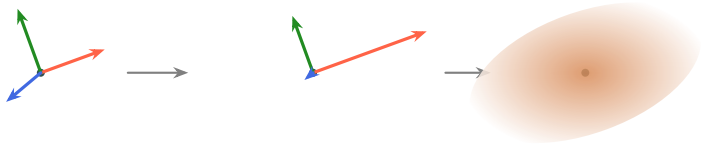
R: the turn

RS: turn and sizes

the blob

- RS's columns: our arrows.

The paper's words: a Gaussian, and its covariance



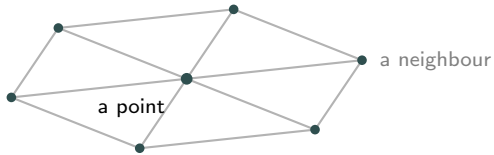
- RS's columns: our arrows.

R: the turn RS: turn and sizes the blob

$$\Sigma = (RS)(RS)^{\mathsf{T}}$$ the shape matrix

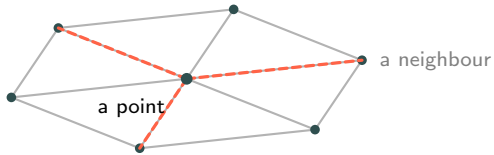
the paper: a blob is a **Gaussian**; Σ is its **covariance**

Suzanne's blobs: one disc on each point



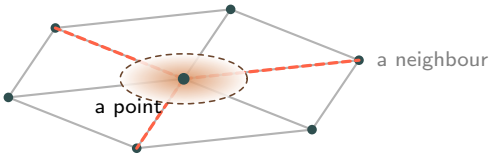
- A point, its neighbours.

Suzanne's blobs: one disc on each point



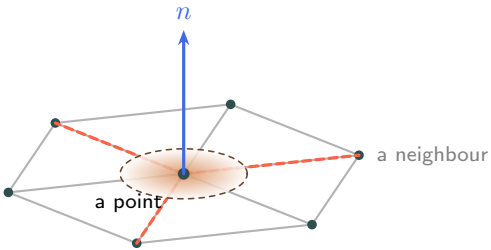
- A point, its neighbours.

Suzanne's blobs: one disc on each point



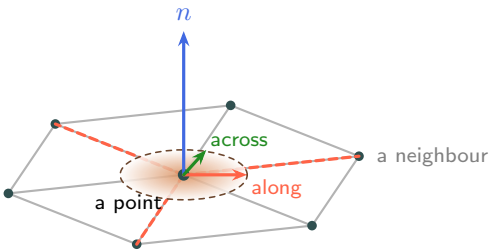
- A point, its neighbours.
- size $= 0.35\times$ average distance

Suzanne's blobs: one disc on each point



- A point, its neighbours.
- size $= 0.35\times$ average distance
- faces along n

Suzanne's blobs: one disc on each point



- A point, its neighbours.
- size $= 0.35\times$ average distance
- faces along n

make_blobs in splat.py, and what it prints

```
for k, centre in enumerate(points):
    near = [length(minus(points[j], centre)) for j in neighbours[k]]
    size = SPREAD * sum(near) / len(near)
```

SPREAD = 0.35

make_blobs in splat.py, and what it prints

```
for k, centre in enumerate(points):
    near = [length(minus(points[j], centre)) for j in neighbours[k]]
    size = SPREAD * sum(near) / len(near)          SPREAD = 0.35
    n = normals[k]
    along = cross(n, up)          up: (0, 1, 0)
    along = scale(along, size / length(along))
    across = cross(n, along)
```

make_blobs in splat.py, and what it prints

```
for k, centre in enumerate(points):
    near = [length(minus(points[j], centre)) for j in neighbours[k]]
    size = SPREAD * sum(near) / len(near)
    n = normals[k]
    along = cross(n, up)
    along = scale(along, size / length(along))
    across = cross(n, along)
    blobs.append(centre + along + across + list(CLAY) + [OPACITY])
```

SPREAD = 0.35

up: (0, 1, 0)

make_blobs in splat.py, and what it prints

```
for k, centre in enumerate(points):
    near = [length(minus(points[j], centre)) for j in neighbours[k]]
    size = SPREAD * sum(near) / len(near)
    n = normals[k]
    along = cross(n, up)
    along = scale(along, size / length(along))
    across = cross(n, along)
    blobs.append(centre + along + across + list(CLAY) + [OPACITY])
```

SPREAD = 0.35

up: (0, 1, 0)

python splat.py splat.wgsl prints:

```
507 blobs x 52 bytes = 26364 bytes in the vertex buffer
blob 0:  0.438 0.164 0.766  -0.027 0.000 0.020  -0.010 -0.029 -0.013  0.850 0.550 0.350  1.000
```

centre | along | across | colour: clay | opacity

make_blobs in splat.py, and what it prints

```
for k, centre in enumerate(points):
    near = [length(minus(points[j], centre)) for j in neighbours[k]]
    size = SPREAD * sum(near) / len(near)          SPREAD = 0.35
    n = normals[k]
    along = cross(n, up)          up: (0, 1, 0)
    along = scale(along, size / length(along))
    across = cross(n, along)
    blobs.append(centre + along + across + list(CLAY) + [OPACITY])
```

python splat.py splat.wgsl prints:

```
507 blobs x 52 bytes = 26364 bytes in the vertex buffer
blob 0:  0.438 0.164 0.766  -0.027 0.000 0.020  -0.010 -0.029 -0.013  0.850 0.550 0.350  1.000
         centre             along              across                 colour: clay       opacity
```

Lecture 29: 6 numbers a corner. Today: 13 a blob.

You Try 1: half the size

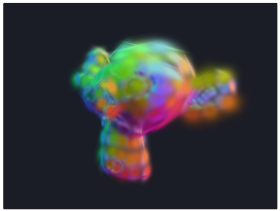
In the Splat Lab, choose **You Try 1**. Guess first, then change the code.

```
let grow = 1.0;
let xyz = centre + grow * (q.x * along + q.y * across);
```

change this line

- **Half the size.**
- Guess: what shows between them?

now



after your change



You Try 1: half the size

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

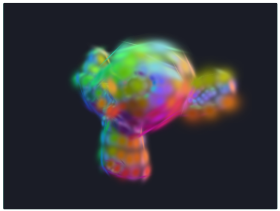
```
let grow = 1.0;
let xyz = centre + grow * (q.x * along + q.y * across);
```

change this line

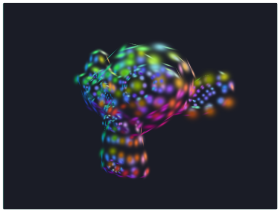
answer: `let grow = 0.5;`

- **Half the size.**
- Guess: what shows between them?
- Gaps: her 507 points.

now



after your change



You Try 2: stand every blob on its edge

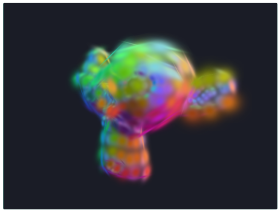
In the Splat Lab, choose **You Try 2**. Guess first, then change the code.

```
let n = normalize(cross(along, across));
```
her normal

```
let xyz = centre + grow * (q.x * along + q.y * across);
```

change this line: n for across

now



after your change



- **Stand each disc up.**
- Guess: still smooth?

You Try 2: stand every blob on its edge

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

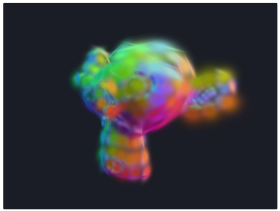
```
let n = normalize(cross(along, across));
```
her normal

```
let xyz = centre + grow * (q.x * along + q.y * across);
```

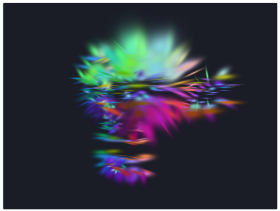
change this line: n for across

answer: `let xyz = centre + grow * (q.x * along + q.y * length(across) * n);`

now



after your change



- **Stand each disc up.**
- Guess: still smooth?
- No: spikes. The turn matters.

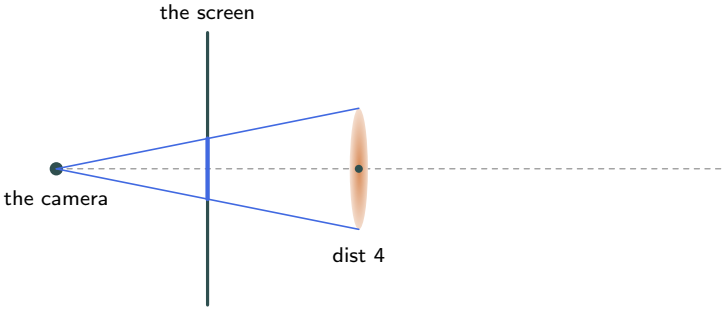
Round 2

Onto the screen

a disc seen by the camera is lecture 35's ellipse

The centre goes through lecture 28's MVP: far blobs shrink

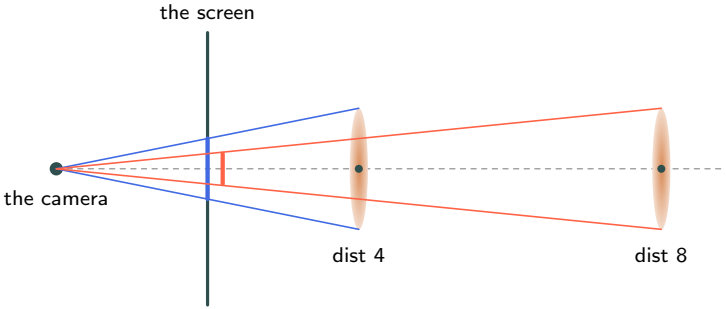
```
u.mvp * centre
```



- Lecture 29's line.

The centre goes through lecture 28's MVP: far blobs shrink

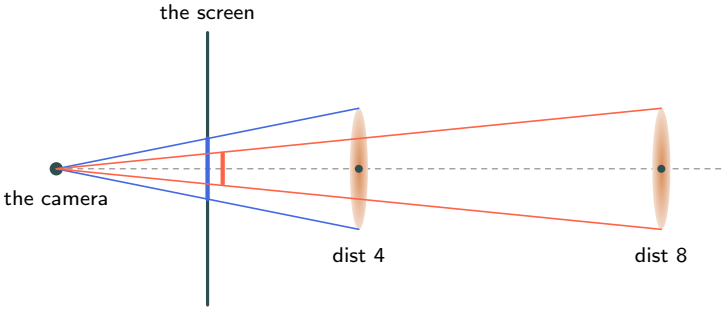
`u.mvp * centre`



- Lecture 29's line.
- Twice as far: half as big.

The centre goes through lecture 28's MVP: far blobs shrink

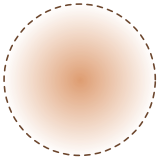
`u.mvp * centre`



- Lecture 29's line.
- Twice as far: half as big.

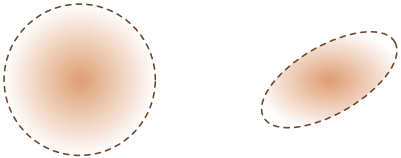
on the screen: $f \times \text{size} / \text{dist}$ lecture 28, slide 21

A disc at a slant is an ellipse: lecture 35's blob



facing you

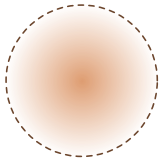
A disc at a slant is an ellipse: lecture 35's blob



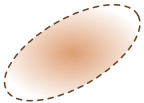
facing you tilted

- Tilted: lecture 35's ellipse.

A disc at a slant is an ellipse: lecture 35's blob



facing you



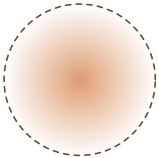
tilted



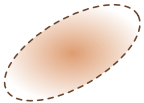
edge on

- Tilted: lecture 35's ellipse.

A disc at a slant is an ellipse: lecture 35's blob



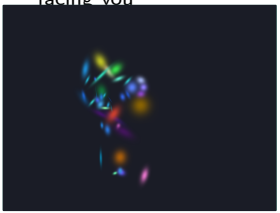
facing you



tilted



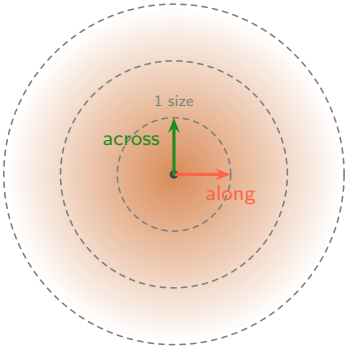
edge on



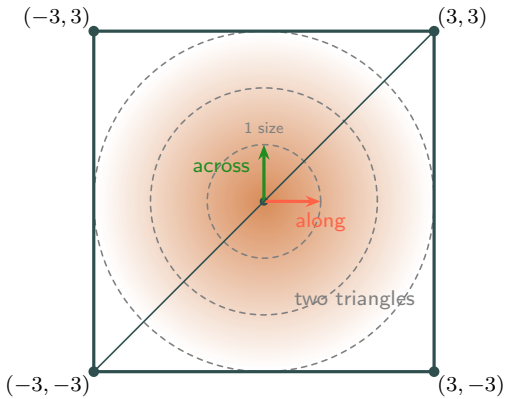
one blob in twelve

- Tilted: lecture 35's ellipse.
- The camera squashes it for us.

A square on the disc, 3 sizes out each way

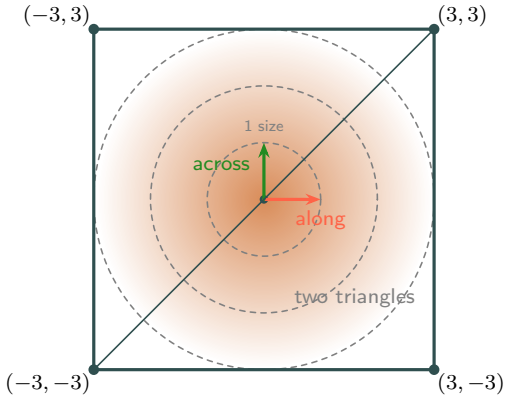


A square on the disc, 3 sizes out each way



- Past 3 sizes: weight < 0.01.

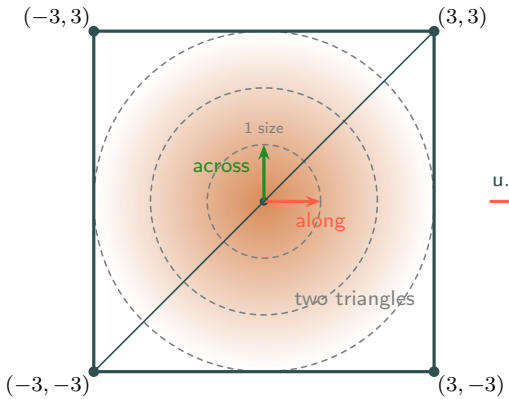
A square on the disc, 3 sizes out each way



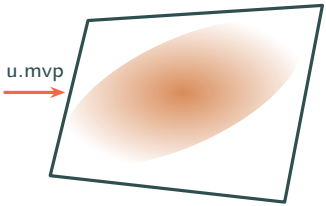
corner = centre + q_x along + q_y across

- Past 3 sizes: weight < 0.01.

A square on the disc, 3 sizes out each way



corner = centre + q_x along + q_y across



on the screen

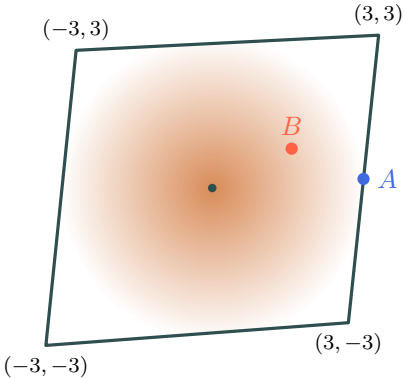
- Past 3 sizes: weight < 0.01.
- Corners through u.mvp.

Worked: q at two pixels, and their weights

The GPU blends q across the square (lecture 29, slide 9).

Try first: q and the weight at A and B.

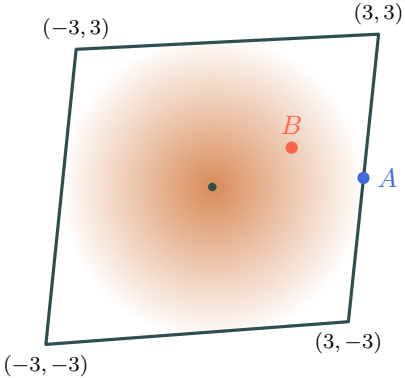
A: middle of an edge. B: half way to a corner.



Worked: q at two pixels, and their weights

The GPU blends q across the square (lecture 29, slide 9).

Try first: q and the weight at A and B.



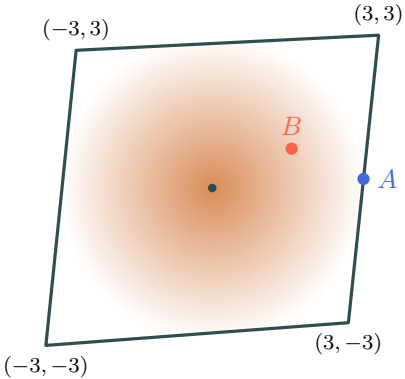
A: middle of an edge. B: half way to a corner.

A: $q = (3, 0)$, $q \cdot q = 9$, weight $e^{-4.5} = 0.011$

Worked: q at two pixels, and their weights

The GPU blends q across the square (lecture 29, slide 9).

Try first: q and the weight at A and B.



A: middle of an edge. B: half way to a corner.

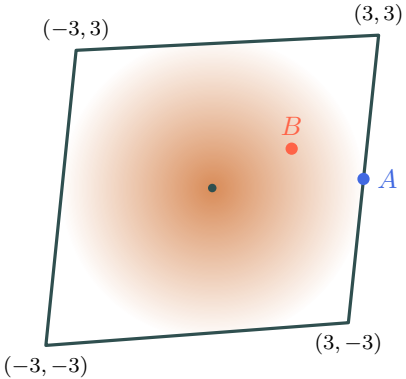
A: $q = (3, 0)$, $q \cdot q = 9$, weight $e^{-4.5} = 0.011$

B: $q = (1.5, 1.5)$, $q \cdot q = 4.5$, weight $e^{-2.25} = 0.105$

Worked: q at two pixels, and their weights

The GPU blends q across the square (lecture 29, slide 9).

Try first: q and the weight at A and B.



A: middle of an edge. B: half way to a corner.

A: $q = (3, 0)$, $q \cdot q = 9$, weight $e^{-4.5} = 0.011$

B: $q = (1.5, 1.5)$, $q \cdot q = 4.5$, weight $e^{-2.25} = 0.105$

Lecture 35's formula, on q.

vs_main: a square around each blob

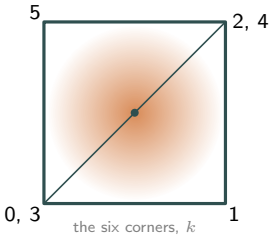
```
fn vs_main(@builtin(vertex_index) k:  u32,        k: which corner, 0 to 5
           @location(0) centre:  vec3f, @location(1) along:  vec3f,
           @location(2) across:  vec3f, ...)  -> Splat {
```

▶ **Run it in the Splat Lab:**
Suzanne as blobs, by normal

vs_main: a square around each blob

```
fn vs_main(@builtin(vertex_index) k:  u32,        k: which corner, 0 to 5
           @location(0) centre:  vec3f, @location(1) along:  vec3f,
           @location(2) across:  vec3f, ...)  -> Splat {

    var square = array<vec2f, 6>(
        vec2f(-1.0, -1.0), vec2f(1.0, -1.0), vec2f(1.0, 1.0),
        vec2f(-1.0, -1.0), vec2f(1.0, 1.0), vec2f(-1.0, 1.0));
```



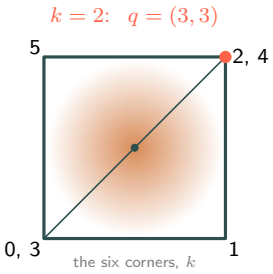
▶ **Run it in the Splat Lab:**
Suzanne as blobs, by normal

vs_main: a square around each blob

```
fn vs_main(@builtin(vertex_index) k: u32,        k: which corner, 0 to 5
           @location(0) centre: vec3f, @location(1) along: vec3f,
           @location(2) across: vec3f, ...) -> Splat {

    var square = array<vec2f, 6>(
        vec2f(-1.0, -1.0), vec2f(1.0, -1.0), vec2f(1.0, 1.0),
        vec2f(-1.0, -1.0), vec2f(1.0, 1.0), vec2f(-1.0, 1.0));

    let reach = 3.0;
    let q = reach * square[k];
```



▶ **Run it in the Splat Lab:**
Suzanne as blobs, by normal

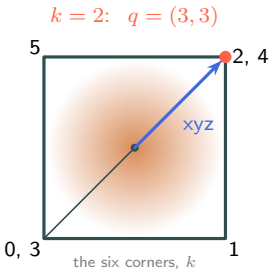
vs_main: a square around each blob

```
fn vs_main(@builtin(vertex_index) k: u32,        k: which corner, 0 to 5
           @location(0) centre: vec3f, @location(1) along: vec3f,
           @location(2) across: vec3f, ...) -> Splat {

    var square = array<vec2f, 6>(
        vec2f(-1.0, -1.0), vec2f(1.0, -1.0), vec2f(1.0, 1.0),
        vec2f(-1.0, -1.0), vec2f(1.0, 1.0), vec2f(-1.0, 1.0));

    let reach = 3.0;
    let q = reach * square[k];

    let grow = 1.0;
    let xyz = centre + grow * (q.x * along + q.y * across);
```



▶ **Run it in the Splat Lab:**
Suzanne as blobs, by normal

vs_main: a square around each blob

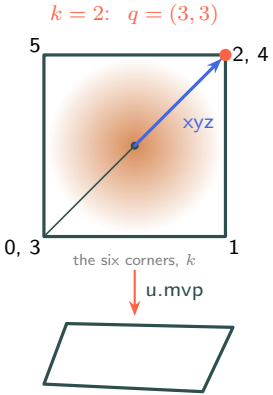
```
fn vs_main(@builtin(vertex_index) k: u32,        k: which corner, 0 to 5
           @location(0) centre: vec3f, @location(1) along: vec3f,
           @location(2) across: vec3f, ...) -> Splat {

    var square = array<vec2f, 6>(
        vec2f(-1.0, -1.0), vec2f(1.0, -1.0), vec2f(1.0, 1.0),
        vec2f(-1.0, -1.0), vec2f(1.0, 1.0), vec2f(-1.0, 1.0));

    let reach = 3.0;
    let q = reach * square[k];

    let grow = 1.0;
    let xyz = centre + grow * (q.x * along + q.y * across);

    out.pos = u.mvp * vec4f(xyz.x, xyz.y, xyz.z, 1.0);   lecture 28, unchanged
    out.q = q;
    ... P, normal, colour, opacity, as in lecture 29 ...
```



▶ **Run it in the Splat Lab:**
Suzanne as blobs, by normal

fs_main: lecture 35's weight, and a

```
fn fs_main(@location(0) q:  vec2f, ...)  -> @location(0) vec4f {
    let weight = exp(-0.5 * dot(q, q));     lecture 35 (slide 17)
    let a = opacity * weight;               lecture 35 (slide 11)




}
```

▶ **Run it in the Splat Lab:**
Suzanne as blobs, by normal

fs_main: lecture 35's weight, and a

```
fn fs_main(@location(0) q:  vec2f, ...)  -> @location(0) vec4f {
    let weight = exp(-0.5 * dot(q, q));      lecture 35 (slide 17)
    let a = opacity * weight;                lecture 35 (slide 11)

    let n = normalize(normal);
    let c = 0.5 * n + 0.5;                   lecture 29 (slide 10)


}
```

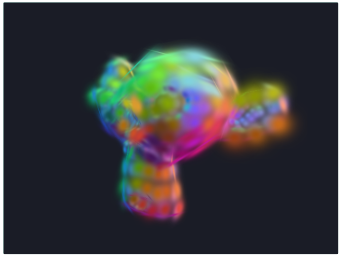
▶ **Run it in the Splat Lab:**
Suzanne as blobs, by normal

fs_main: lecture 35's weight, and a

```
fn fs_main(@location(0) q:  vec2f, ...)  -> @location(0) vec4f {
    let weight = exp(-0.5 * dot(q, q));      lecture 35 (slide 17)
    let a = opacity * weight;                lecture 35 (slide 11)

    let n = normalize(normal);
    let c = 0.5 * n + 0.5;                   lecture 29 (slide 10)

    return vec4f(c.r, c.g, c.b, a);          NEW: a goes out too
}
```

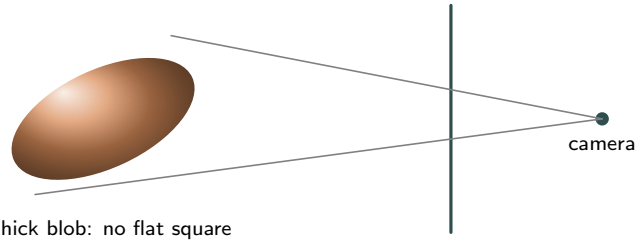


splat.wgsl

Lecture 35: `mix` in the shader.
Today: the GPU blends (Round 3).

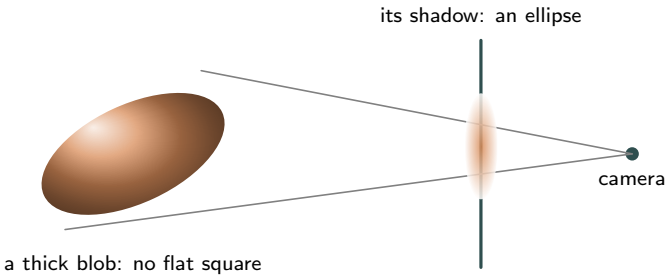
▶ **Run it in the Splat Lab:**
Suzanne as blobs, by normal

Thick blobs: the paper's way (for reading)



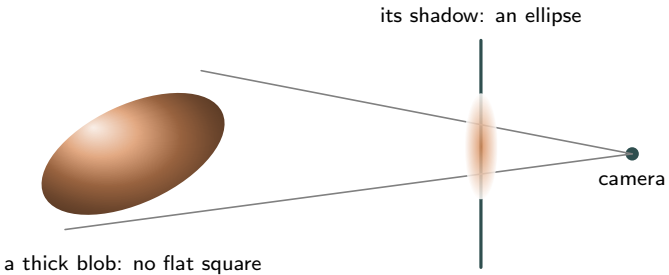
a thick blob: no flat square

Thick blobs: the paper's way (for reading)



- Thick: still an ellipse.

Thick blobs: the paper's way (for reading)

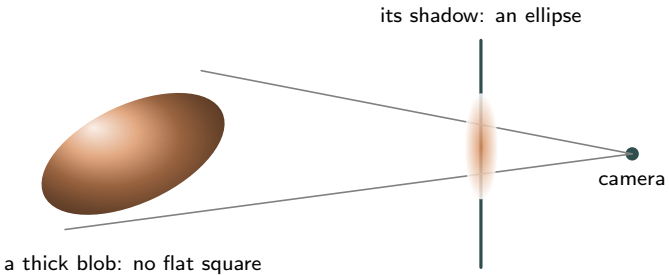


- Thick: still an ellipse.

$$\Sigma' = J W \Sigma W^{\mathsf{T}} J^{\mathsf{T}}$$

W: V M. J: the perspective. Σ': the ellipse.

Thick blobs: the paper's way (for reading)



- Thick: still an ellipse.
- Flat discs: the square does it.

$$\Sigma' = J W \Sigma W^{\mathsf{T}} J^{\mathsf{T}}$$

W: V M. J: the perspective. Σ': the ellipse.

Zwicker, Pfister, van Baar, Gross, *EWA Volume Splatting*, IEEE Visualization 2001. Kerbl et al. 2023, section 4.

Flat discs for surfaces: Huang et al., *2D Gaussian Splatting for Geometrically Accurate Radiance Fields*, SIGGRAPH 2024.

You Try 3: a short square

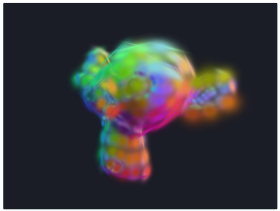
In the Splat Lab, choose **You Try 3**. Guess first, then change the code.

```
let reach = 3.0;
let q = reach * square[k];
```

change this line

- **Reach 1 size, not 3.**
- Guess: the blob's edge?

now



after your change



You Try 3: a short square

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

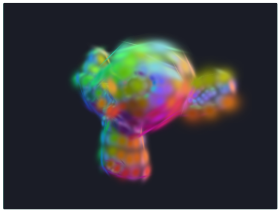
```
let reach = 3.0;
let q = reach * square[k];
```

change this line

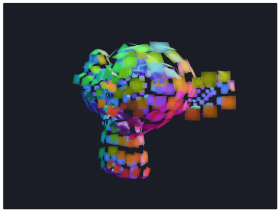
answer: `let reach = 1.0;`

- **Reach 1 size, not 3.**
- Guess: the blob's edge?
- Cut off square: the weight there is still 0.61.

now



after your change



You Try 4: a hard edge

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

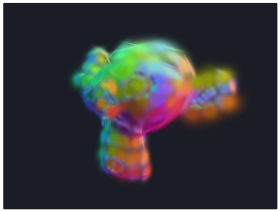
```
let weight = exp(-0.5 * dot(q, q));
let a = opacity * weight;
```

change this line

lecture 26's step

- **1 inside one size, 0 outside.**
- Guess: still covered?

now



after your change



You Try 4: a hard edge

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

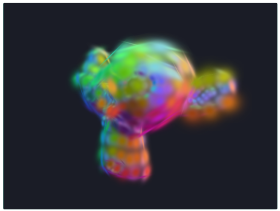
```
let weight = exp(-0.5 * dot(q, q));
```
change this line

```
let a = opacity * weight;
```

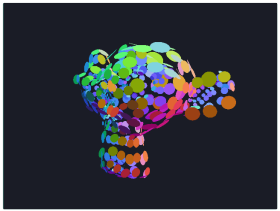
lecture 26's step

answer: `let weight = 1.0 - step(1.0, length(q));`

now



after your change



- **1 inside one size, 0 outside.**
- Guess: still covered?
- No: sequins. The soft tails filled the gaps.

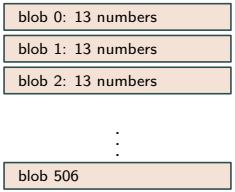
Round 3

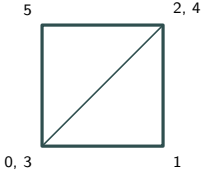
Drawing 507 blobs

one square, many times, blended in order

One square, drawn 507 times

the buffer: a row a blob

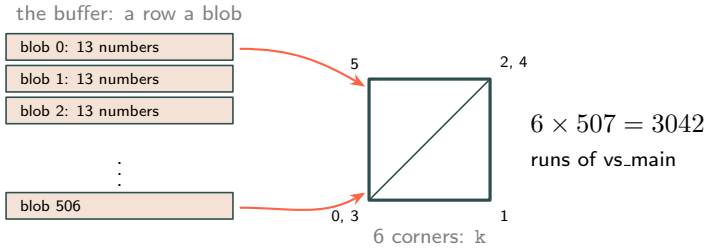




6 corners: `k`

- A row a blob.

One square, drawn 507 times

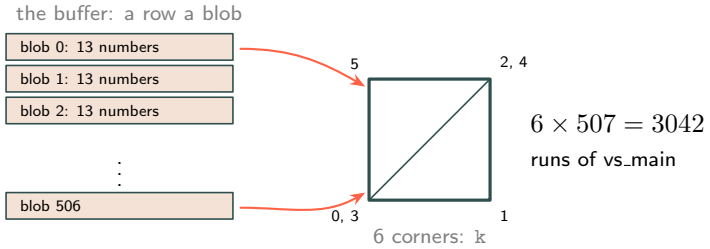


$6 \times 507 = 3042$

runs of vs_main

- A row a blob.
- The same square each time.

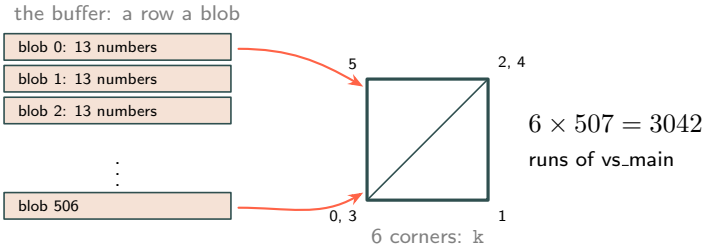
One square, drawn 507 times



- A row a blob.
- The same square each time.
- **Instancing**.

```
"step_mode":  "instance",          # NEW: a row a square
render_pass.draw(6, len(blobs))    # 6 corners, 507 times
```

One square, drawn 507 times



$6 \times 507 = 3042$

runs of vs_main

- A row a blob.
- The same square each time.
- **Instancing**.

```
"step_mode":  "instance",          # NEW: a row a square
render_pass.draw(6, len(blobs))    # 6 corners, 507 times
```

Lecture 29: a row a corner. Today: a row a square.

The GPU blends each square over the picture so far

```
blend_over = {"color": {"src_factor": "src-alpha",
                        "dst_factor": "one-minus-src-alpha", ...}}
```

$$\text{pixel} = a \cdot c + (1 - a) \cdot \text{pixel}$$ lecture 35's `mix`

The GPU blends each square over the picture so far

```
blend_over = {"color": {"src_factor": "src-alpha",
                        "dst_factor": "one-minus-src-alpha", ...}}
```

$$\text{pixel} = a \cdot c + (1 - a) \cdot \text{pixel}$$ lecture 35's `mix`

Depth buffer: off.

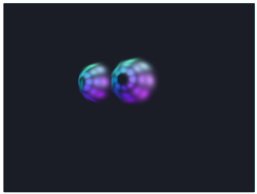
The GPU blends each square over the picture so far

```
blend_over = {"color":  {"src_factor":  "src-alpha",
                         "dst_factor":  "one-minus-src-alpha", ...}}
```

$$\text{pixel} \;=\; a \cdot c \;+\; (1-a) \cdot \text{pixel}$$ lecture 35's `mix`

Depth buffer: off.

in the file's order



after 60

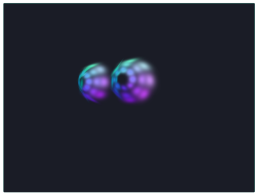
The GPU blends each square over the picture so far

```
blend_over = {"color":  {"src_factor":  "src-alpha",
                         "dst_factor":  "one-minus-src-alpha", ...}}
```

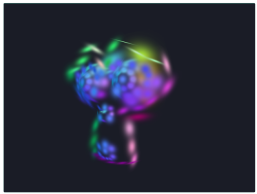
$$\text{pixel} \;=\; a \cdot c \;+\; (1-a) \cdot \text{pixel}$$ lecture 35's `mix`

Depth buffer: off.

in the file's order



after 60



after 150

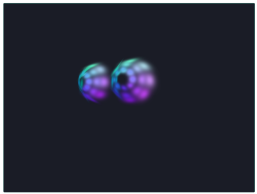
The GPU blends each square over the picture so far

```
blend_over = {"color": {"src_factor": "src-alpha",
                        "dst_factor": "one-minus-src-alpha", ...}}
```

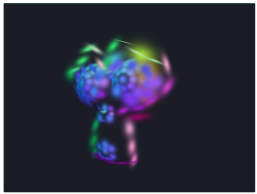
$$\text{pixel} = a \cdot c + (1 - a) \cdot \text{pixel}$$ lecture 35's `mix`

Depth buffer: off.

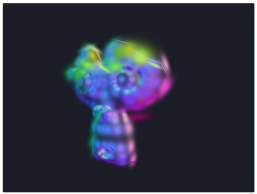
in the file's order



after 60



after 150



after 300

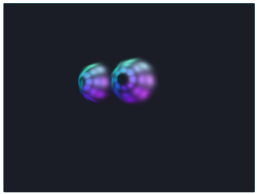
The GPU blends each square over the picture so far

```
blend_over = {"color":  {"src_factor":  "src-alpha",
                         "dst_factor":  "one-minus-src-alpha", ...}}
```

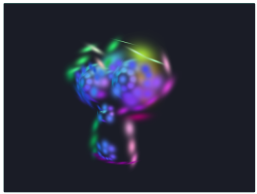
$$\text{pixel} \;=\; a \cdot c \;+\; (1-a) \cdot \text{pixel}$$ lecture 35's `mix`

Depth buffer: off.

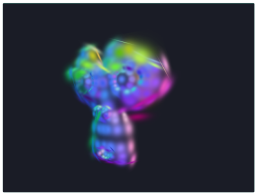
in the file's order



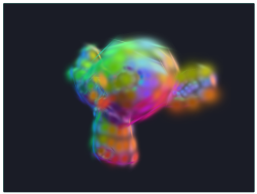
after 60



after 150



after 300



after 507

You Try 5: switch depth writing on

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

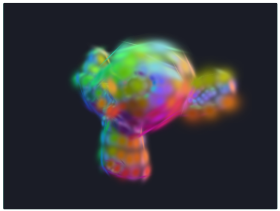
```
// depth writing:  off
```

off becomes on

In the lab: tick **Depth writing**.

- **Depth writing on.**
- Guess: better or worse?

now



after your change



You Try 5: switch depth writing on

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

```
// depth writing:  off
```

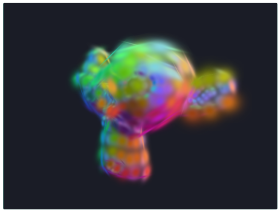
off becomes on

In the lab: tick **Depth writing**.

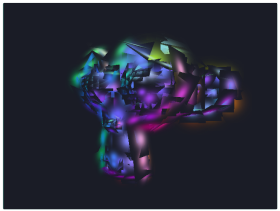
answer: `// depth writing:  on`

- **Depth writing on.**
- Guess: better or worse?
- Worse: faint corners hide what comes later.

now



after your change



You Try 6: light the blobs

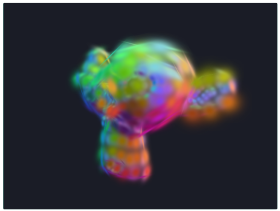
In the Splat Lab, choose **You Try 6**. Guess first, then change the code.

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

change this line

- **Lecture 29's lamp.**
- Guess: like lecture 29?

now



after your change



You Try 6: light the blobs

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

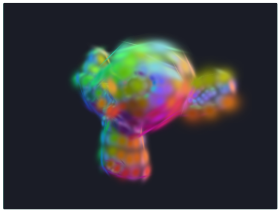
```
let n = normalize(normal);
let c = 0.5 * n + 0.5;
```

change this line

answer:

```
let l = normalize(vec3f(-3.0, 3.0, 3.0) - P);
let c = colour * (0.1 + max(dot(n, l), 0.0));
```

now



after your change



- **Lecture 29's lamp.**
- Guess: like lecture 29?
- Dark patches: back blobs drawn last.

What blobs get wrong: the file's order

from the front



the file's order

- Dark patches.

What blobs get wrong: the file's order

from the front



the file's order



far to near

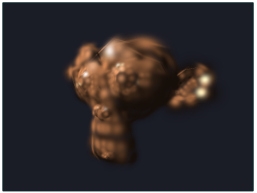
- Dark patches.

What blobs get wrong: the file's order

from the front

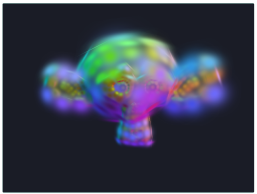


the file's order



far to near

from behind



the file's order

- Dark patches.
- Her face shows through.

What blobs get wrong: the file's order

from the front

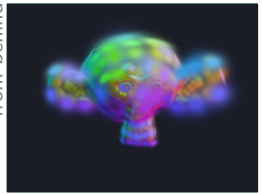


the file's order

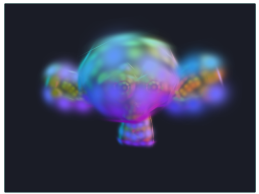


far to near

from behind



the file's order



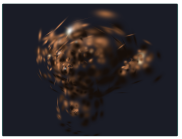
far to near

- Dark patches.
- Her face shows through.
- Lecture 37: sort every frame.

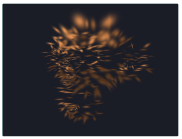
Now make your own picture



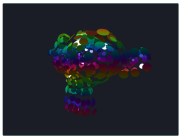
A painting
`long strokes, each turned`



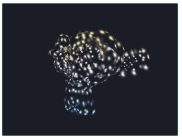
She bursts and comes back
`blobs fly along n`



A furry Suzanne
`blobs stand up and sway`



Glitter
`hard, crooked sequins`



A constellation
`tiny blobs that twinkle`



She melts
`blobs slide down, stretch`

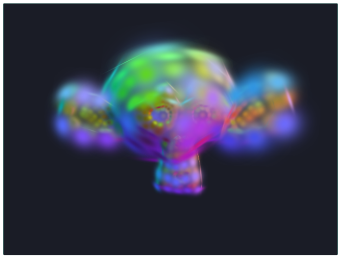
All six are in the Splat Lab, under **Ideas**. Change one. Mix two. Show your neighbour.

Summary

1	A blob in 3D	three arrows: a turn and three sizes
2	Flat blobs	a disc on each of her 507 points
3	Onto the screen	a square, through u.mvp; weight on q
4	Drawing them	instancing; the GPU blends
5	What it gets wrong	the file's order: her face shows through

Next lecture: order, and a real scene

today: the file's order



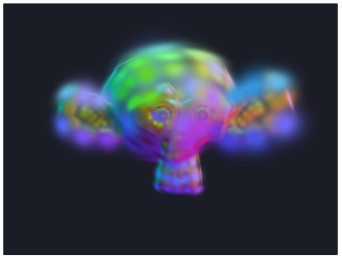
lecture 37: a captured place



- Sort, far to near.

Next lecture: order, and a real scene

today: the file's order



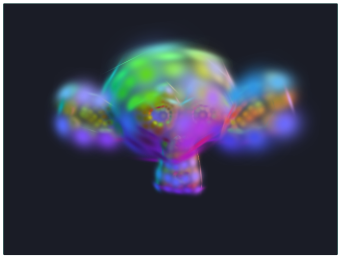
lecture 37: a captured place

?

- Sort, far to near.
- A real scene: a million blobs.

Next lecture: order, and a real scene

today: the file's order



lecture 37: a captured place

?

- Sort, far to near.
- A real scene: a million blobs.
- Can a laptop keep up?

Exit Ticket

Today's question *(2 minutes)*

(a) A pixel lands on a blob's disc at $q = (1, 2)$. What is its weight?

(b) A scene has a million flat blobs, each drawn as one square of two triangles. How many times does vs_main run each frame?

CS 453 — Exit Ticket #36

Name: _____

Roll #: _____ Date: _____

Answer:

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