Extra 7: Learned Scenes

Photographs In, a Scene Out

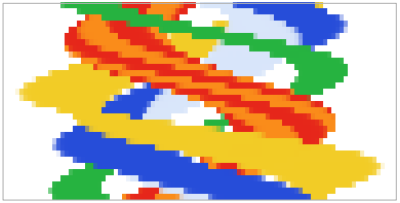
CS 453 — Computer Graphics

Fakhir Shaheen

Forman Christian University

Today: photographs in, a scene out

Extra 7: self-study, not on the exams

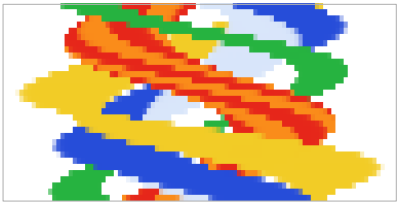


32 photographs (one row each)

- Until now: a scene given, pictures out.

Today: photographs in, a scene out

Extra 7: self-study, not on the exams



32 photographs (one row each)

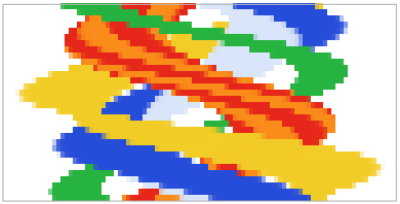


the scene found, from above

- Until now: a scene given, pictures out.
- Today: pictures given, the scene out.

Today: photographs in, a scene out

Extra 7: self-study, not on the exams



32 photographs (one row each)

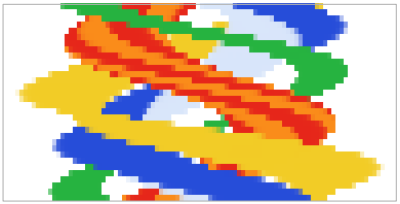


the scene found, from above

- Until now: a scene given, pictures out.
- Today: pictures given, the scene out.
- Guess, render, compare, nudge; repeat.

Today: photographs in, a scene out

Extra 7: self-study, not on the exams



32 photographs (one row each)



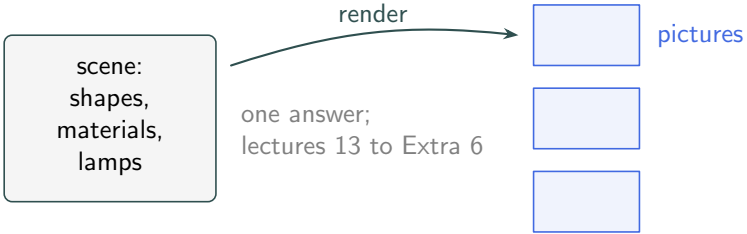
the scene found, from above

- Until now: a scene given, pictures out.
- Today: pictures given, the scene out.
- Guess, render, compare, nudge; repeat.
- Then any new view is a render.

The inverse problem

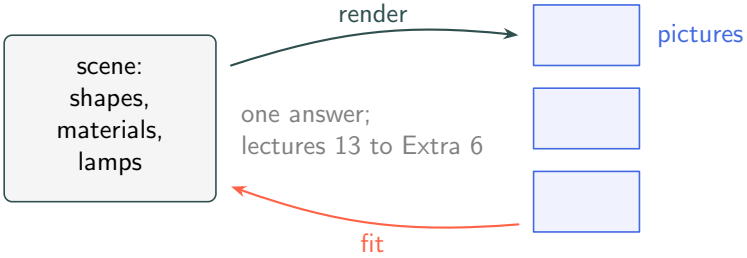
From pictures back to a scene

Rendering, run backwards



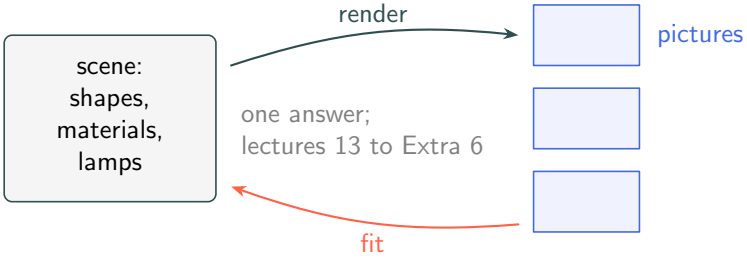
- Rendering: scene in, pictures out.

Rendering, run backwards



- Rendering: scene in, pictures out.
- Backwards: pictures in, a scene that would give them.

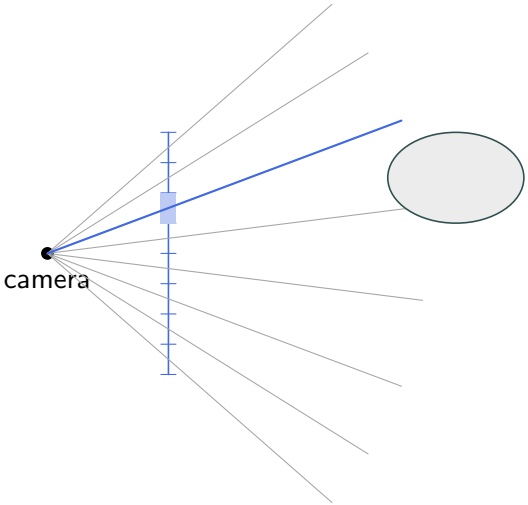
Rendering, run backwards



- Rendering: scene in, pictures out.
- Backwards: pictures in, a scene that would give them.
- No formula runs it backwards: we search.

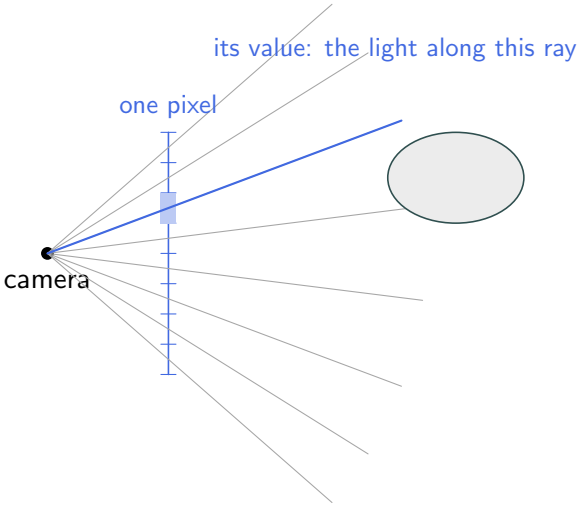
Guess a scene, render it, compare, change the guess.

A photograph: one number for each ray



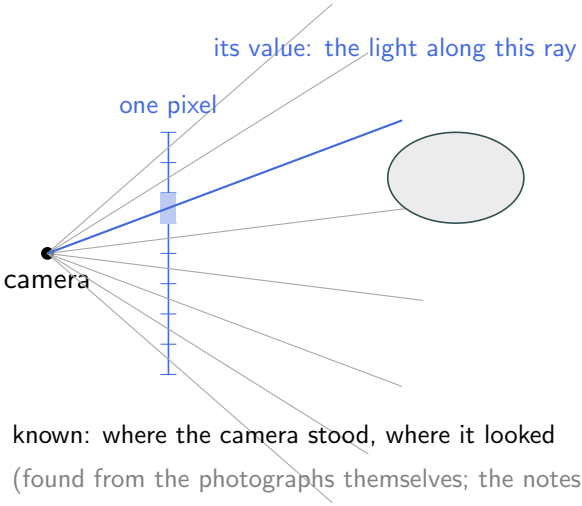
- A camera: a fan of rays.

A photograph: one number for each ray



- A camera: a fan of rays.
- Each pixel: the light along one ray.

A photograph: one number for each ray



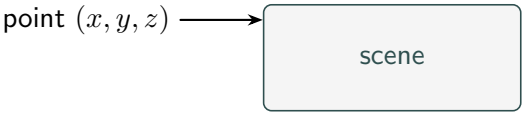
known: where the camera stood, where it looked

(found from the photographs themselves; the notes)

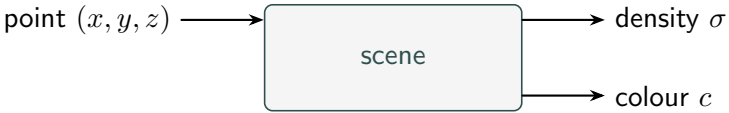
- A camera: a fan of rays.
- Each pixel: the light along one ray.
- A million pixels: a million measurements.

A scene as a function of position and direction

- Ask any point.



A scene as a function of position and direction



- Ask any point.
- It answers: how dense, what colour.

σ: how much stuff is here (Extra 5's σ_t); c: the light it sends

A scene as a function of position and direction



- Ask any point.
- It answers: how dense, what colour.

σ: how much stuff is here (Extra 5's σ_t); c: the light it sends

the colour depends on the direction: shine looks different from each side

A scene as a function of position and direction



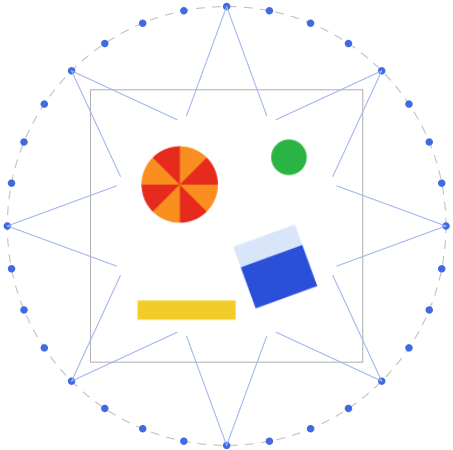
- Ask any point.
- It answers: how dense, what colour.
- The lamps are baked into the colour.

σ: how much stuff is here (Extra 5's σ_t); c: the light it sends

the colour depends on the direction: shine looks different from each side

Five numbers in, four out: no shapes, no materials, no lamps.

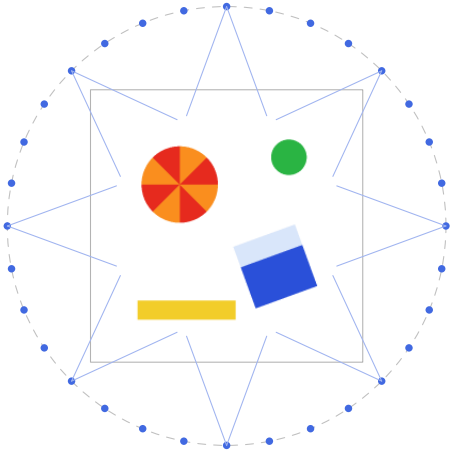
Flatland: a scene small enough to see whole



32 cameras round it

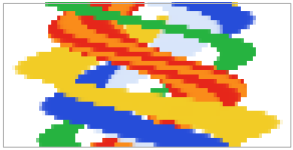
- A 2D world: shapes on a table, seen from above.

Flatland: a scene small enough to see whole



32 cameras round it

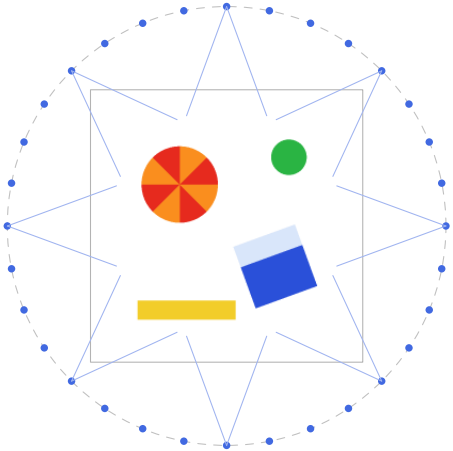
one row of 96 pixels each, stacked



their photographs

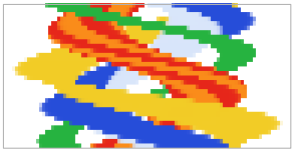
- A 2D world: shapes on a table, seen from above.
- A photograph is one row.

Flatland: a scene small enough to see whole



32 cameras round it

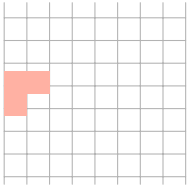
one row of 96 pixels each, stacked



their photographs

- A 2D world: shapes on a table, seen from above.
- A photograph is one row.
- All of today's ideas, small enough to fit in seconds.

Worked: unknowns against measurements

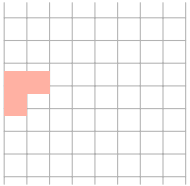


a grid over the world

The unknowns: a 48×48 grid, 4 numbers a point

σ, red, green, blue; in between, read bilinearly

Worked: unknowns against measurements



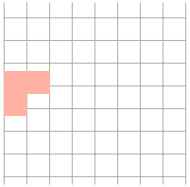
a grid over the world

The unknowns: a 48×48 grid, 4 numbers a point

σ, red, green, blue; in between, read bilinearly

$48 \times 48 \times 4 = 9{,}216$ unknowns

Worked: unknowns against measurements



a grid over the world

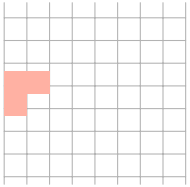
The unknowns: a 48×48 grid, 4 numbers a point

σ, red, green, blue; in between, read bilinearly

$48 \times 48 \times 4 = 9{,}216$ unknowns

32 photographs: $32 \times 96 \times 3 = \mathbf{9{,}216}$ measurements

Worked: unknowns against measurements



a grid over the world

The unknowns: a 48×48 grid, 4 numbers a point

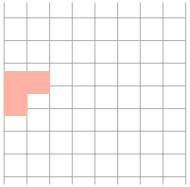
σ, red, green, blue; in between, read bilinearly

$48 \times 48 \times 4 = 9{,}216$ unknowns

32 photographs: $32 \times 96 \times 3 = \mathbf{9{,}216}$ measurements

4 photographs: $4 \times 96 \times 3 = \mathbf{1{,}152}$ measurements

Worked: unknowns against measurements



a grid over the world

The unknowns: a 48×48 grid, 4 numbers a point

σ, red, green, blue; in between, read bilinearly

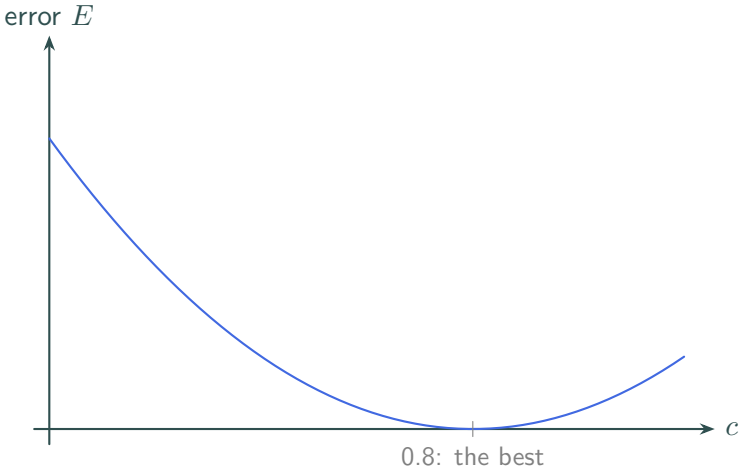
$48 \times 48 \times 4 = 9{,}216$ unknowns

32 photographs: $32 \times 96 \times 3 = \mathbf{9{,}216}$ measurements

4 photographs: $4 \times 96 \times 3 = \mathbf{1{,}152}$ measurements

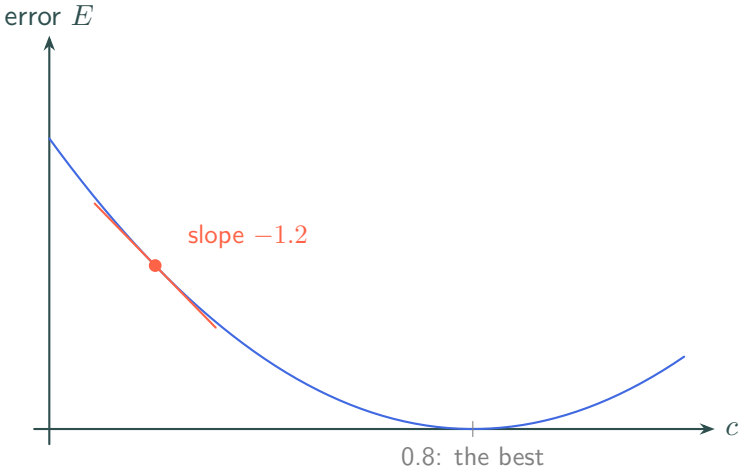
Fewer measurements than unknowns: many scenes fit, and the search picks one.

Gradient descent: walk downhill on the error



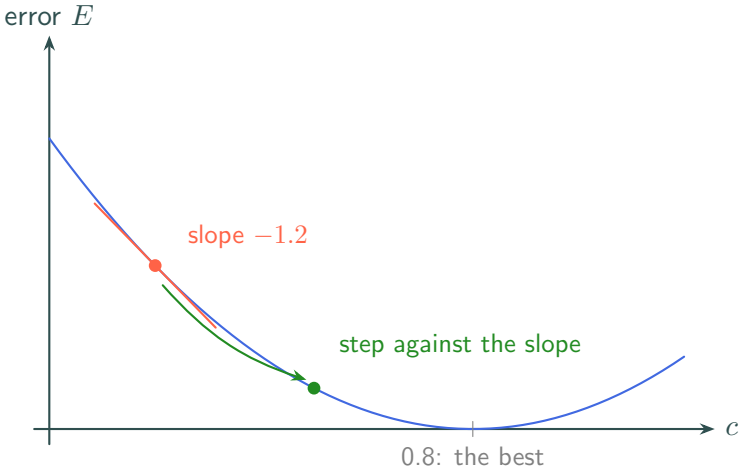
- One unknown c, one error E.

Gradient descent: walk downhill on the error



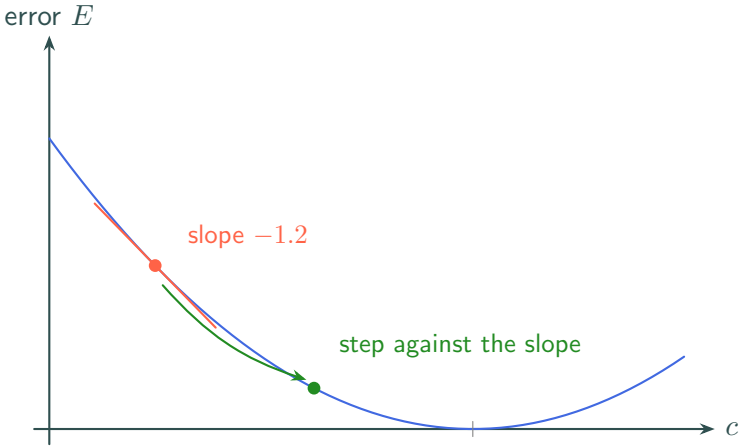
- One unknown c, one error E.
- The slope says which way is down.

Gradient descent: walk downhill on the error



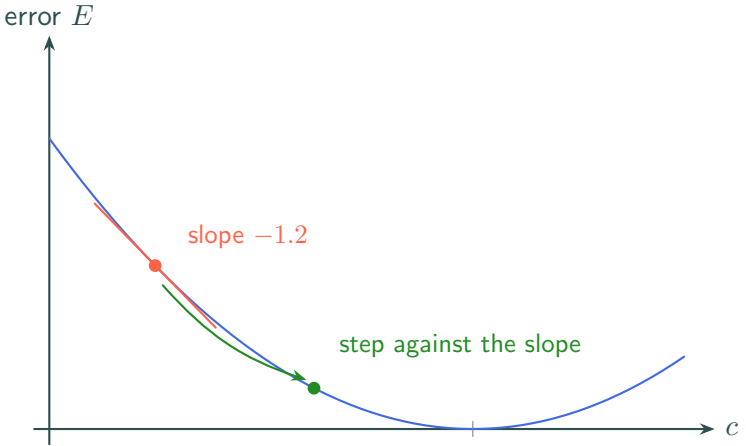
- One unknown c, one error E.
- The slope says which way is down.
- Step that way; repeat.

Gradient descent: walk downhill on the error



- One unknown c, one error E.
- The slope says which way is down.
- Step that way; repeat.

Gradient descent: walk downhill on the error



- One unknown c, one error E.
- The slope says which way is down.
- Step that way; repeat.
- 9,216 unknowns: 9,216 slopes, all at once.

Worked: one number, four steps

$$E = (c - 0.8)^2, \quad \frac{dE}{dc} = 2(c - 0.8), \quad \eta = 0.25$$

Try first: from $c = 0.2$, take two steps.

step	c	slope	new c

Worked: one number, four steps

$$E = (c - 0.8)^2, \quad \frac{dE}{dc} = 2(c - 0.8), \quad \eta = 0.25$$

Try first: from $c = 0.2$, take two steps.

step	c	slope	new c
1	0.2	-1.2	0.5

- $0.2 + 0.25 \times 1.2 = 0.5$.

Worked: one number, four steps

$$E = (c - 0.8)^2, \quad \frac{dE}{dc} = 2(c - 0.8), \quad \eta = 0.25$$

Try first: from $c = 0.2$, take two steps.

step	c	slope	new c
1	0.2	-1.2	0.5
2	0.5	-0.6	0.65

- $0.2 + 0.25 \times 1.2 = 0.5$.

Worked: one number, four steps

$$E = (c - 0.8)^2, \quad \frac{dE}{dc} = 2(c - 0.8), \quad \eta = 0.25$$

Try first: from $c = 0.2$, take two steps.

step	c	slope	new c
1	0.2	-1.2	0.5
2	0.5	-0.6	0.65
3	0.65	-0.3	0.725

- $0.2 + 0.25 \times 1.2 = 0.5$.

Worked: one number, four steps

$$E = (c - 0.8)^2, \quad \frac{dE}{dc} = 2(c - 0.8), \quad \eta = 0.25$$

Try first: from $c = 0.2$, take two steps.

step	c	slope	new c
1	0.2	-1.2	0.5
2	0.5	-0.6	0.65
3	0.65	-0.3	0.725
4	0.725	-0.15	0.7625

- $0.2 + 0.25 \times 1.2 = 0.5$.
- The gap halves every step.

Worked: one number, four steps

$$E = (c - 0.8)^2, \quad \frac{dE}{dc} = 2(c - 0.8), \quad \eta = 0.25$$

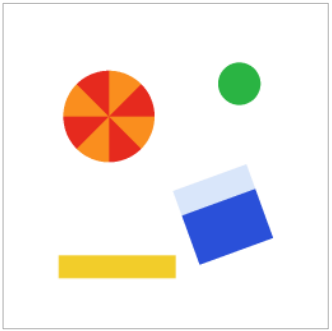
Try first: from $c = 0.2$, take two steps.

step	c	slope	new c
1	0.2	-1.2	0.5
2	0.5	-0.6	0.65
3	0.65	-0.3	0.725
4	0.725	-0.15	0.7625

- $0.2 + 0.25 \times 1.2 = 0.5$.
- The gap halves every step.
- Too big a step overshoots; too small crawls.

Rendered: four photographs against thirty-two

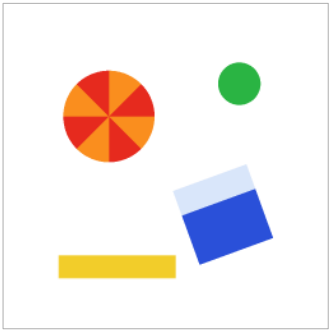
The 48×48 grid, fitted by gradient descent, seen from above



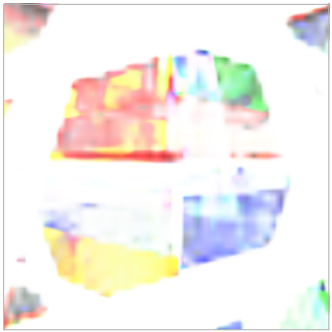
the truth

Rendered: four photographs against thirty-two

The 48 × 48 grid, fitted by gradient descent, seen from above



the truth

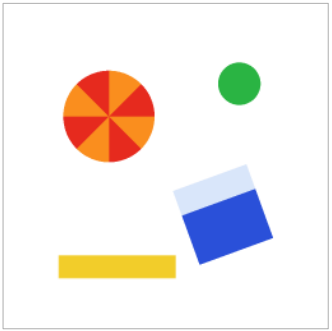


fitted to 4 photographs

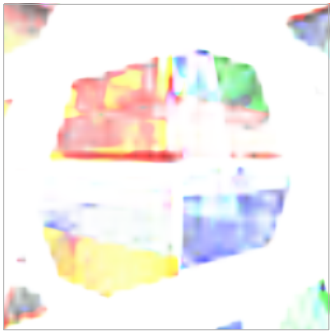
4: every photograph matched, the scene wrong.

Rendered: four photographs against thirty-two

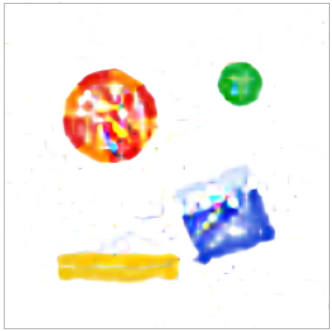
The 48 × 48 grid, fitted by gradient descent, seen from above



the truth



fitted to 4 photographs

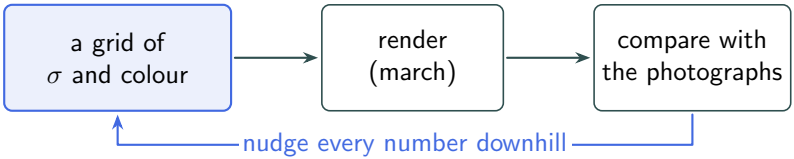


fitted to 32

4: every photograph matched, the scene wrong.

32: the shapes where they are.

Recap: the loop



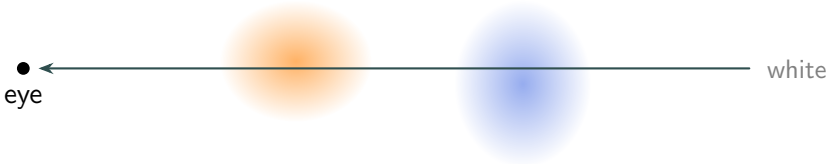
store	numbers in our fits	what one look-up costs
1 a grid (σ, colour)	9,216 (flatland)	4 reads

NeRF

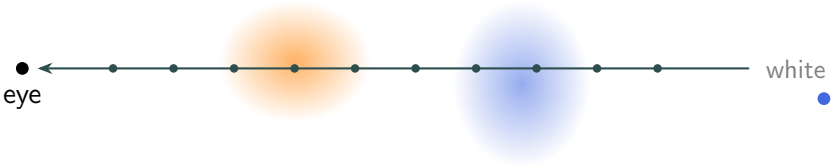
A fog that glows, and a network that stores it

A fog that glows

- Extra 5's medium, with light of its own.



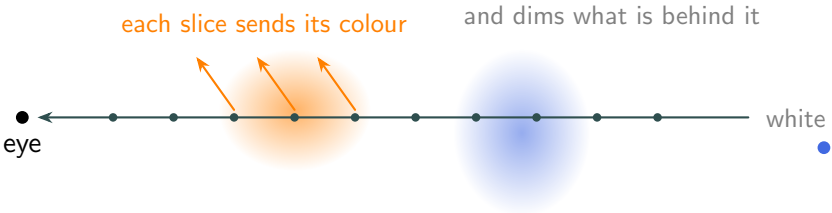
A fog that glows



samples along the ray, δ apart

- Extra 5's medium, with light of its own.
- No lamps, no shadow rays, no bounces.

A fog that glows



- Extra 5's medium, with light of its own.
- No lamps, no shadow rays, no bounces.
- The pixel: what reaches the eye.

The sum along a ray

$\alpha_i = 1 - e^{-\sigma_i \delta}$ slice i's opacity

- Extra 5's $e^{-\sigma\delta}$, one slice.

The sum along a ray

$\alpha_i = 1 - e^{-\sigma_i \delta}$ slice i's opacity

$T_i = (1 - \alpha_1)(1 - \alpha_2) \cdots (1 - \alpha_{i-1})$ what gets through in front

- Extra 5's $e^{-\sigma\delta}$, one slice.

The sum along a ray

$\alpha_i = 1 - e^{-\sigma_i \delta}$ slice i's opacity

$T_i = (1 - \alpha_1)(1 - \alpha_2) \cdots (1 - \alpha_{i-1})$ what gets through in front

$w_i = T_i \, \alpha_i$ slice i's share of the pixel

- Extra 5's $e^{-\sigma \delta}$, one slice.
- Shares add up to 1 with what is left.

The sum along a ray

$\alpha_i = 1 - e^{-\sigma_i \delta}$ slice i's opacity

$T_i = (1-\alpha_1)(1-\alpha_2)\cdots(1-\alpha_{i-1})$ what gets through in front

$w_i = T_i\,\alpha_i$ slice i's share of the pixel

$C = \sum_i w_i\,c_i \;+\; T_{\text{end}} \times \text{background}$

- Extra 5's $e^{-\sigma\delta}$, one slice.
- Shares add up to 1 with what is left.

The sum along a ray

$\alpha_i = 1 - e^{-\sigma_i \delta}$ slice i's opacity

$T_i = (1-\alpha_1)(1-\alpha_2)\cdots(1-\alpha_{i-1})$ what gets through in front

$w_i = T_i\,\alpha_i$ slice i's share of the pixel

$C = \sum_i w_i\,c_i \;+\; T_{\text{end}} \times \text{background}$

α as in a paint program's layers: the same sum.

- Extra 5's $e^{-\sigma\delta}$, one slice.
- Shares add up to 1 with what is left.
- Smooth in every σ and c: it has slopes.

Worked: four samples, one pixel

$\delta = 0.5$, a white background (1), one colour channel

Try first: the four αs, then the Ts.

i	σ_i	c_i	α_i	T_i	w_i	$w_i c_i$
1	0	0				
2	0.6	0.2				
3	3.0	0.9				
4	1.0	0.5				

Worked: four samples, one pixel

$\delta = 0.5$, a white background (1), one colour channel

Try first: the four αs, then the Ts.

i	σ_i	c_i	α_i	T_i	w_i	$w_i c_i$
1	0	0	0			
2	0.6	0.2	0.259			
3	3.0	0.9	0.777			
4	1.0	0.5	0.393			

Worked: four samples, one pixel

$\delta = 0.5$, a white background (1), one colour channel

Try first: the four αs, then the Ts.

i	σ_i	c_i	α_i	T_i	w_i	$w_i c_i$
1	0	0	0	1		
2	0.6	0.2	0.259	1		
3	3.0	0.9	0.777	0.741		
4	1.0	0.5	0.393	0.165		

Worked: four samples, one pixel

$\delta = 0.5$, a white background (1), one colour channel

Try first: the four αs, then the Ts.

i	σ_i	c_i	α_i	T_i	w_i	$w_i c_i$
1	0	0	0	1	0	0
2	0.6	0.2	0.259	1	0.259	0.052
3	3.0	0.9	0.777	0.741	0.576	0.518
4	1.0	0.5	0.393	0.165	0.065	0.033

- Sample 3 takes 58%: the surface is there.

Worked: four samples, one pixel

$\delta = 0.5$, a white background (1), one colour channel

Try first: the four αs, then the Ts.

i	σ_i	c_i	α_i	T_i	w_i	$w_i c_i$
1	0	0	0	1	0	0
2	0.6	0.2	0.259	1	0.259	0.052
3	3.0	0.9	0.777	0.741	0.576	0.518
4	1.0	0.5	0.393	0.165	0.065	0.033

left over: $T_{\text{end}} = 0.165 \times 0.607 = 0.100$, times white

- Sample 3 takes 58%: the surface is there.

Worked: four samples, one pixel

$\delta = 0.5$, a white background (1), one colour channel

Try first: the four αs, then the Ts.

i	σ_i	c_i	α_i	T_i	w_i	$w_i c_i$
1	0	0	0	1	0	0
2	0.6	0.2	0.259	1	0.259	0.052
3	3.0	0.9	0.777	0.741	0.576	0.518
4	1.0	0.5	0.393	0.165	0.065	0.033

left over: $T_{\text{end}} = 0.165 \times 0.607 = 0.100$, times white

$C = 0.052 + 0.518 + 0.033 + 0.100 = \mathbf{0.703}$

- Sample 3 takes 58%: the surface is there.
- $0.259 + 0.576 + 0.065 + 0.100 = 1.$

The error runs back along the ray

The photograph says 0.8; we rendered 0.703.

The error runs back along the ray

The photograph says 0.8; we rendered 0.703.

$$E = (C - 0.8)^2, \quad \frac{dE}{dC} = 2(0.703 - 0.8) = -0.195$$

The error runs back along the ray

The photograph says 0.8; we rendered 0.703.

$$E = (C - 0.8)^2, \quad \frac{dE}{dC} = 2(0.703 - 0.8) = -0.195$$

colour: $\dfrac{dC}{dc_3} = w_3 = 0.576, \quad \dfrac{dE}{dc_3} = -0.195 \times 0.576 = -0.112$

negative: raise c_3

The error runs back along the ray

The photograph says 0.8; we rendered 0.703.

$$E = (C - 0.8)^2, \quad \frac{dE}{dC} = 2(0.703 - 0.8) = -0.195$$

colour: $\dfrac{dC}{dc_3} = w_3 = 0.576, \quad \dfrac{dE}{dc_3} = -0.195 \times 0.576 = -0.112$

negative: raise c_3

density: $\dfrac{dC}{d\sigma_3} = \delta\,(T_4\,c_3 - \text{all behind}) = 0.5\,(0.165 \times 0.9 - 0.133) = +0.008$

more stuff at 3: more of its 0.9, less of the dimmer light behind

The error runs back along the ray

The photograph says 0.8; we rendered 0.703.

$$E = (C - 0.8)^2, \quad \frac{dE}{dC} = 2(0.703 - 0.8) = -0.195$$

colour: $\dfrac{dC}{dc_3} = w_3 = 0.576, \quad \dfrac{dE}{dc_3} = -0.195 \times 0.576 = -0.112$

negative: raise c_3

density: $\dfrac{dC}{d\sigma_3} = \delta\,(T_4\,c_3 - \text{all behind}) = 0.5\,(0.165 \times 0.9 - 0.133) = +0.008$

more stuff at 3: more of its 0.9, less of the dimmer light behind

Every sample on every ray adds its slope to the numbers it read.

Rendered: the fit, step by step

Fitted to the 32 photographs; each step, 256 random rays



after 100 steps

Rendered: the fit, step by step

Fitted to the 32 photographs; each step, 256 random rays



after 100 steps



after 500

Rendered: the fit, step by step

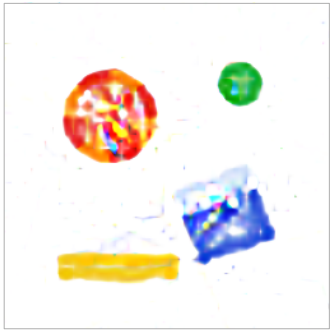
Fitted to the 32 photographs; each step, 256 random rays



after 100 steps



after 500

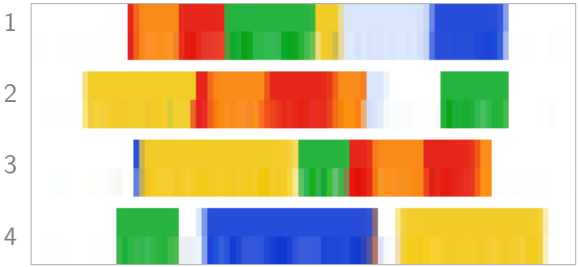


after 4,000

Faint blobs first, then shapes, then edges.

Rendered: cameras the fit never saw

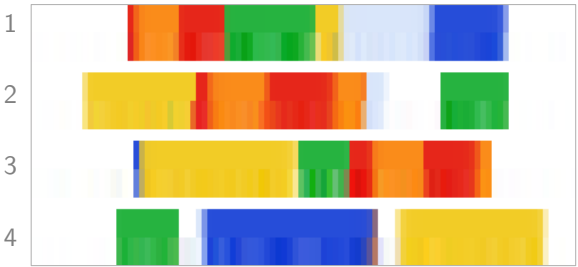
4 new cameras, each half-way between two of the 32



- Each pair: the true photograph above, the fit's below.

Rendered: cameras the fit never saw

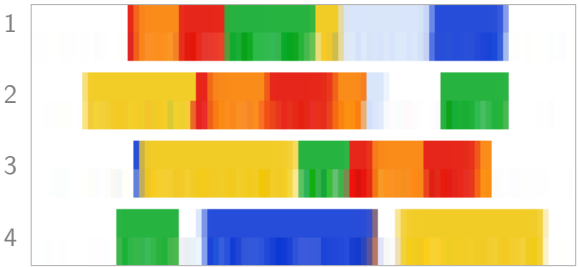
4 new cameras, each half-way between two of the 32



- Each pair: the true photograph above, the fit's below.
- The same shapes, in the same places.

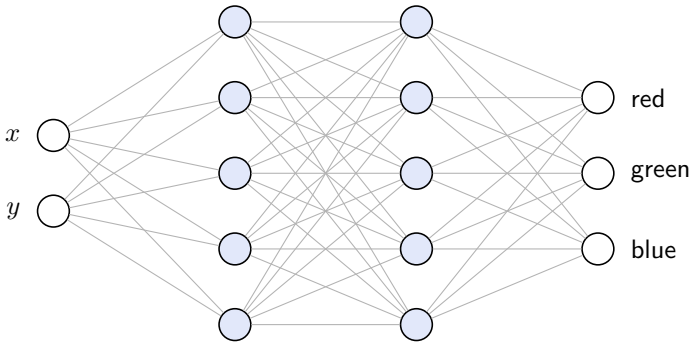
Rendered: cameras the fit never saw

4 new cameras, each half-way between two of the 32



- Each pair: the true photograph above, the fit's below.
- The same shapes, in the same places.
- A new view is just a render of the fit.

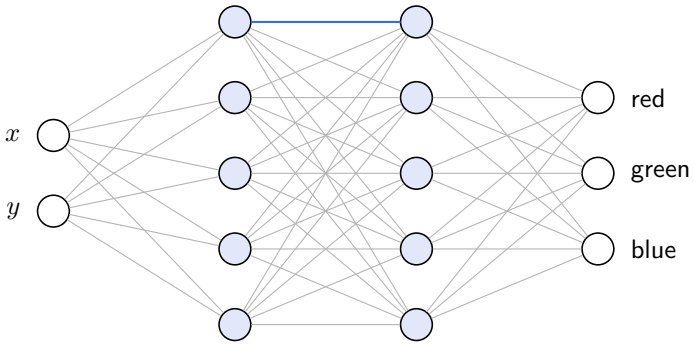
A network: a function made of weights



- Numbers in, numbers out.

A network: a function made of weights

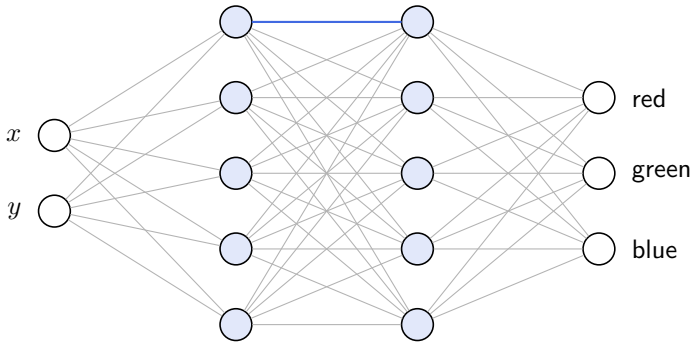
each line: a weight



- Numbers in, numbers out.
- The weights are the unknowns.

A network: a function made of weights

each line: a weight

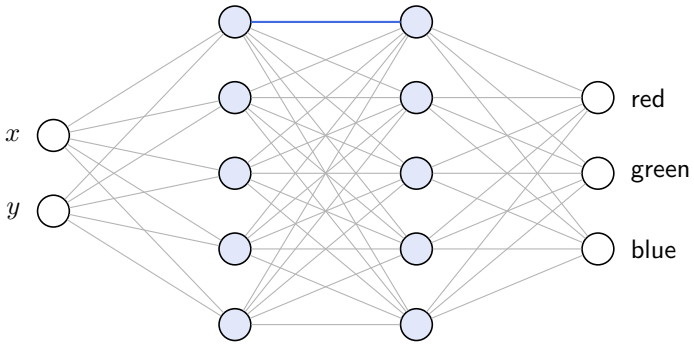


- Numbers in, numbers out.
- The weights are the unknowns.

a circle: $\max(0,$ weighted sum of the circles on its left $+$ a bias$)$

A network: a function made of weights

each line: a weight



a circle: $\max(0,$ weighted sum of the circles on its left $+$ a bias$)$

- Numbers in, numbers out.
- The weights are the unknowns.
- Slopes for every weight: the chain rule, layer by layer.

Rendered: a network fed (x, y) is blurry

Fit one picture: the network gets a pixel's place, must give its colour



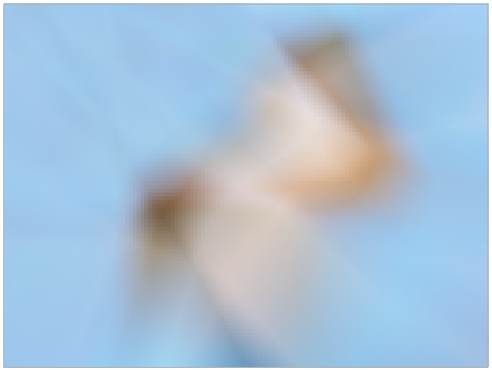
the picture: 128×96 pixels

Rendered: a network fed (x, y) is blurry

Fit one picture: the network gets a pixel's place, must give its colour



the picture: 128×96 pixels



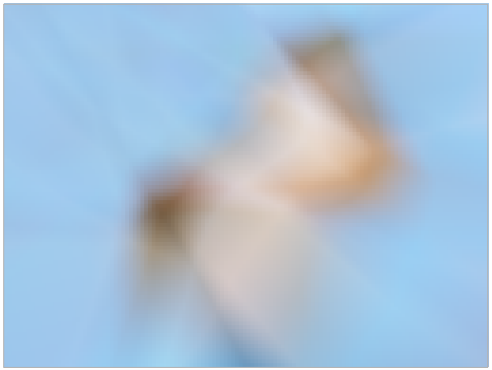
a network, $2 \to 64 \to 64 \to 3$: 4,547 weights

Rendered: a network fed (x, y) is blurry

Fit one picture: the network gets a pixel's place, must give its colour



the picture: 128×96 pixels

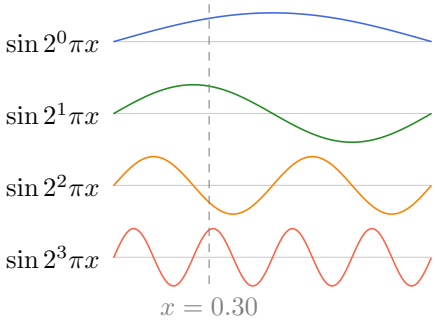


a network, $2 \to 64 \to 64 \to 3$: 4,547 weights

A cream smear on blue: no spots, no eyes, no legs.

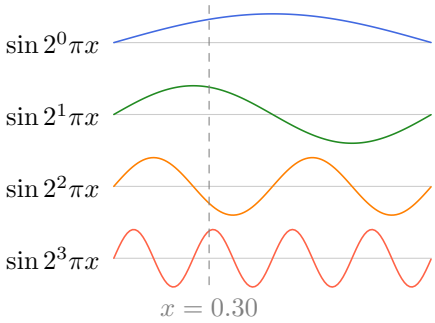
Positional encoding: sines that double

$$\gamma(x) = \left(x,\ \sin \pi x,\ \cos \pi x,\ \sin 2\pi x,\ \cos 2\pi x,\ \ldots,\ \sin 2^{L-1}\pi x,\ \cos 2^{L-1}\pi x\right)$$



Positional encoding: sines that double

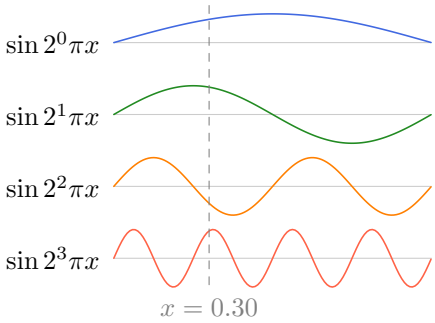
$$\gamma(x) = \left(x,\ \sin \pi x,\ \cos \pi x,\ \sin 2\pi x,\ \cos 2\pi x,\ \ldots,\ \sin 2^{L-1}\pi x,\ \cos 2^{L-1}\pi x\right)$$



k	$x = 0.30$	$x = 0.31$
0	0.809	0.827
1	0.951	0.930
2	-0.588	-0.685
3	0.951	0.998

Positional encoding: sines that double

$$\gamma(x) = \left(x,\ \sin \pi x,\ \cos \pi x,\ \sin 2\pi x,\ \cos 2\pi x,\ \ldots,\ \sin 2^{L-1}\pi x,\ \cos 2^{L-1}\pi x\right)$$

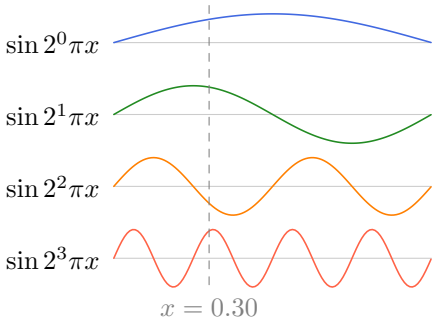


k	$x = 0.30$	$x = 0.31$
0	0.809	0.827
1	0.951	0.930
2	-0.588	-0.685
3	0.951	0.998

- Neighbours that look alike become different.

Positional encoding: sines that double

$$\gamma(x) = \big(x,\ \sin \pi x,\ \cos \pi x,\ \sin 2\pi x,\ \cos 2\pi x,\ \ldots,\ \sin 2^{L-1}\pi x,\ \cos 2^{L-1}\pi x\big)$$



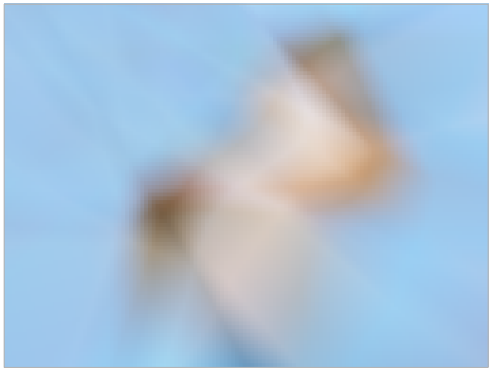
k	$x = 0.30$	$x = 0.31$
0	0.809	0.827
1	0.951	0.930
2	−0.588	−0.685
3	0.951	0.998

- Neighbours that look alike become different.

Our fit: $L = 7$, for x and for y: $2 + 28 = 30$ numbers in.

Rendered: the same budget, encoded

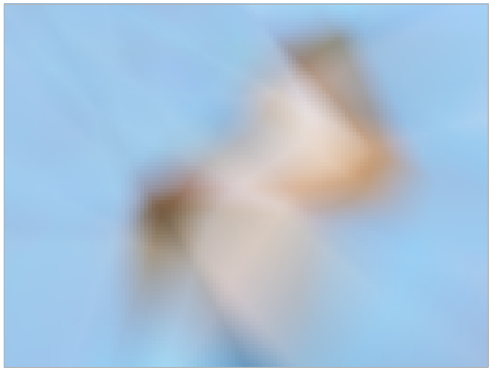
The same picture, the same steps, the same size of network



fed (x, y): 4,547 weights

Rendered: the same budget, encoded

The same picture, the same steps, the same size of network



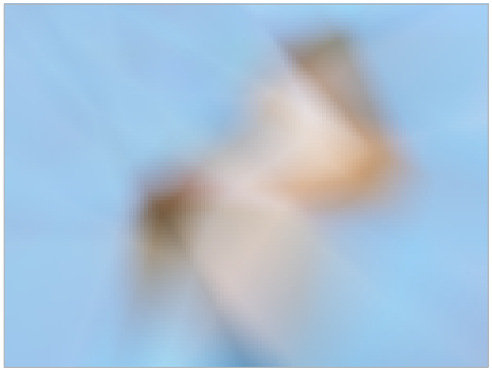
fed (x, y): 4,547 weights



fed $\gamma(x), \gamma(y)$: 4,527 weights

Rendered: the same budget, encoded

The same picture, the same steps, the same size of network



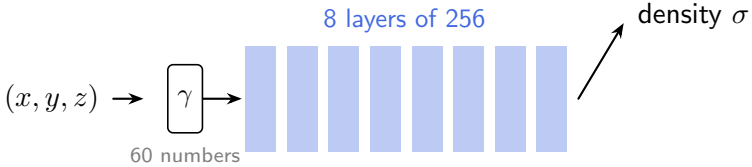
fed (x, y): 4,547 weights



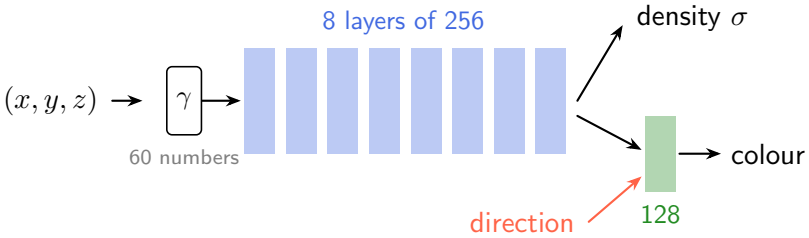
fed $\gamma(x), \gamma(y)$: 4,527 weights

Spots, eyes, legs: the encoding, not the weights, made the difference.

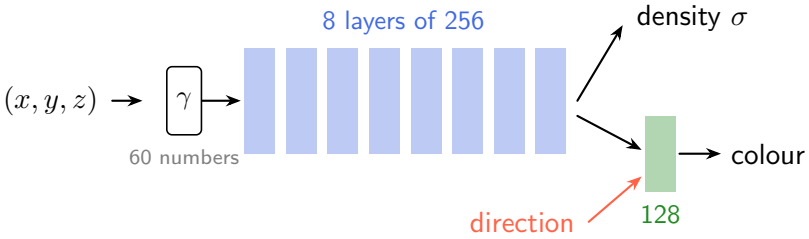
NeRF, the whole network



NeRF, the whole network

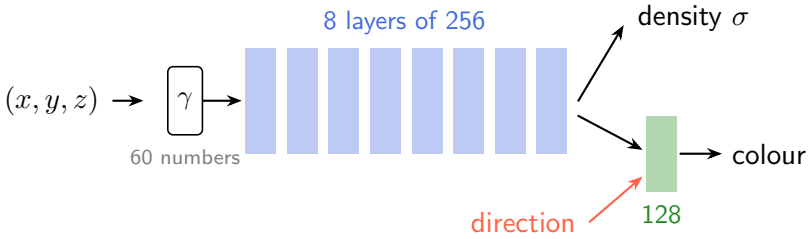


NeRF, the whole network



the density does not see the direction: a wall is where it is, from every side

NeRF, the whole network



the density does not see the direction: a wall is where it is, from every side

two passes along a ray: 64 even samples, then 128 where the weights were high

(Mildenhall et al. 2020; the sizes from the paper, from memory)

Worked: the price of one NeRF picture

One picture, 800×800 pixels, 192 samples a ray

Try first: how many times is the network run?

Worked: the price of one NeRF picture

One picture, 800×800 pixels, 192 samples a ray

Try first: how many times is the network run?

$640{,}000 \times 192 =$ **123 million** network runs

Worked: the price of one NeRF picture

One picture, 800×800 pixels, 192 samples a ray

Try first: how many times is the network run?

$640{,}000 \times 192 =$ **123 million** network runs

one run: about $8 \times 256 \times 256 \approx$ **0.5 million** multiply-adds

Worked: the price of one NeRF picture

One picture, 800×800 pixels, 192 samples a ray

Try first: how many times is the network run?

$640{,}000 \times 192 =$ **123 million** network runs

one run: about $8 \times 256 \times 256 \approx$ **0.5 million** multiply-adds

one picture: $\approx 7 \times 10^{13}$ multiply-adds

- All for one picture.

Worked: the price of one NeRF picture

One picture, 800×800 pixels, 192 samples a ray

Try first: how many times is the network run?

$640{,}000 \times 192 =$ **123 million** network runs

one run: about $8 \times 256 \times 256 \approx$ **0.5 million** multiply-adds

one picture: $\approx 7 \times 10^{13}$ multiply-adds

the paper: about half a minute a picture, a day or two to fit

(on one large graphics card, 2020; from memory)

- All for one picture.
- Fitting renders millions of rays, many times over.

Quick check 1: the sum and the network

1. A slice has $\sigma = 0.4$ and $\delta = 0.5$. What is its α?

2. Two slices in front of a sample each have $\alpha = 0.18$. What is the sample's T?

3. A sample has weight $w = 0$. How much does gradient descent change its colour?

4. Positional encoding with $L = 10$: how much faster does the last sine wiggle than the first?

Quick check 1: the sum and the network

1. A slice has $\sigma = 0.4$ and $\delta = 0.5$. What is its α?

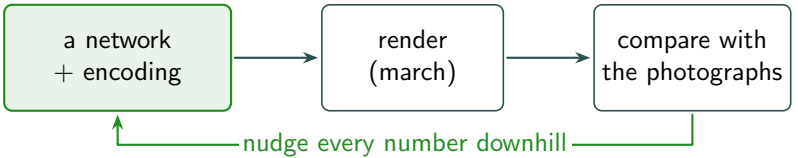
2. Two slices in front of a sample each have $\alpha = 0.18$. What is the sample's T?

3. A sample has weight $w = 0$. How much does gradient descent change its colour?

4. Positional encoding with $L = 10$: how much faster does the last sine wiggle than the first?

1: $1 - e^{-0.2} = 0.181$. 2: $0.82 \times 0.82 = 0.67$. 3: not at all (its slope is w). 4: $2^9 = 512$ times.

Recap: the loop

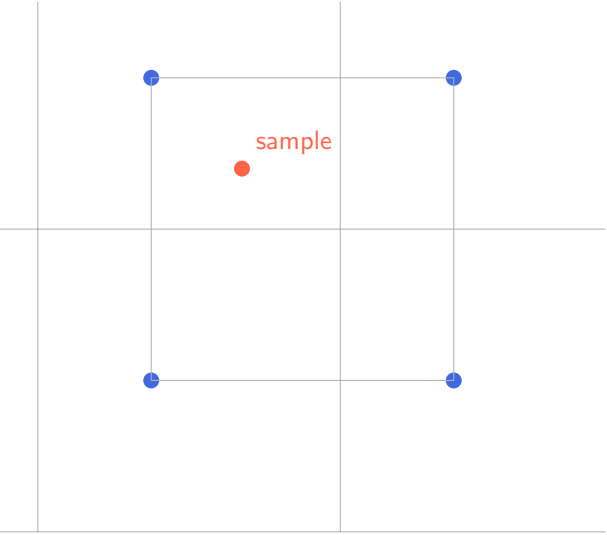


store	numbers in our fits	what one look-up costs
1 a grid (σ, colour)	9,216 (flatland)	4 reads
2 a network, encoded	4,527 (the picture)	every weight: 4,500 multiplies

Making it fast

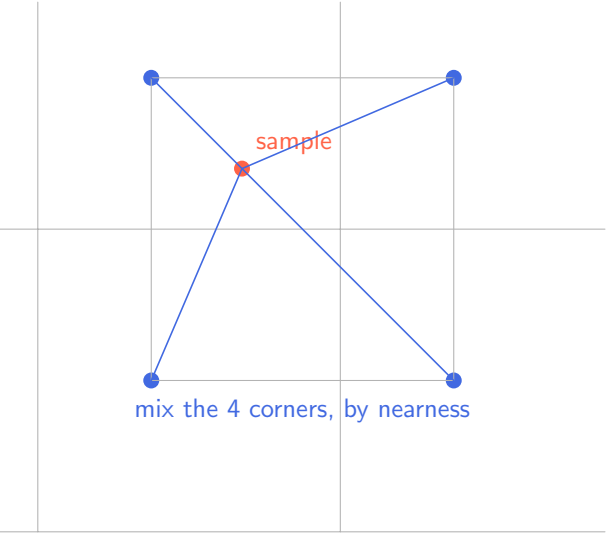
Store more, compute less

Store, don't compute: a grid is four reads



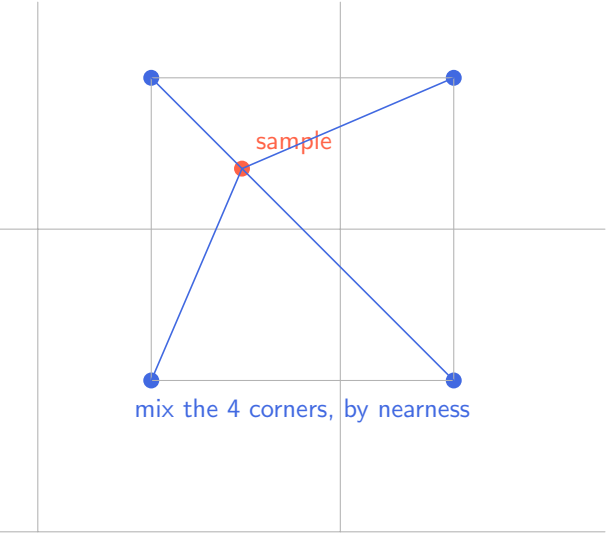
- Our flatland grid: no network at all.

Store, don't compute: a grid is four reads



- Our flatland grid: no network at all.
- A look-up: 4 reads in 2D, 8 in 3D, and a mix.

Store, don't compute: a grid is four reads



- Our flatland grid: no network at all.
- A look-up: 4 reads in 2D, 8 in 3D, and a mix.
- Plenoxels (2022): NeRF's pictures, no network, a hundred times faster.

Worked: a fine 3D grid's memory

A grid fine enough for a room's detail: $512 \times 512 \times 512$

Try first: 4 numbers a point, 4 bytes a number.

Worked: a fine 3D grid's memory

A grid fine enough for a room's detail: $512 \times 512 \times 512$

Try first: 4 numbers a point, 4 bytes a number.

$512^3 = 134{,}217{,}728$ points

Worked: a fine 3D grid's memory

A grid fine enough for a room's detail: $512 \times 512 \times 512$

Try first: 4 numbers a point, 4 bytes a number.

$512^3 = 134{,}217{,}728$ points

$\times\ 4 \times 4$ bytes $=$ **2.15 GB**

- Fast to read, huge to keep.

Worked: a fine 3D grid's memory

A grid fine enough for a room's detail: $512 \times 512 \times 512$

Try first: 4 numbers a point, 4 bytes a number.

$512^3 = 134{,}217{,}728$ points

$\times\ 4 \times 4$ bytes $=$ **2.15 GB**

most of it air, or the inside of walls

- Fast to read, huge to keep.

Worked: a fine 3D grid's memory

A grid fine enough for a room's detail: $512 \times 512 \times 512$

Try first: 4 numbers a point, 4 bytes a number.

$512^3 = 134{,}217{,}728$ points

$\times\ 4 \times 4$ bytes $=$ **2.15 GB**

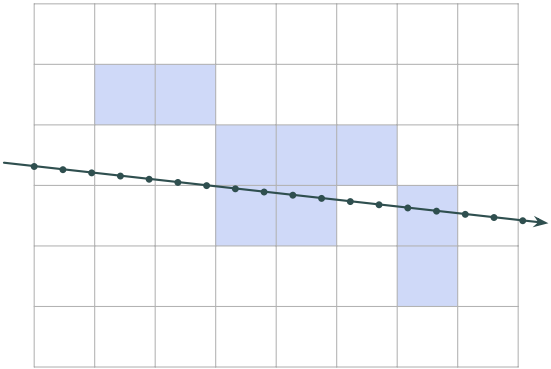
most of it air, or the inside of walls

Double the detail: 8 times the memory

NeRF's network: about half a million weights, 2 MB

- Fast to read, huge to keep.
- We want a grid's speed at a network's size.

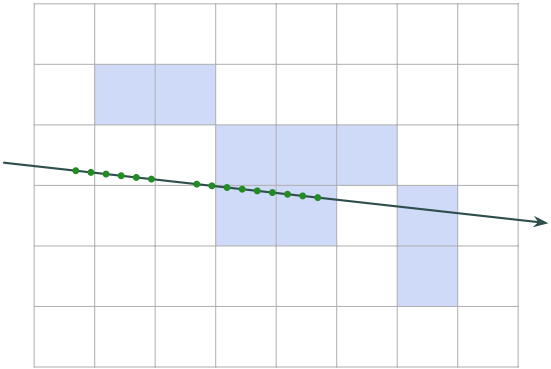
Skip the air



shaded: a cell with something in it

- Even samples: most land in air.

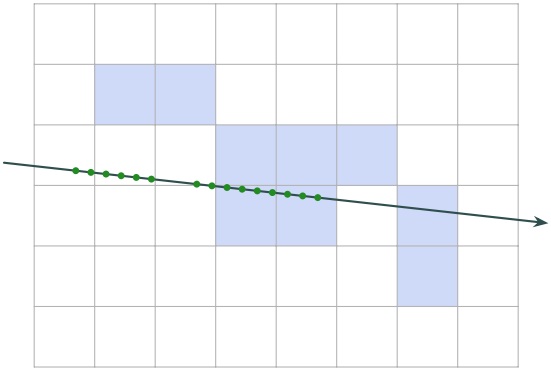
Skip the air



shaded: a cell with something in it

- Even samples: most land in air.
- A coarse map of where things are: sample only there.

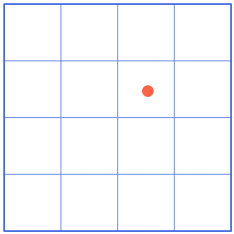
Skip the air



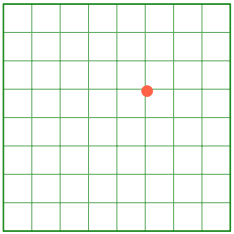
shaded: a cell with something in it

- Even samples: most land in air.
- A coarse map of where things are: sample only there.
- The map is updated from the density as the fit goes.

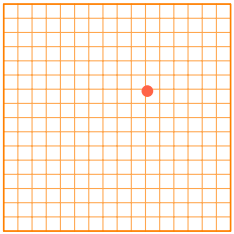
Grids from coarse to fine



4×4



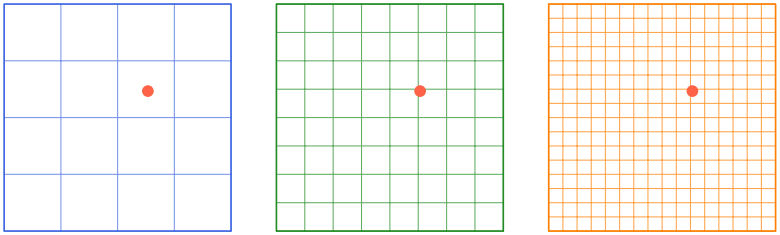
8×8



16×16

- The same point, read on several grids.

Grids from coarse to fine

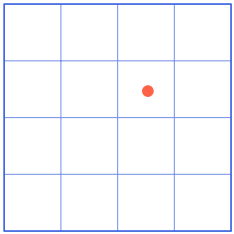


- The same point, read on several grids.
- Coarse levels: the broad shapes.

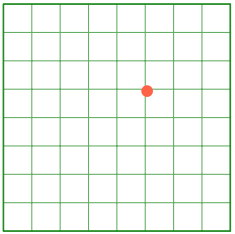
each level: a few numbers (features) at each corner

a sample: read every level, put the features side by side, a tiny network on top

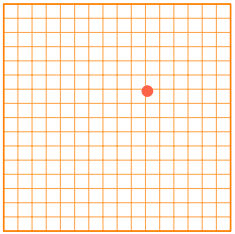
Grids from coarse to fine



4×4



8×8



16×16

- The same point, read on several grids.
- Coarse levels: the broad shapes.
- Fine levels: the edges, if there is room.

each level: a few numbers (features) at each corner

a sample: read every level, put the features side by side, a tiny network on top

The hash: fine grids share a small table

Our levels: $(N+1)^2$ corners, but only 256 slots each

$N = 4, 6, 10$: 25, 49, 121 corners: one slot each

The hash: fine grids share a small table

Our levels: $(N+1)^2$ corners, but only 256 slots each

$N = 4, 6, 10$: 25, 49, 121 corners: one slot each

$N = 17$ to 128: 324 to 16,641 corners into 256 slots

- Fine levels: many corners to a slot.

The hash: fine grids share a small table

Our levels: $(N+1)^2$ corners, but only 256 slots each

$N = 4, 6, 10$: 25, 49, 121 corners: one slot each

$N = 17$ to 128: 324 to 16,641 corners into 256 slots

slot $= \big(i \oplus (j \times 2654435761)\big) \bmod 256$

$\oplus$: exclusive or, bit by bit; the multiply in 32 bits

- Fine levels: many corners to a slot.

The hash: fine grids share a small table

Our levels: $(N+1)^2$ corners, but only 256 slots each

$N = 4, 6, 10$: 25, 49, 121 corners: one slot each

$N = 17$ to 128: 324 to 16,641 corners into 256 slots

slot $= \big(i \oplus (j \times 2654435761)\big) \bmod 256$

$\oplus$: exclusive or, bit by bit; the multiply in 32 bits

corner $(37, 52)$: $52 \times 177 \bmod 256 = 244$

- Fine levels: many corners to a slot.

The hash: fine grids share a small table

Our levels: $(N+1)^2$ corners, but only 256 slots each

$N = 4, 6, 10$: 25, 49, 121 corners: one slot each

$N = 17$ to 128: 324 to 16,641 corners into 256 slots

slot $= \big(i \ \oplus \ (j \times 2654435761)\big) \bmod 256$

$\oplus$: exclusive or, bit by bit; the multiply in 32 bits

corner $(37, 52)$: $52 \times 177 \bmod 256 = 244$

$244 \oplus 37 = 11110100 \oplus 00100101 = 11010001 = \mathbf{209}$

- Fine levels: many corners to a slot.

The hash: fine grids share a small table

Our levels: $(N+1)^2$ corners, but only 256 slots each

$N = 4, 6, 10$: 25, 49, 121 corners: one slot each

$N = 17$ to 128: 324 to 16,641 corners into 256 slots

slot $= \big(i \oplus (j \times 2654435761)\big) \bmod 256$

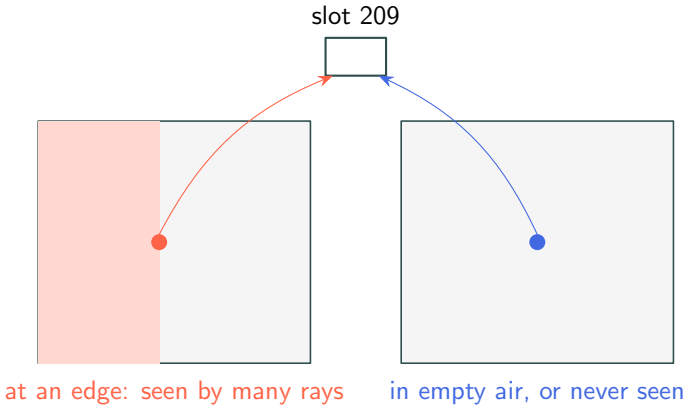
$\oplus$: exclusive or, bit by bit; the multiply in 32 bits

corner $(37, 52)$: $52 \times 177 \bmod 256 = 244$

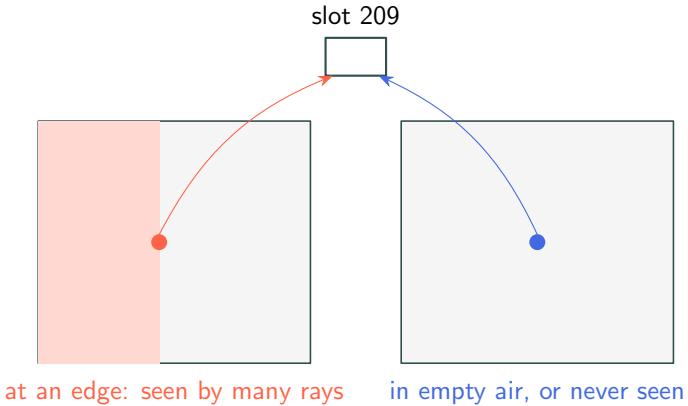
$244 \oplus 37 = 11110100 \oplus 00100101 = 11010001 = \mathbf{209}$

- Fine levels: many corners to a slot.
- No list of what is where: just arithmetic.

Collisions: let the error sort them out

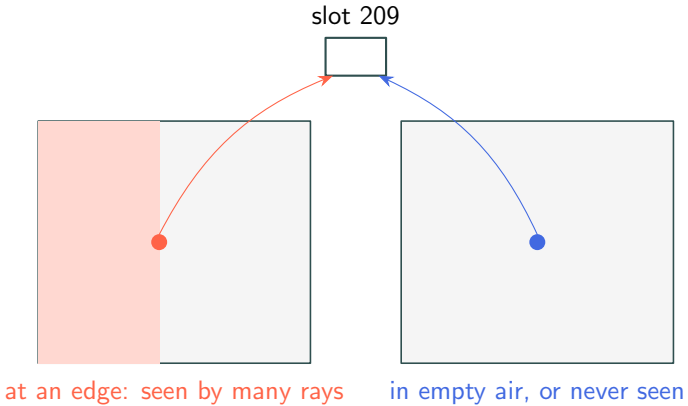


Collisions: let the error sort them out



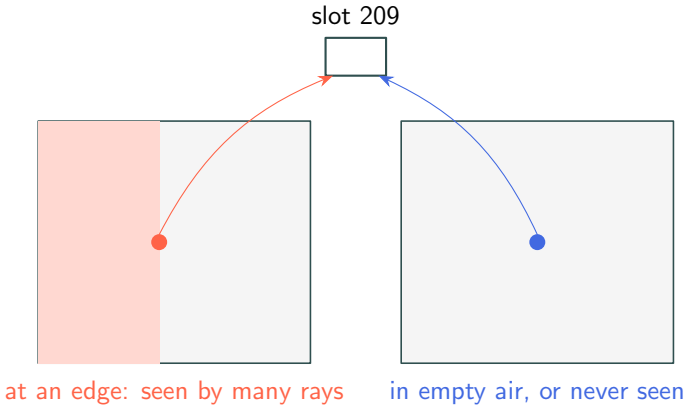
- The busy corner's slopes win.

Collisions: let the error sort them out



- The busy corner's slopes win.
- The other corner gets its values, and they hardly matter.

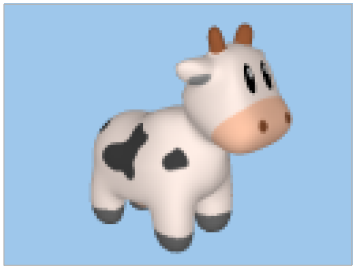
Collisions: let the error sort them out



- The busy corner's slopes win.
- The other corner gets its values, and they hardly matter.
- The coarse levels and the network tell them apart.

Rendered: a grid against a hash grid

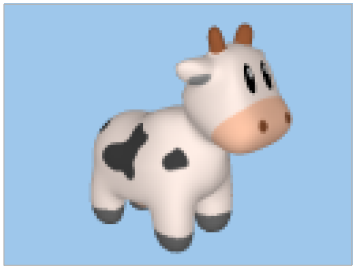
The same picture as C17, the same budget



the picture

Rendered: a grid against a hash grid

The same picture as C17, the same budget



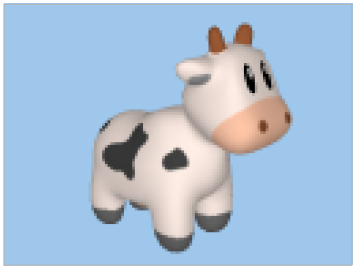
the picture



one grid, 46×34
4,692 numbers

Rendered: a grid against a hash grid

The same picture as C17, the same budget



the picture



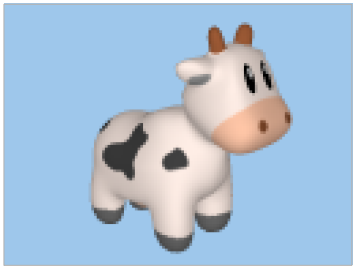
one grid, 46×34
4,692 numbers



8 hashed levels + a tiny network
4,739 numbers

Rendered: a grid against a hash grid

The same picture as C17, the same budget



the picture



one grid, 46×34
4,692 numbers



8 hashed levels + a tiny network
4,739 numbers

Same memory: the hash grid's finest level is as fine as the picture.

From a day to seconds

- NeRF: a day or two.



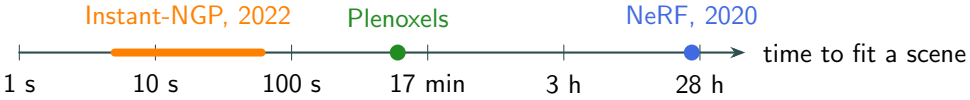
From a day to seconds

- NeRF: a day or two.



From a day to seconds

- NeRF: a day or two.
- Instant-NGP: seconds to a minute.



From a day to seconds

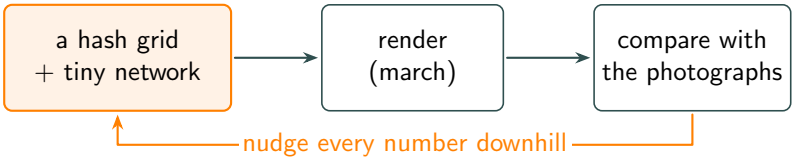
- NeRF: a day or two.
- Instant-NGP: seconds to a minute.



hash grid + tiny network + skipping air + code written for the card

(the papers' figures, from memory; each on one graphics card of its year)

Recap: the loop

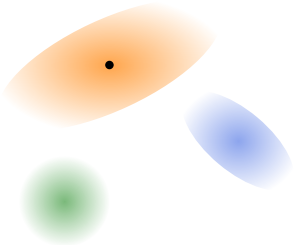


store	numbers in our fits	what one look-up costs
1 a grid (σ, colour)	9,216 (flatland)	4 reads
2 a network, encoded	4,527 (the picture)	every weight: 4,500 multiplies
3 a hash grid	4,739	32 reads + 600 multiplies

Gaussian splatting

No march: the scene as blobs, rasterised

A scene of blobs

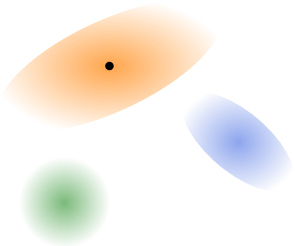


each: a soft ellipsoid

a Gaussian's numbers

- No grid, no network: a list.

A scene of blobs

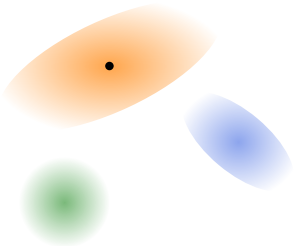


each: a soft ellipsoid

a Gaussian's numbers	
centre	3
size along 3 axes	3
turn (a quaternion)	4

- No grid, no network: a list.

A scene of blobs

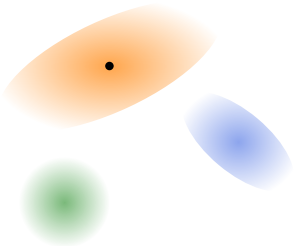


each: a soft ellipsoid

a Gaussian's numbers	
centre	3
size along 3 axes	3
turn (a quaternion)	4
opacity o	1

- No grid, no network: a list.
- Each blob fades from its centre outward.

A scene of blobs

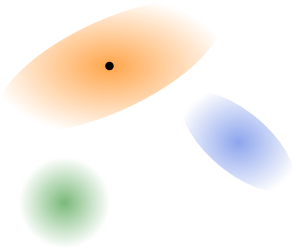


each: a soft ellipsoid

a Gaussian's numbers	
centre	3
size along 3 axes	3
turn (a quaternion)	4
opacity o	1
colour by direction	48

- No grid, no network: a list.
- Each blob fades from its centre outward.

A scene of blobs

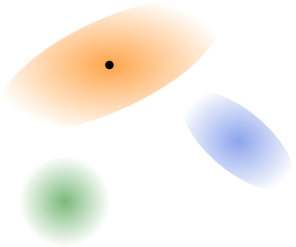


each: a soft ellipsoid

a Gaussian's numbers	
centre	3
size along 3 axes	3
turn (a quaternion)	4
opacity o	1
colour by direction	48
in all	**59**

- No grid, no network: a list.
- Each blob fades from its centre outward.

A scene of blobs



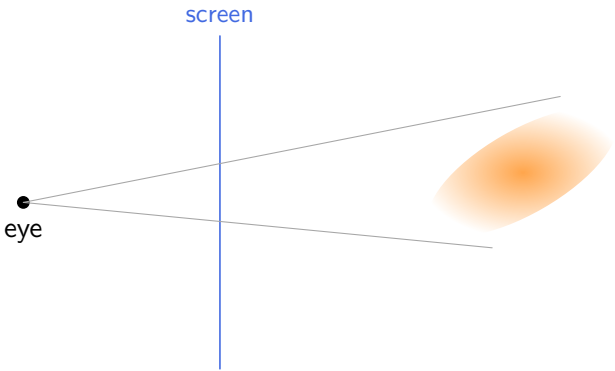
each: a soft ellipsoid

a Gaussian's numbers	
centre	3
size along 3 axes	3
turn (a quaternion)	4
opacity o	1
colour by direction	48
in all	**59**

- No grid, no network: a list.
- Each blob fades from its centre outward.

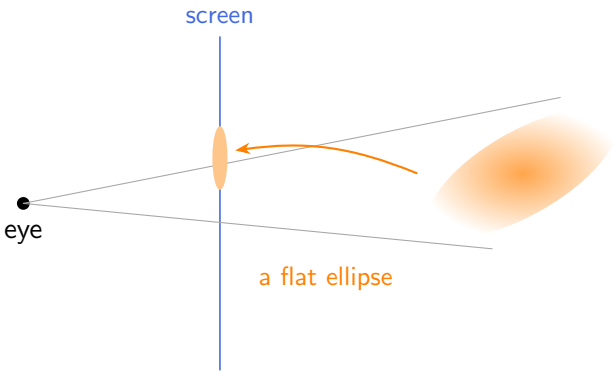
A scene: a few million of them, started from points the cameras' matching found.

A Gaussian on the screen is an ellipse



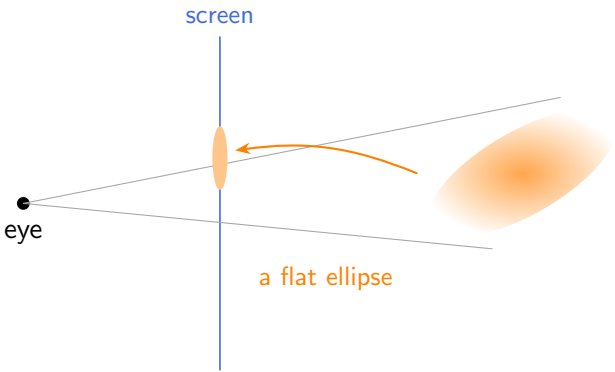
- A 3D blob, a camera.

A Gaussian on the screen is an ellipse



- A 3D blob, a camera.
- Its shadow on the screen: a 2D Gaussian.

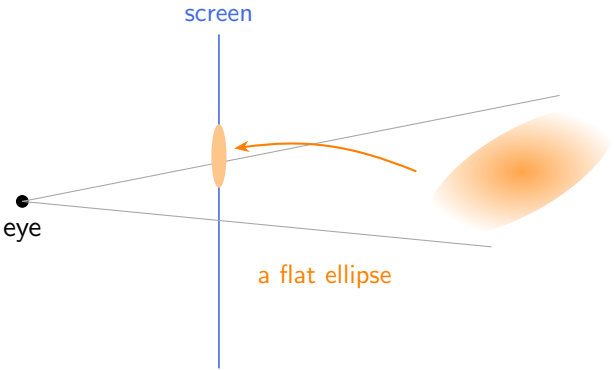
A Gaussian on the screen is an ellipse



- A 3D blob, a camera.
- Its shadow on the screen: a 2D Gaussian.

$\Sigma' = J\,W\,\Sigma\,W^{\mathsf{T}}\,J^{\mathsf{T}}$ the ellipsoid's shape, seen through the camera

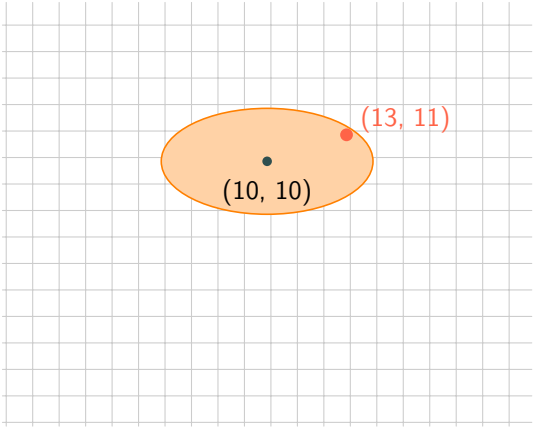
A Gaussian on the screen is an ellipse



- A 3D blob, a camera.
- Its shadow on the screen: a 2D Gaussian.
- Every blob: one ellipse, drawn like a triangle.

$\Sigma' = J\,W\,\Sigma\,W^{\mathsf{T}}\,J^{\mathsf{T}}$ the ellipsoid's shape, seen through the camera

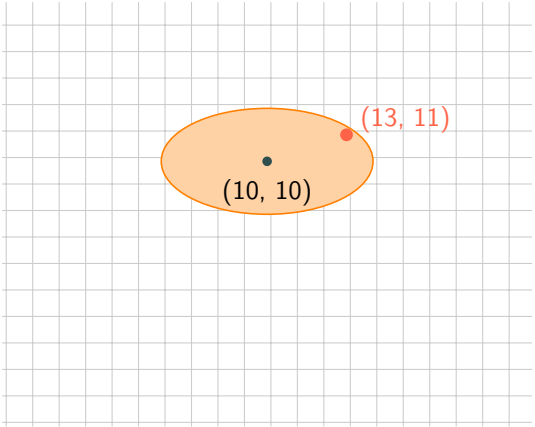
Worked: one splat at one pixel



spread: 4 across, 2 up; $o = 0.8$

Try first: α at the pixel.

Worked: one splat at one pixel

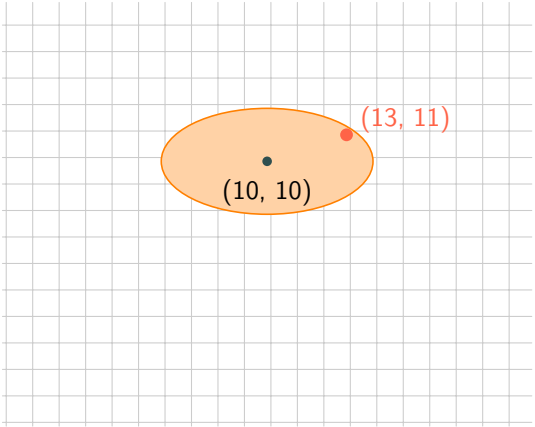


spread: 4 across, 2 up; $o = 0.8$

Try first: α at the pixel.

$d = (3, 1)$

Worked: one splat at one pixel



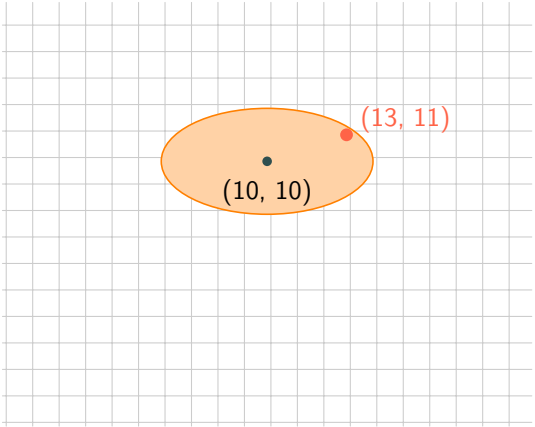
spread: 4 across, 2 up; $o = 0.8$

Try first: α at the pixel.

$d = (3, 1)$

$-\frac{1}{2}\left(\frac{3^2}{4^2} + \frac{1^2}{2^2}\right) = -0.406$

Worked: one splat at one pixel



spread: 4 across, 2 up; $o = 0.8$

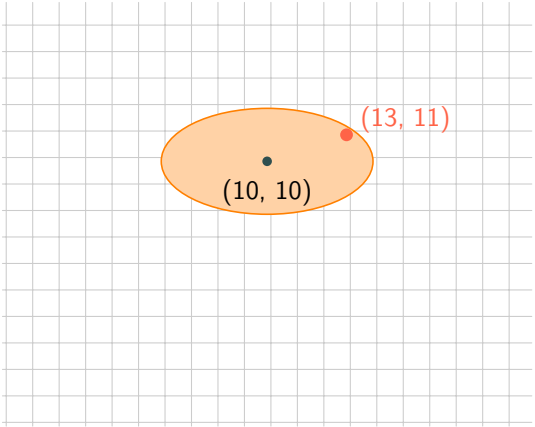
Try first: α at the pixel.

$d = (3, 1)$

$-\frac{1}{2}\left(\frac{3^2}{4^2} + \frac{1^2}{2^2}\right) = -0.406$

$e^{-0.406} = 0.666$

Worked: one splat at one pixel



spread: 4 across, 2 up; $o = 0.8$

Try first: α at the pixel.

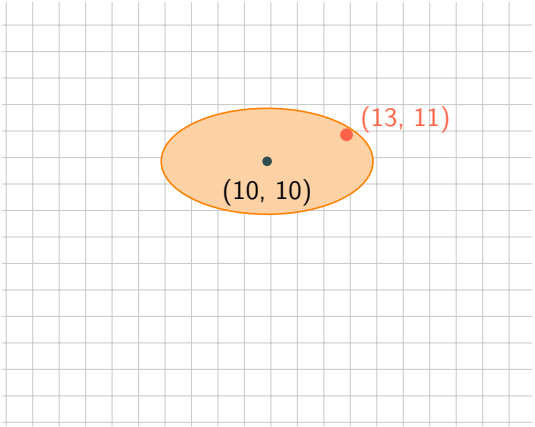
$d = (3, 1)$

$-\frac{1}{2}\left(\frac{3^2}{4^2} + \frac{1^2}{2^2}\right) = -0.406$

$e^{-0.406} = 0.666$

$\alpha = 0.8 \times 0.666 = \mathbf{0.533}$

Worked: one splat at one pixel



spread: 4 across, 2 up; $o = 0.8$

Try first: α at the pixel.

$d = (3, 1)$

$-\frac{1}{2}\left(\frac{3^2}{4^2} + \frac{1^2}{2^2}\right) = -0.406$

$e^{-0.406} = 0.666$

$\alpha = 0.8 \times 0.666 = \mathbf{0.533}$

below $1/255$: skipped;
above 0.99: held at 0.99

Sort, then part two's sum

Three splats over one pixel, nearest first; black behind

splat	α	c	T	$w = T\alpha$	wc
1	0.533	0.9			
2	0.40	0.2			
3	0.70	0.6			

- Sort the splats by depth.

Sort, then part two's sum

Three splats over one pixel, nearest first; black behind

splat	α	c	T	$w = T\alpha$	wc
1	0.533	0.9	1		
2	0.40	0.2	0.467		
3	0.70	0.6	0.280		

- Sort the splats by depth.

Sort, then part two's sum

Three splats over one pixel, nearest first; black behind

splat	α	c	T	$w = T\alpha$	wc
1	0.533	0.9	1	0.533	0.480
2	0.40	0.2	0.467	0.187	0.037
3	0.70	0.6	0.280	0.196	0.118

- Sort the splats by depth.
- Front to back: T shrinks as before.

Sort, then part two's sum

Three splats over one pixel, nearest first; black behind

splat	α	c	T	$w = T\alpha$	wc
1	0.533	0.9	1	0.533	0.480
2	0.40	0.2	0.467	0.187	0.037
3	0.70	0.6	0.280	0.196	0.118

$C = 0.480 + 0.037 + 0.118 = \mathbf{0.635}$; left over 0.084, black

- Sort the splats by depth.
- Front to back: T shrinks as before.

Sort, then part two's sum

Three splats over one pixel, nearest first; black behind

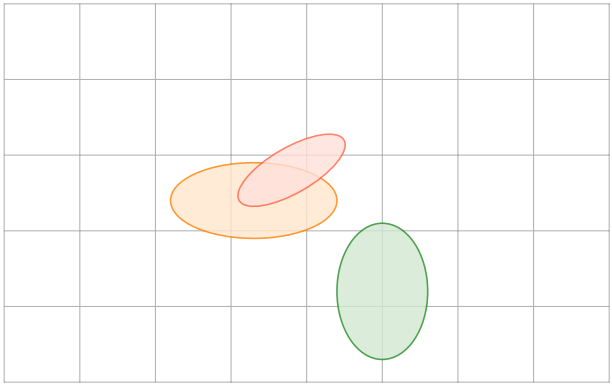
splat	α	c	T	$w = T\alpha$	wc
1	0.533	0.9	1	0.533	0.480
2	0.40	0.2	0.467	0.187	0.037
3	0.70	0.6	0.280	0.196	0.118

$C = 0.480 + 0.037 + 0.118 = \mathbf{0.635}$; left over 0.084, black

Exactly C11's sum: so C13's slopes, and the same loop.

- Sort the splats by depth.
- Front to back: T shrinks as before.
- Only one sort a picture, not one a pixel.

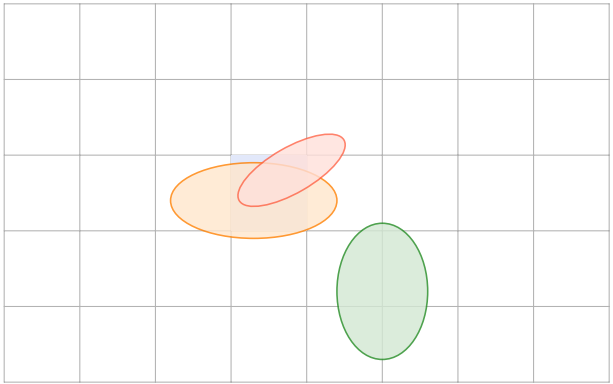
Tiles: every pixel gets a short list



the screen in tiles of 16×16 pixels

- Cut the screen into tiles.

Tiles: every pixel gets a short list

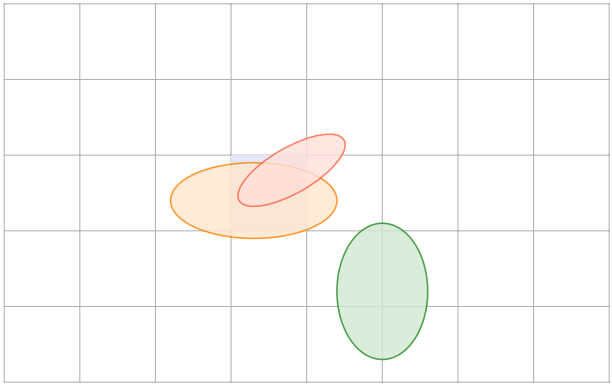


the screen in tiles of 16×16 pixels

- Cut the screen into tiles.

this tile's list: orange, red

Tiles: every pixel gets a short list



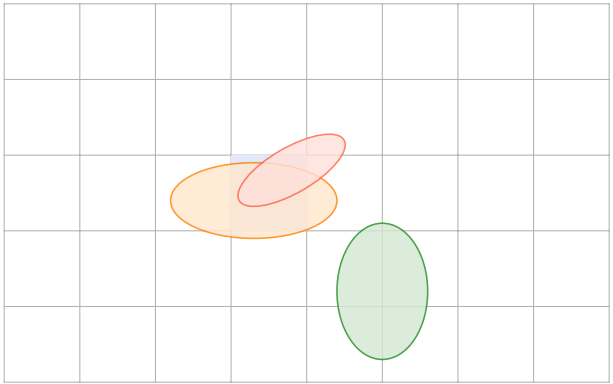
the screen in tiles of 16×16 pixels

- Cut the screen into tiles.

this tile's list: orange, red

sorted by depth, once

Tiles: every pixel gets a short list



the screen in tiles of 16×16 pixels

- Cut the screen into tiles.

this tile's list: orange, red

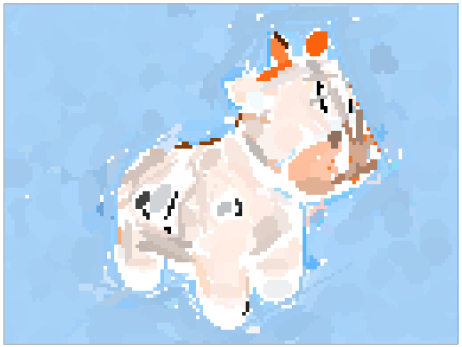
sorted by depth, once

its 16×16 pixels walk it together

- Our picture: about 15 splats a pixel.

Rendered: the blobs, and what they make

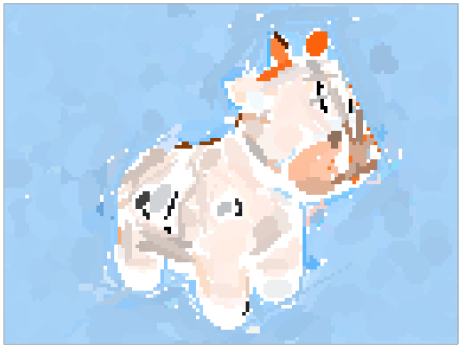
2D splats fitted to the picture by the loop



512 splats, each drawn solid
to one standard deviation

Rendered: the blobs, and what they make

2D splats fitted to the picture by the loop



512 splats, each drawn solid
to one standard deviation



the same 512, composited
4,608 numbers

Rendered: the blobs, and what they make

2D splats fitted to the picture by the loop



512 splats, each drawn solid
to one standard deviation



the same 512, composited
4,608 numbers

Big splats on the sky, thin ones on the eyes and outline: the fit chose their shapes.

Rendered: one picture, 4,700 numbers, four ways

The same picture, about 4,700 numbers each, fitted by the same loop



network, encoded
4,527



one grid
4,692



hash grid
4,739



512 splats
4,608

Rendered: one picture, 4,700 numbers, four ways

The same picture, about 4,700 numbers each, fitted by the same loop



network, encoded
4,527



one grid
4,692



hash grid
4,739



512 splats
4,608

- Hash grid: the sharpest. Splats: smooth, a little soft.

Rendered: one picture, 4,700 numbers, four ways

The same picture, about 4,700 numbers each, fitted by the same loop



network, encoded
4,527



one grid
4,692



hash grid
4,739



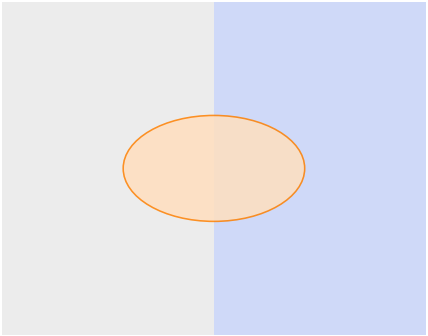
512 splats
4,608

Each store fails in its own way: grain, blur, faint mottling, softness.

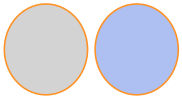
Splats win in 3D on speed: no march, and they grow where needed.

- Hash grid: the sharpest. Splats: smooth, a little soft.

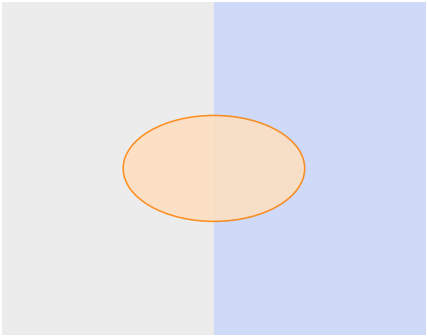
Grow and prune



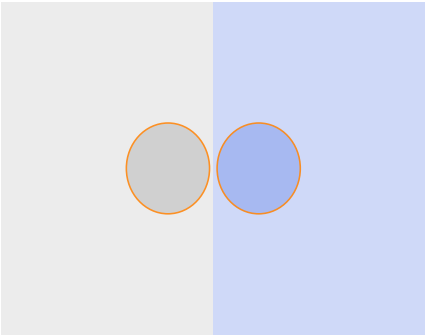
one splat across an edge:
its slopes keep pulling both ways



Grow and prune



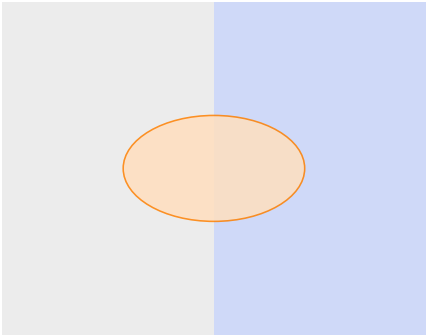
one splat across an edge:
its slopes keep pulling both ways



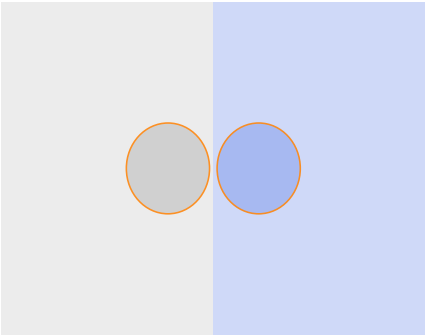
split in two:
each takes one side

Grow and prune

every few hundred steps: split the big ones that are pulled hard, copy the small ones;



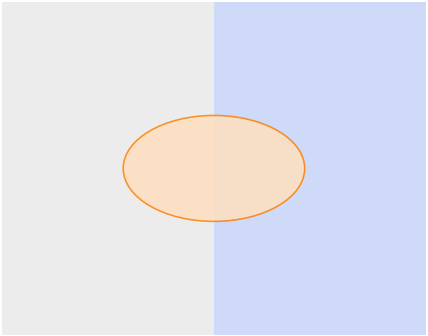
one splat across an edge:
its slopes keep pulling both ways



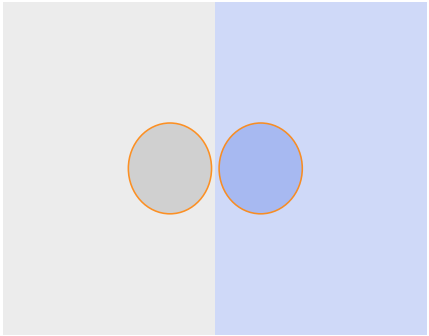
split in two:
each takes one side

Grow and prune

every few hundred steps: split the big ones that are pulled hard, copy the small ones;



one splat across an edge:
its slopes keep pulling both ways



split in two:
each takes one side

and
drop the
nearly clear

Rendered: 128, 512 and 2,048 splats

The picture, fitted with more and more splats



128 splats
1,152 numbers

Rendered: 128, 512 and 2,048 splats

The picture, fitted with more and more splats



128 splats
1,152 numbers



512
4,608

Rendered: 128, 512 and 2,048 splats

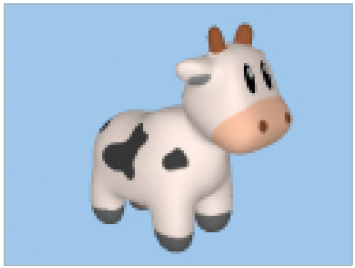
The picture, fitted with more and more splats



128 splats
1,152 numbers



512
4,608



2,048
18,432

More splats, finer detail: what growing them buys.

Quick check 2: splats

1. The two nearest splats over a pixel each have $\alpha = 0.5$. How much of the pixel is left for the third?

2. The third has $\alpha = 0.6$. What is its weight?

3. How many numbers do 2,048 of our 2D splats need?

4. Why is a big splat that is pulled hard split, and not just moved?

Quick check 2: splats

1. The two nearest splats over a pixel each have $\alpha = 0.5$. How much of the pixel is left for the third?

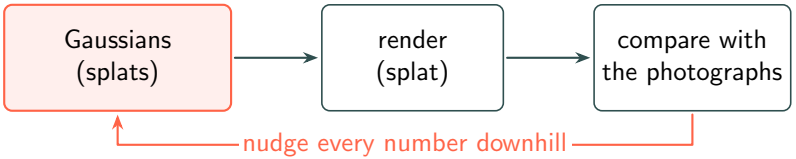
2. The third has $\alpha = 0.6$. What is its weight?

3. How many numbers do 2,048 of our 2D splats need?

4. Why is a big splat that is pulled hard split, and not just moved?

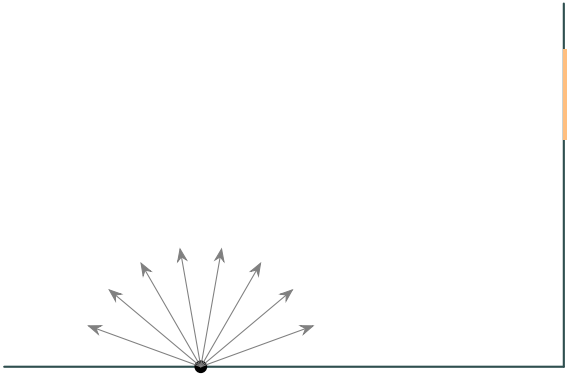
1: $0.5 \times 0.5 = 0.25$. 2: $0.25 \times 0.6 = 0.15$. 3: $2{,}048 \times 9 = 18{,}432$. 4: it sits across detail it cannot fit alone.

Recap: the loop



store	numbers in our fits	what one look-up costs
1 a grid (σ, colour)	9,216 (flatland)	4 reads
2 a network, encoded	4,527 (the picture)	every weight: 4,500 multiplies
3 a hash grid	4,739	32 reads + 600 multiplies
4 512 splats	4,608	no march: about 15 splats a pixel

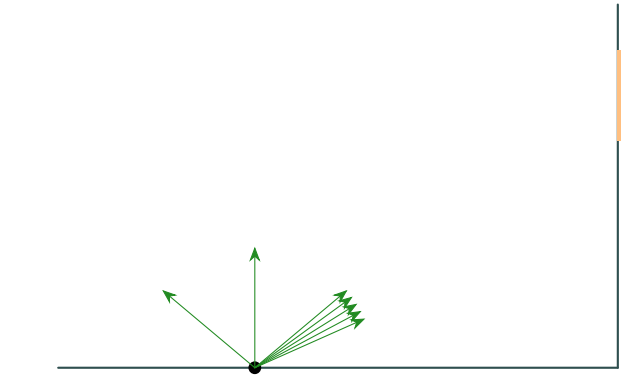
Learning inside a path tracer



lecture 20: any direction, by the cosine

- The same idea, fitting, used inside lecture 20's tracer.

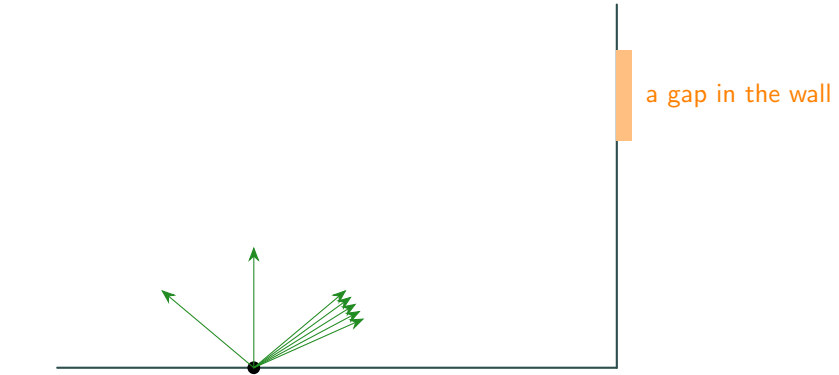
Learning inside a path tracer



guided: mostly toward where light was found

- The same idea, fitting, used inside lecture 20's tracer.
- Path guiding: learn, from the paths so far, where light comes from.

Learning inside a path tracer



guided: mostly toward where light was found

- The same idea, fitting, used inside lecture 20's tracer.
- Path guiding: learn, from the paths so far, where light comes from.
- Neural radiance caching: learn the light itself (Extra 8).

Where production renderers are

Films: path tracers, every one

Pixar's RenderMan, Disney's Hyperion, Wētā's Manuka, Arnold, Blender's Cycles

- Every frame of an animated film is path traced.

Where production renderers are

Films: path tracers, every one

Pixar's RenderMan, Disney's Hyperion, Wētā's Manuka, Arnold, Blender's Cycles

lecture 20, Extras 1 to 5: what they are built from

MIS, microfacets, spectra, volumes; few paths, then a learned denoiser

- Every frame of an animated film is path traced.

Where production renderers are

Films: path tracers, every one

Pixar's RenderMan, Disney's Hyperion, Wētā's Manuka, Arnold, Blender's Cycles

lecture 20, Extras 1 to 5: what they are built from

MIS, microfacets, spectra, volumes; few paths, then a learned denoiser

Games: Extra 6, one path per pixel and filters

- Every frame of an animated film is path traced.

Where production renderers are

Films: path tracers, every one

Pixar's RenderMan, Disney's Hyperion, Wētā's Manuka, Arnold, Blender's Cycles

lecture 20, Extras 1 to 5: what they are built from

MIS, microfacets, spectra, volumes; few paths, then a learned denoiser

Games: Extra 6, one path per pixel and filters

Captured places: today's fits

splats in phone apps and viewers, maps, visual effects plates

- Every frame of an animated film is path traced.
- The newest: scenes nobody modelled.

Summary

1 The inverse problem

$$\theta \leftarrow \theta - \eta \frac{dE}{d\theta}$$

guess, render, compare, nudge

2 NeRF

$$C = \sum_i T_i \, \alpha_i \, c_i$$

a network, fed sines at doubling frequencies

3 Making it fast

$$\text{slot} = (i \oplus j \, \pi_2) \bmod T$$

grids coarse to fine, hashed; skip the air

4 Gaussian splatting

$$\alpha = o \, e^{-\frac{1}{2} d^{\mathsf{T}} \Sigma'^{-1} d}$$

blobs sorted in tiles; split and prune

Reading

Mildenhall, Srinivasan, Tancik, Barron, Ramamoorthi and Ng, "NeRF: representing scenes as neural radiance fields for view synthesis", ECCV 2020

Tancik et al., "Fourier features let networks learn high frequency functions in low dimensional domains", NeurIPS 2020

Müller, Evans, Schied and Keller, "Instant neural graphics primitives with a multiresolution hash encoding", SIGGRAPH 2022

Kerbl, Kopanas, Leimkühler and Drettakis, "3D Gaussian splatting for real-time radiance field rendering", SIGGRAPH 2023

Müller, Rousselle, Novák and Keller, "Real-time neural radiance caching for path tracing", SIGGRAPH 2021

Fridovich-Keil, Yu et al., "Plenoxels: radiance fields without neural networks", CVPR 2022; **Müller, Gross and Novák**, "Practical path guiding", 2017

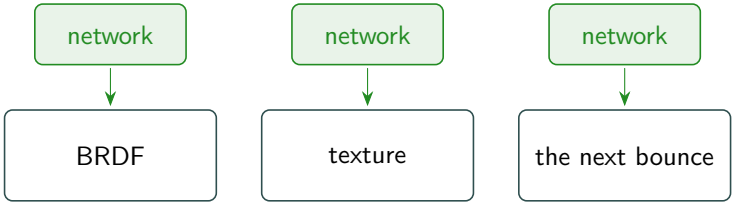
Next: Extra 8, neural appearance

- Today a network stored a whole scene.



pieces of lecture 20's tracer

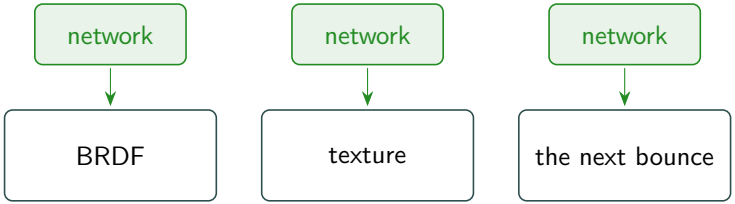
Next: Extra 8, neural appearance



pieces of lecture 20's tracer

- Today a network stored a whole scene.
- Extra 8: small networks as pieces of a renderer.

Next: Extra 8, neural appearance



pieces of lecture 20's tracer

- Today a network stored a whole scene.
- Extra 8: small networks as pieces of a renderer.
- Measured materials, compressed textures, the learned bounce.