Extra 8: Neural Appearance

Small Networks Inside a Renderer

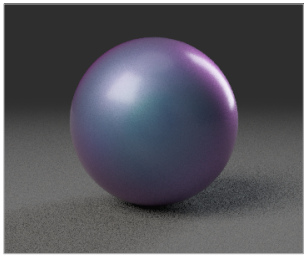
CS 453 — Computer Graphics

Fakhir Shaheen

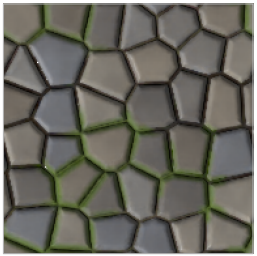
Forman Christian University

Today: small networks inside a renderer

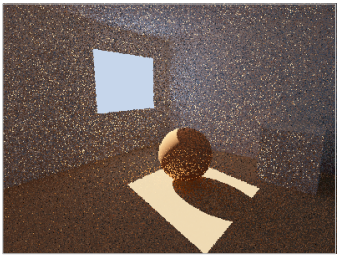
Extra 8: self-study, not on the exams



a paint, as a network



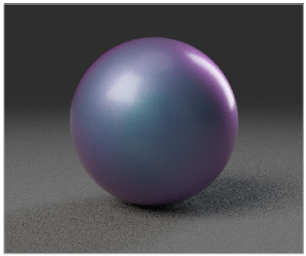
a texture, as a network



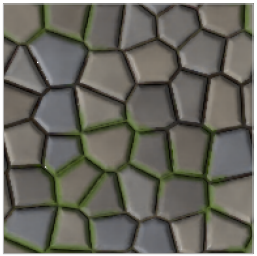
light, from a network

Today: small networks inside a renderer

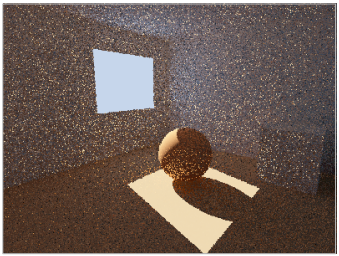
Extra 8: self-study, not on the exams



a paint, as a network



a texture, as a network

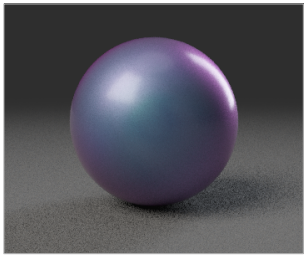


light, from a network

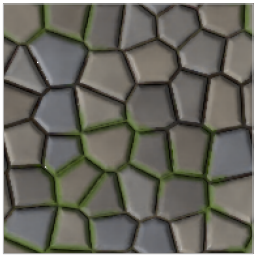
- The scene is ordinary: shapes, materials, a path tracer.

Today: small networks inside a renderer

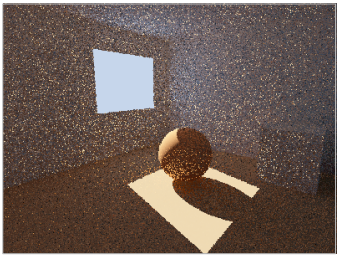
Extra 8: self-study, not on the exams



a paint, as a network



a texture, as a network

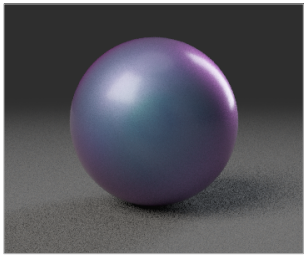


light, from a network

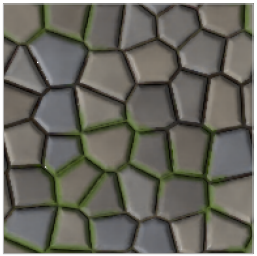
- The scene is ordinary: shapes, materials, a path tracer.
- Three of its pieces are handed to small networks.

Today: small networks inside a renderer

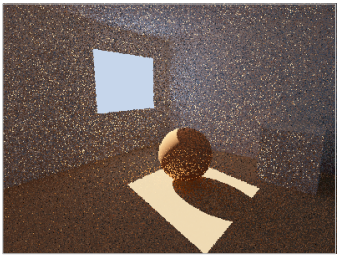
Extra 8: self-study, not on the exams



a paint, as a network



a texture, as a network



light, from a network

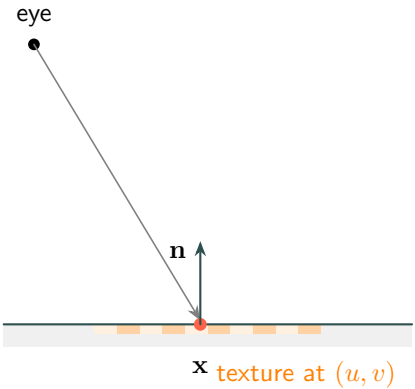
- The scene is ordinary: shapes, materials, a path tracer.
- Three of its pieces are handed to small networks.
- Each network is fitted to that piece's right answers.

A network the renderer calls

Three calls, three reasons

Three calls at every hit

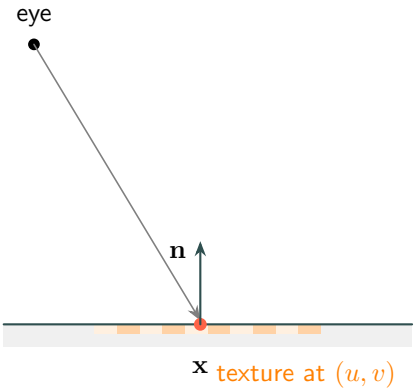
```
def trace(ray, depth):
    rec = scene.hit(ray)
    x = ray.evaluate(rec.t)
    m = material.texture(rec.uv)
    rho = m.evaluate(l, v, rec.normal)
        (for each lamp, and for d)
    d = a direction picked at x
    L = trace(Ray(x, d), depth - 1)
    return lamps' share + ...  * L
```



Grey: lecture 20's and Extras 1 and 2's tracer, unchanged.

Three calls at every hit

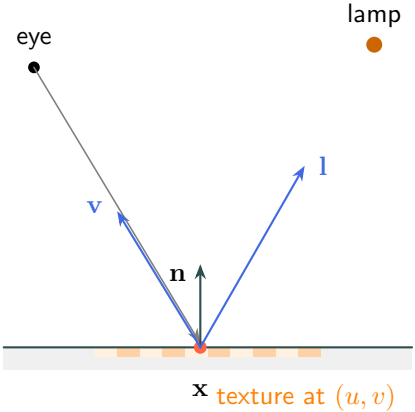
```
def trace(ray, depth):
    rec = scene.hit(ray)
    x = ray.evaluate(rec.t)
    m = material.texture(rec.uv)   3
    rho = m.evaluate(l, v, rec.normal)
        (for each lamp, and for d)
    d = a direction picked at x
    L = trace(Ray(x, d), depth - 1)
    return lamps' share + ...  * L
```



Grey: lecture 20's and Extras 1 and 2's tracer, unchanged.

Three calls at every hit

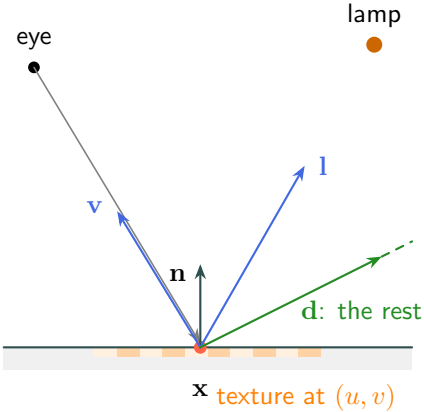
```
def trace(ray, depth):
    rec = scene.hit(ray)
    x = ray.evaluate(rec.t)
    m = material.texture(rec.uv)        3
    rho = m.evaluate(l, v, rec.normal)  2
        (for each lamp, and for d)
    d = a direction picked at x
    L = trace(Ray(x, d), depth - 1)
    return lamps' share + ...  * L
```



Grey: lecture 20's and Extras 1 and 2's tracer, unchanged.

Three calls at every hit

```
def trace(ray, depth):
    rec = scene.hit(ray)
    x = ray.evaluate(rec.t)
    m = material.texture(rec.uv)                3
    rho = m.evaluate(l, v, rec.normal)          2
        (for each lamp, and for d)
    d = a direction picked at x
    L = trace(Ray(x, d), depth - 1)             4
    return lamps' share + ...  * L
```

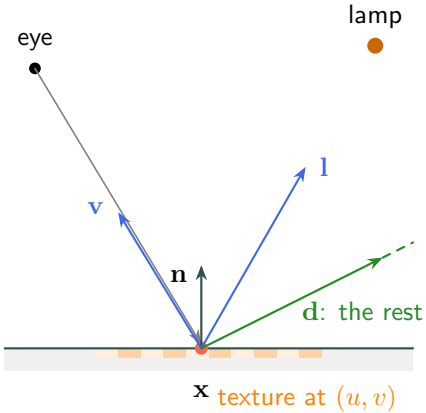


Grey: lecture 20's and Extras 1 and 2's tracer, unchanged.

Three calls at every hit

```
def trace(ray, depth):
    rec = scene.hit(ray)
    x = ray.evaluate(rec.t)
    m = material.texture(rec.uv)                3
    rho = m.evaluate(l, v, rec.normal)          2
        (for each lamp, and for d)
    d = a direction picked at x
    L = trace(Ray(x, d), depth - 1)             4
    return lamps' share + ...  * L
```

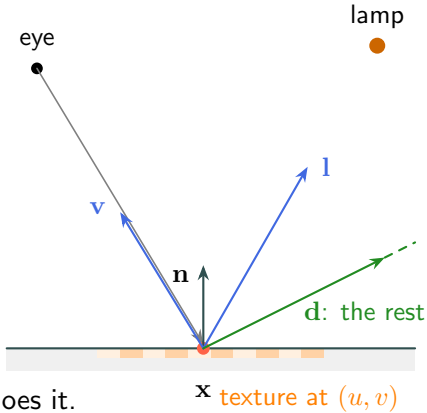
- Each call: a few numbers in, a few numbers out.



Grey: lecture 20's and Extras 1 and 2's tracer, unchanged.

Three calls at every hit

```
def trace(ray, depth):
    rec = scene.hit(ray)
    x = ray.evaluate(rec.t)
    m = material.texture(rec.uv)   3
    rho = m.evaluate(l, v, rec.normal)   2
        (for each lamp, and for d)
    d = a direction picked at x
    L = trace(Ray(x, d), depth - 1)   4
    return lamps' share + ...  * L
```

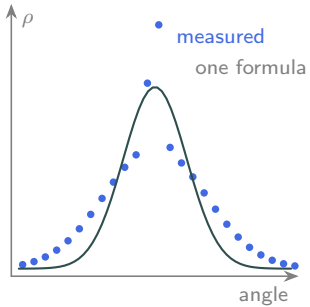


- Each call: a few numbers in, a few numbers out.
- Any of them can be a network. Tags: the part that does it.

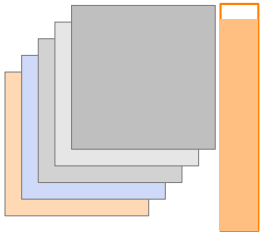
Grey: lecture 20's and Extras 1 and 2's tracer, unchanged.

Three reasons: too simple, too big, too slow

2 the BRDF

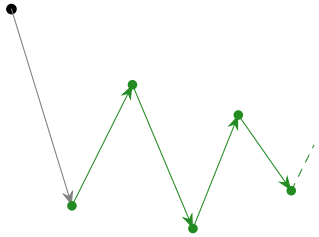


3 the textures



8 maps a material

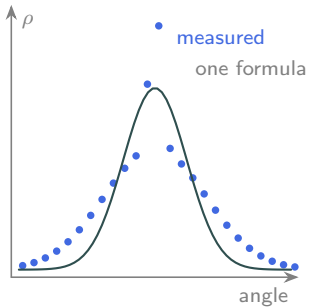
4 the rest of the path



a trace at every bounce

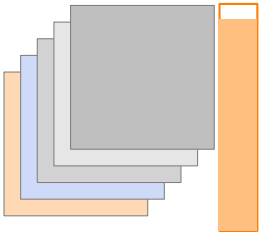
Three reasons: too simple, too big, too slow

2 the BRDF



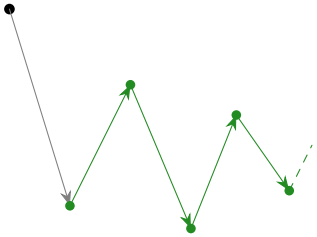
a formula is too simple:
no lobe fits a real paint

3 the textures



8 maps a material

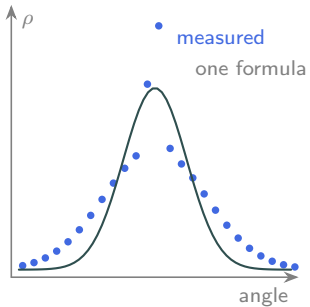
4 the rest of the path



a trace at every bounce

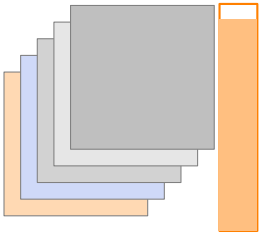
Three reasons: too simple, too big, too slow

2 the BRDF



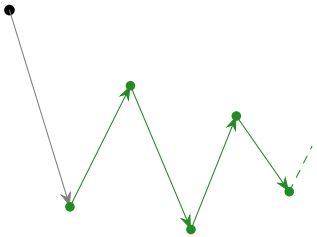
a formula is too simple:
no lobe fits a real paint

3 the textures



a table is too big:
gigabytes a scene

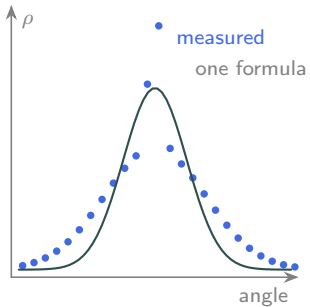
4 the rest of the path



a trace at every bounce

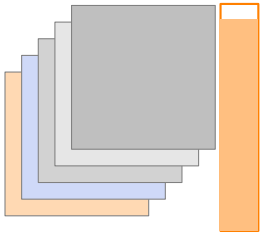
Three reasons: too simple, too big, too slow

2 the BRDF



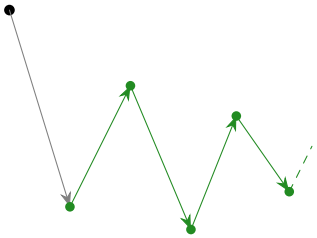
a formula is too simple:
no lobe fits a real paint

3 the textures



a table is too big:
gigabytes a scene

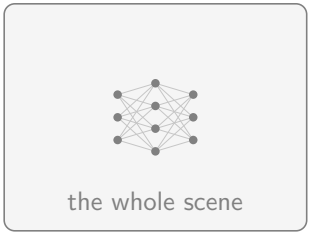
4 the rest of the path



a recursion is too slow:
the tail costs most

A piece, not the scene

Extra 7

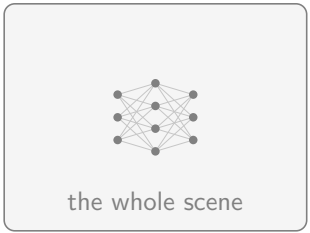


fitted to photographs;
asked along every ray

- Extra 7: the network was the scene.

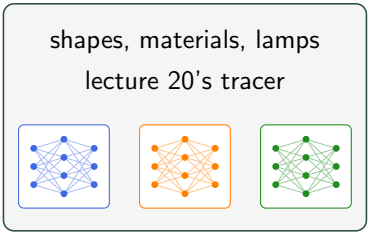
A piece, not the scene

Extra 7



fitted to photographs;
asked along every ray

Today

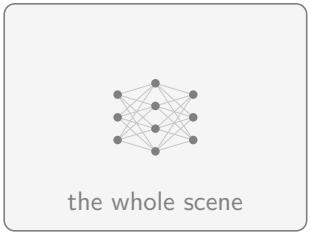


each fitted to its own piece;
called at every hit of every path

- Extra 7: the network was the scene.
- Today: an ordinary scene; pieces are networks.

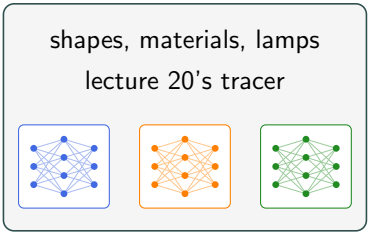
A piece, not the scene

Extra 7



fitted to photographs;
asked along every ray

Today



each fitted to its own piece;
called at every hit of every path

- Extra 7: the network was the scene.
- Today: an ordinary scene; pieces are networks.
- Called billions of times a second: it must be small.

Worked: what one call costs

A small network: 32 inputs, two layers of 64, 3 outputs

Try first: how many multiply-adds in one call?

Worked: what one call costs

A small network: 32 inputs, two layers of 64, 3 outputs

Try first: how many multiply-adds in one call?

$32 \times 64 + 64 \times 64 + 64 \times 3 = \mathbf{6{,}336}$

Worked: what one call costs

A small network: 32 inputs, two layers of 64, 3 outputs

Try first: how many multiply-adds in one call?

$32 \times 64 + 64 \times 64 + 64 \times 3 = \mathbf{6{,}336}$

one call a pixel, 1920×1080: **13.1 billion** a frame

Worked: what one call costs

A small network: 32 inputs, two layers of 64, 3 outputs

Try first: how many multiply-adds in one call?

$32 \times 64 + 64 \times 64 + 64 \times 3 = \mathbf{6{,}336}$

one call a pixel, 1920×1080: **13.1 billion** a frame

a graphics card: about 10^{14} a second $\Rightarrow$ under 1% of a frame
(its matrix units, in half precision; from memory)

- The arithmetic is cheap.

Worked: what one call costs

A small network: 32 inputs, two layers of 64, 3 outputs

Try first: how many multiply-adds in one call?

$32 \times 64 + 64 \times 64 + 64 \times 3 = \mathbf{6{,}336}$

one call a pixel, 1920×1080: **13.1 billion** a frame

a graphics card: about 10^{14} a second $\Rightarrow$ under 1% of a frame
(its matrix units, in half precision; from memory)

its weights: 6,467 numbers $\times$ 2 bytes $=$ **13 kB**

- The arithmetic is cheap.

Worked: what one call costs

A small network: 32 inputs, two layers of 64, 3 outputs

Try first: how many multiply-adds in one call?

$32 \times 64 + 64 \times 64 + 64 \times 3 =$ **6,336**

one call a pixel, 1920×1080: **13.1 billion** a frame

a graphics card: about 10^{14} a second $\Rightarrow$ under 1% of a frame
(its matrix units, in half precision; from memory)

its weights: 6,467 numbers $\times$ 2 bytes = **13 kB**

fetched for each pixel: $2{,}073{,}600 \times 13$ kB = **27 GB** a frame

- The arithmetic is cheap.
- Fetching the weights is not: keep them on the chip.

Where the right answers come from

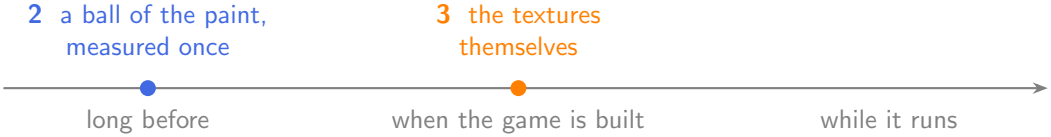


Where the right answers come from



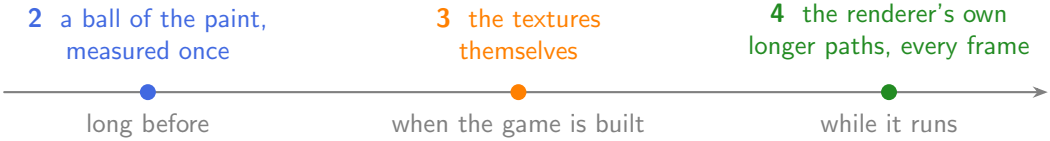
- Training data: what the piece should have answered.

Where the right answers come from



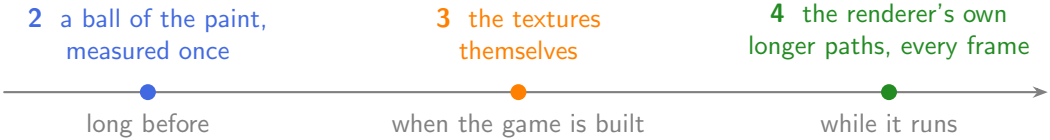
- Training data: what the piece should have answered.

Where the right answers come from



- Training data: what the piece should have answered.

Where the right answers come from



- Training data: what the piece should have answered.
- Part 4 learns during the very frames it serves.

Recap: three calls, three reasons

the BRDF $\rho(\mathbf{l}, \mathbf{v})$	the textures colour, normal, ... at (u, v)	the rest of the path light from there
a formula: too simple	a table: too big	a recursion: too slow

network for	trained on	when	ours

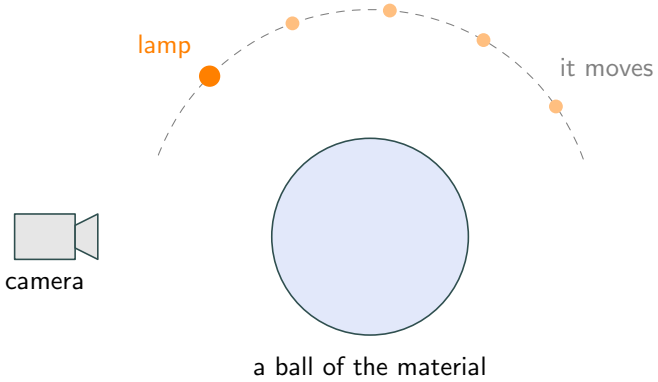
parts 2, 3 and 4 fill this in, one row each

Each call: numbers in, numbers out. Any of them can be a network.

A measured material

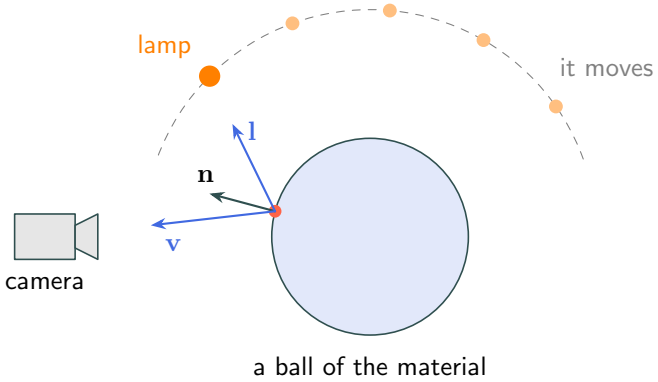
A table no formula fits

Measuring a material



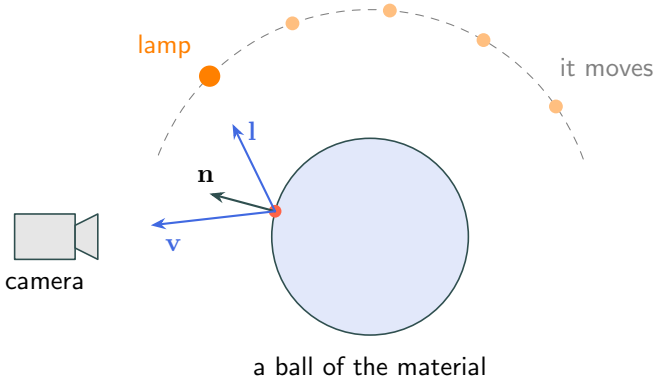
- A ball of the material, a camera, a lamp that moves.

Measuring a material



- A ball of the material, a camera, a lamp that moves.
- Each pixel: one normal, so one pair $(\mathbf{l}, \mathbf{v})$.

Measuring a material



- A ball of the material, a camera, a lamp that moves.
- Each pixel: one normal, so one pair $(\mathbf{l}, \mathbf{v})$.
- Hundreds of photographs fill a table of ρ.

MERL, 2003: 100 materials measured this way (paints, metals, plastics, fabrics).

Worked: a measured table's size

MERL's table: $90 \times 90 \times 180$ cells, one colour in each

(a cell for each triple of angles; next slide) Try first: how many numbers?

Worked: a measured table's size

MERL's table: $90 \times 90 \times 180$ cells, one colour in each

(a cell for each triple of angles; next slide) Try first: how many numbers?

$90 \times 90 \times 180 = 1{,}458{,}000$ cells; $\times 3$ colours $=$ **4,374,000** numbers

Worked: a measured table's size

MERL's table: $90 \times 90 \times 180$ cells, one colour in each

(a cell for each triple of angles; next slide) Try first: how many numbers?

$90 \times 90 \times 180 = 1{,}458{,}000$ cells; $\times 3$ colours = **4,374,000** numbers

$\times 8$ bytes each = **35 MB** for one material

- Exact, but 35 MB for one paint.

Worked: a measured table's size

MERL's table: $90 \times 90 \times 180$ cells, one colour in each

(a cell for each triple of angles; next slide) Try first: how many numbers?

$90 \times 90 \times 180 = 1{,}458{,}000$ cells; $\times 3$ colours = **4,374,000** numbers

$\times 8$ bytes each = **35 MB** for one material

a look-up: one read, and a blend of the neighbouring cells

- Exact, but 35 MB for one paint.

Worked: a measured table's size

MERL's table: $90 \times 90 \times 180$ cells, one colour in each

(a cell for each triple of angles; next slide) Try first: how many numbers?

$90 \times 90 \times 180 = 1{,}458{,}000$ cells; $\times 3$ colours = **4,374,000** numbers

$\times 8$ bytes each = **35 MB** for one material

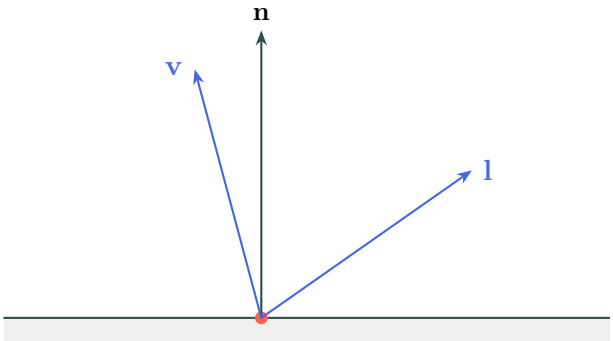
a look-up: one read, and a blend of the neighbouring cells

nothing to edit: no "roughness", no "colour"

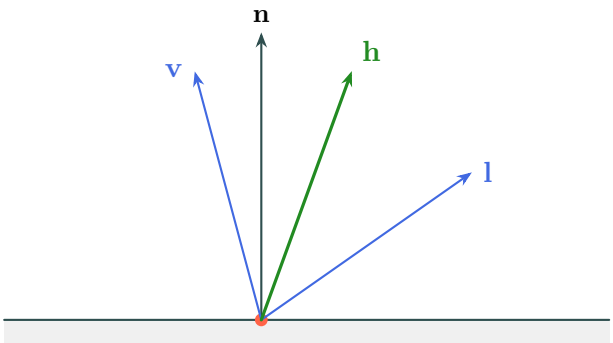
nothing between cells, nothing outside them

- Exact, but 35 MB for one paint.

Three angles for a pair of directions

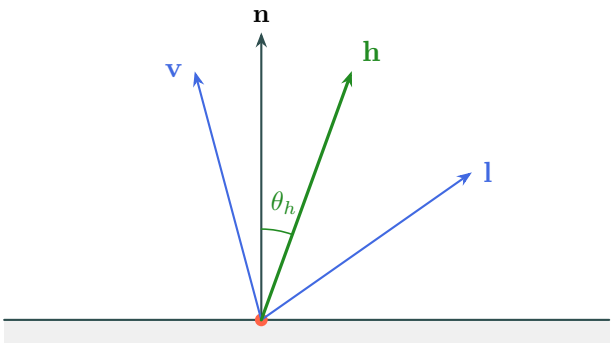


Three angles for a pair of directions



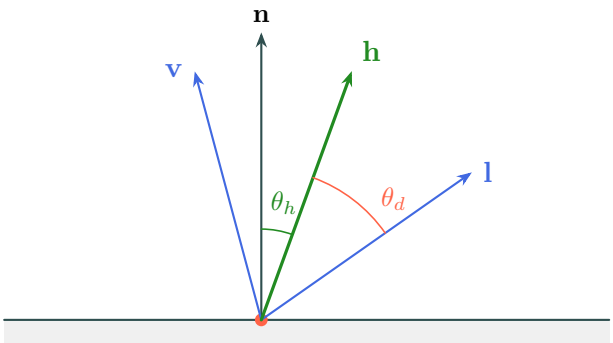
- $\mathbf{h}$: halfway between $\mathbf{l}$ and $\mathbf{v}$.

Three angles for a pair of directions



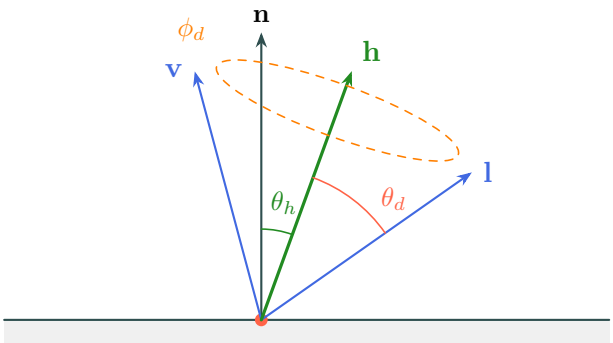
- $\mathbf{h}$: halfway between $\mathbf{l}$ and $\mathbf{v}$.
- θ_h: how far $\mathbf{h}$ leans from $\mathbf{n}$.

Three angles for a pair of directions



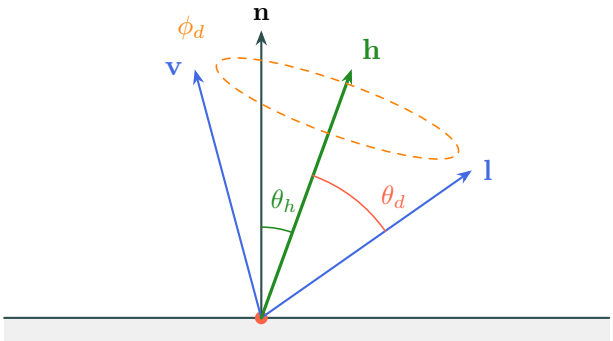
- $\mathbf{h}$: halfway between $\mathbf{l}$ and $\mathbf{v}$.
- θ_h: how far $\mathbf{h}$ leans from $\mathbf{n}$.
- θ_d: how far $\mathbf{l}$ is from $\mathbf{h}$.

Three angles for a pair of directions



- $\mathbf{h}$: halfway between $\mathbf{l}$ and $\mathbf{v}$.
- θ_h: how far $\mathbf{h}$ leans from $\mathbf{n}$.
- θ_d: how far $\mathbf{l}$ is from $\mathbf{h}$.
- ϕ_d: $\mathbf{l}$'s turn about $\mathbf{h}$.

Three angles for a pair of directions



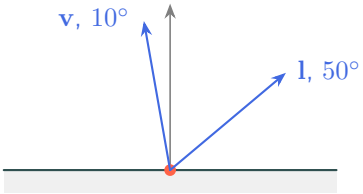
- $\mathbf{h}$: halfway between $\mathbf{l}$ and $\mathbf{v}$.
- θ_h: how far $\mathbf{h}$ leans from $\mathbf{n}$.
- θ_d: how far $\mathbf{l}$ is from $\mathbf{h}$.
- ϕ_d: $\mathbf{l}$'s turn about $\mathbf{h}$.

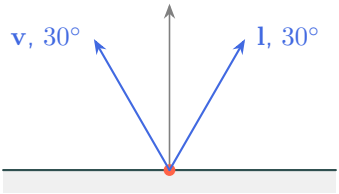
The highlight is where $\theta_h = 0$, for every θ_d: Rusinkiewicz, 1998.

Worked: two pairs, their angles

A **B**

Try first: θ_h and θ_d of each pair (all in one plane).

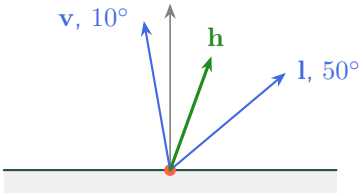


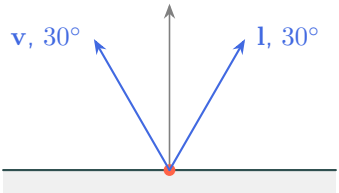


Worked: two pairs, their angles

A **B**

Try first: θ_h and θ_d of each pair (all in one plane).



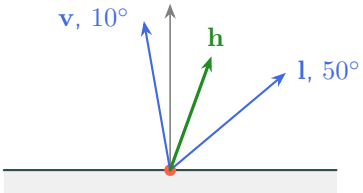


h: $(50 - 10)/2 = 20°$ right
$\theta_h = 20°$, $\theta_d = 50 - 20 = 30°$
$\phi_d = 0$ (**l** in the plane)

Worked: two pairs, their angles

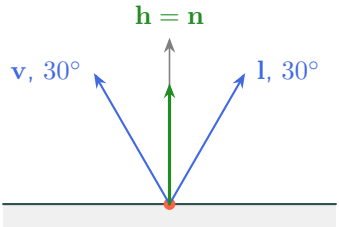
A

Try first: θ_h and θ_d of each pair (all in one plane).



h: $(50 - 10)/2 = 20°$ right
$\theta_h = 20°$, $\theta_d = 50 - 20 = 30°$
$\phi_d = 0$ (**l** in the plane)

B



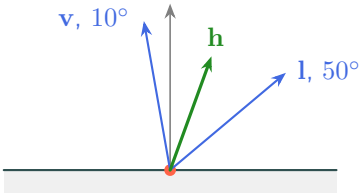
$\theta_h = 0$, $\theta_d = 30°$:
a mirror pair

- Every mirror pair has $\theta_h = 0$.

Worked: two pairs, their angles

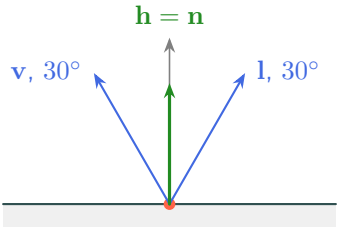
A

Try first: θ_h and θ_d of each pair (all in one plane).



h: $(50 - 10)/2 = 20°$ right
$\theta_h = 20°$, $\theta_d = 50 - 20 = 30°$
$\phi_d = 0$ (**l** in the plane)

B

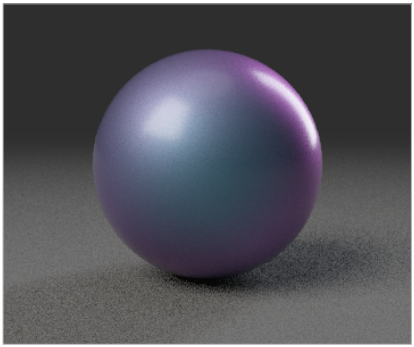


$\theta_h = 0$, $\theta_d = 30°$:
a mirror pair

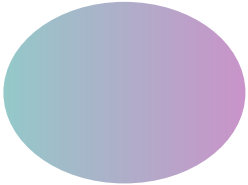
- Every mirror pair has $\theta_h = 0$.
- θ_h on a square-root scale, as MERL's cells are.

A's inputs for the network: $\sqrt{20/90} = 0.471$, $30/90 = 0.333$, $\cos 0 = 1$

Our stand-in: a paint we made up

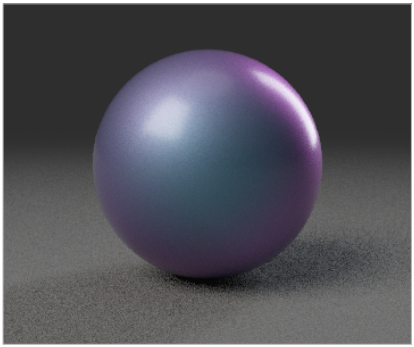


the made-up paint, rendered

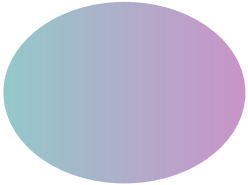


- We have no measured table here.

Our stand-in: a paint we made up

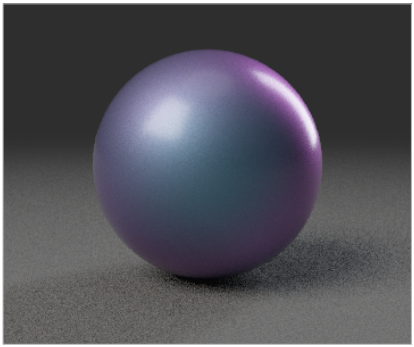


the made-up paint, rendered

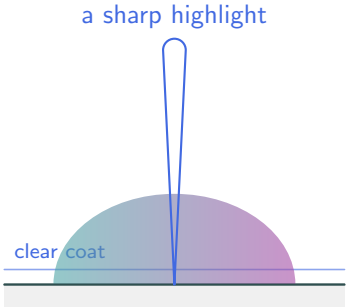


- We have no measured table here.
- A paint we made up stands in: we can ask it anything.

Our stand-in: a paint we made up



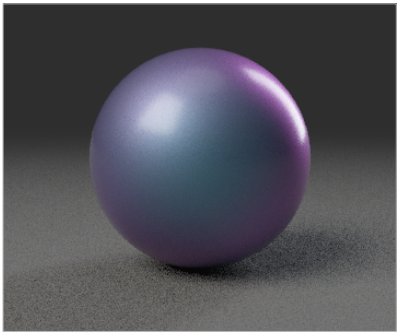
the made-up paint, rendered



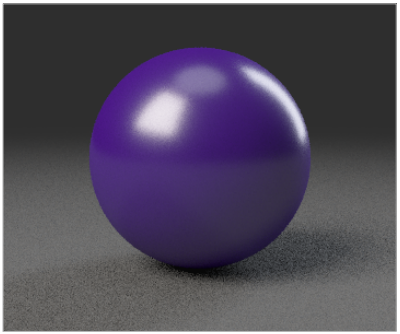
under it: a broad sheen,
teal head-on, violet at a slant

- We have no measured table here.
- A paint we made up stands in: we can ask it anything.
- A clear coat over a sheen that turns colour.

Rendered: the best single lobe



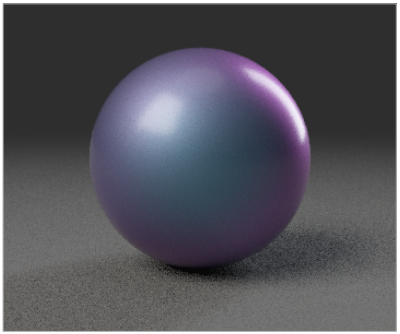
the paint



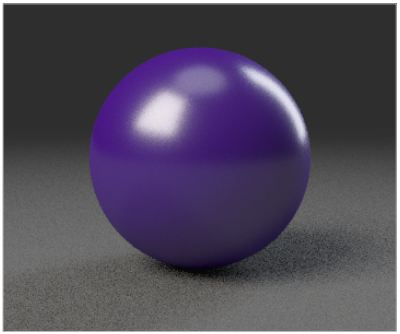
one GGX lobe, 7 numbers

- Extra 2's lobe and a diffuse colour, fitted by gradient descent.

Rendered: the best single lobe



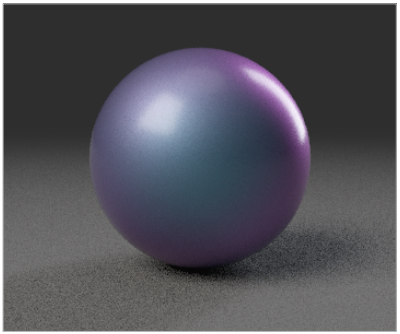
the paint



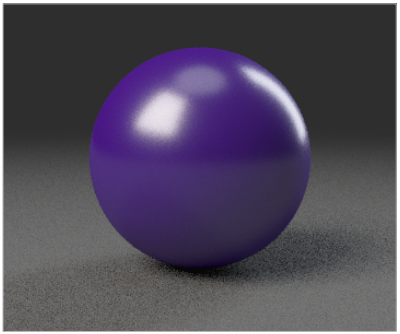
one GGX lobe, 7 numbers

- Extra 2's lobe and a diffuse colour, fitted by gradient descent.
- It takes the coat's sharpness, $\alpha = 0.07$.

Rendered: the best single lobe



the paint



one GGX lobe, 7 numbers

- Extra 2's lobe and a diffuse colour, fitted by gradient descent.
- It takes the coat's sharpness, $\alpha = 0.07$.
- One colour: the teal is gone.

The best formula of this kind is a violet plastic.

Worked: fit the logarithm

Three of our paint's values (green), each one 10% wrong:

pair	ρ	squared error	in $\log(1+\rho)$
lamp $70°$, eye $20°$	0.025		
mirror pair at $30°$	1.99		
mirror pair at $80°$	416		

Worked: fit the logarithm

Three of our paint's values (green), each one 10% wrong:

pair	ρ	squared error	in $\log(1+\rho)$
lamp $70°$, eye $20°$	0.025	0.000006	
mirror pair at $30°$	1.99	0.040	
mirror pair at $80°$	416	**1,735**	

- Plain: the grazing peak outweighs all.

Worked: fit the logarithm

Three of our paint's values (green), each one 10% wrong:

pair	ρ	squared error	in $\log(1+\rho)$
lamp $70°$, eye $20°$	0.025	0.000006	0.000006
mirror pair at $30°$	1.99	0.040	0.0041
mirror pair at $80°$	416	**1,735**	**0.0090**

- Plain: the grazing peak outweighs all.
- In logarithms: errors count as shares.

Worked: fit the logarithm

Three of our paint's values (green), each one 10% wrong:

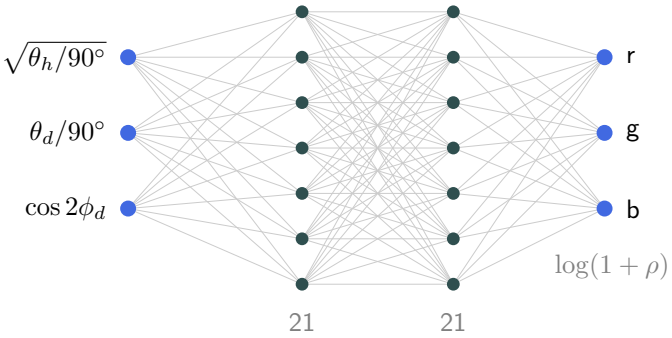
pair	ρ	squared error	in $\log(1+\rho)$
lamp $70°$, eye $20°$	0.025	0.000006	0.000006
mirror pair at $30°$	1.99	0.040	0.0041
mirror pair at $80°$	416	**1,735**	**0.0090**

fit $y \approx \log(1+\rho)$, then $\rho = e^y - 1$

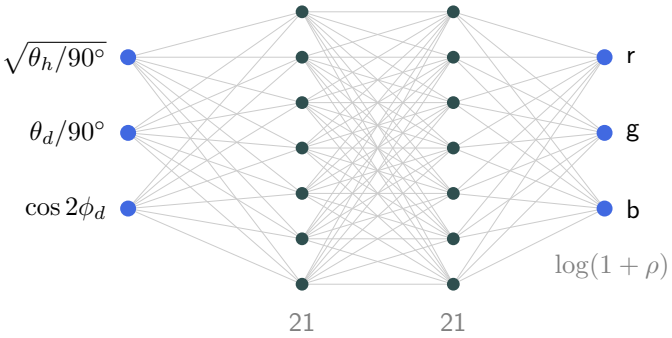
the paint's values run from 0.003 to over 400: five powers of ten

- Plain: the grazing peak outweighs all.
- In logarithms: errors count as shares.
- The network gives y; we use $\rho = e^y - 1$.

The network: three angles in, a colour out



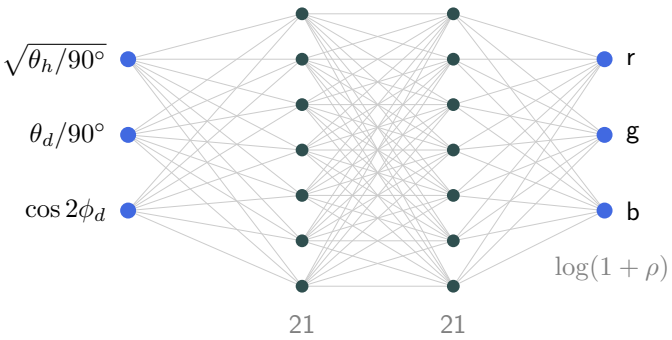
The network: three angles in, a colour out



- 612 numbers for a whole material.

$3 \times 21 + 21 + 21 \times 21 + 21 + 21 \times 3 + 3 =$ **612** numbers

The network: three angles in, a colour out

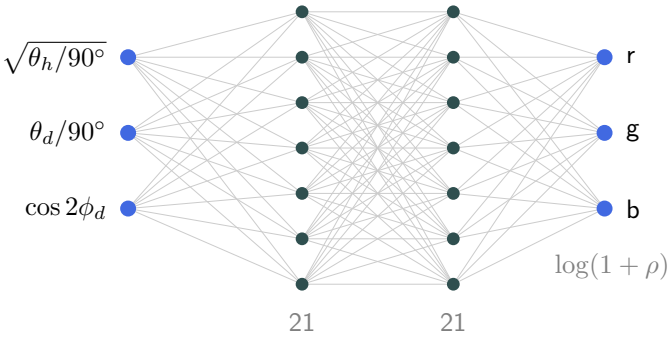


- 612 numbers for a whole material.

$3 \times 21 + 21 + 21 \times 21 + 21 + 21 \times 3 + 3 =$ **612** numbers

MERL's table: 4,374,000, about 7,000 times more

The network: three angles in, a colour out



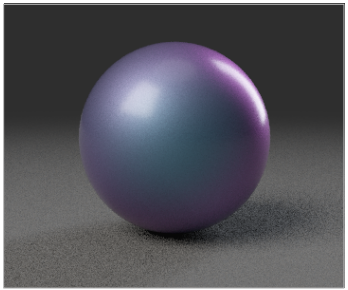
$3 \times 21 + 21 + 21 \times 21 + 21 + 21 \times 3 + 3 =$ **612** numbers

MERL's table: 4,374,000, about 7,000 times more

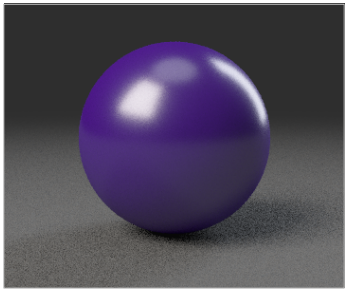
- 612 numbers for a whole material.
- Fitted to the 40,000 measurements by Adam.

Rendered: the paint, one lobe, the network

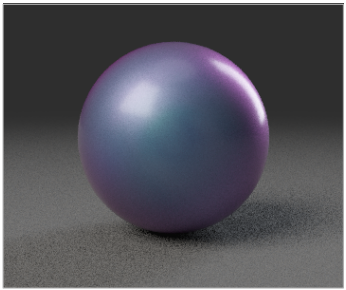
256 paths a pixel each; the same studio, the same floor



the paint



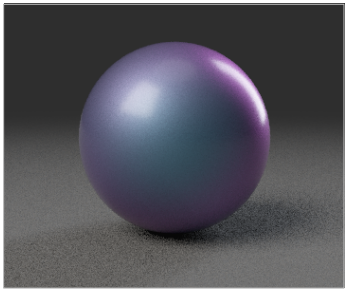
one lobe (7 numbers)



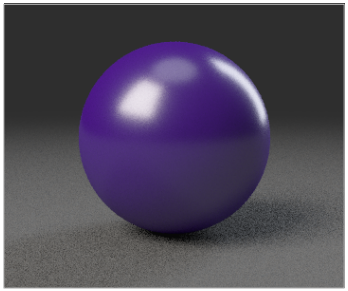
the network (612)

Rendered: the paint, one lobe, the network

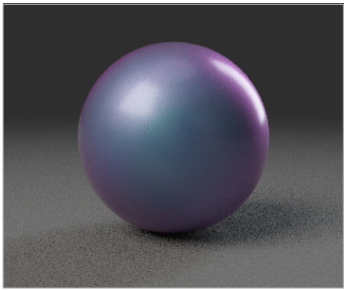
256 paths a pixel each; the same studio, the same floor



the paint



one lobe (7 numbers)

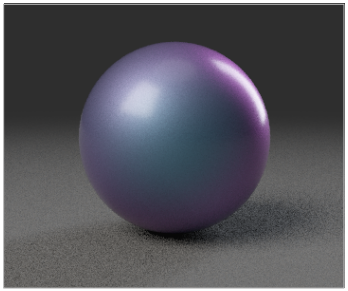


the network (612)

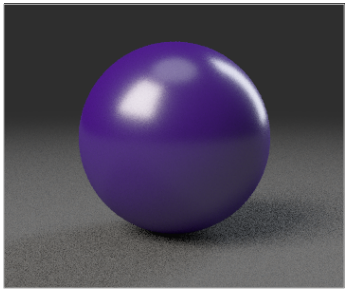
Error of $\log(1+\rho)$ at 5,000 new angles: one lobe **0.165**, the network **0.0005**

Rendered: the paint, one lobe, the network

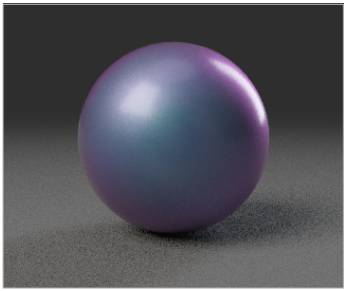
256 paths a pixel each; the same studio, the same floor



the paint



one lobe (7 numbers)



the network (612)

Error of $\log(1+\rho)$ at 5,000 new angles: one lobe **0.165**, the network **0.0005**

The network keeps the coat, the sheen and the turn of colour.

Worked: does the network keep energy?

Extra 2's furnace: the share of light sent back, $\int \rho \cos\theta \, d\omega$

(red, green, blue; each must stay below 1)

eye at	the paint		the network	one lobe
$0°$	0.123	0.342	0.453	
$45°$	0.269	0.226	0.447	
$75°$	0.463	0.325	0.535	

- Nothing in a network keeps it below 1.

Worked: does the network keep energy?

Extra 2's furnace: the share of light sent back, $\int \rho \cos\theta \, d\omega$

(red, green, blue; each must stay below 1)

eye at	the paint			the network			one lobe
$0°$	0.123	0.342	0.453	0.131	0.344	0.463	
$45°$	0.269	0.226	0.447	0.261	0.236	0.464	
$75°$	0.463	0.325	0.535	0.478	0.324	0.536	

- Nothing in a network keeps it below 1.
- Here it follows the paint, at every angle.

Worked: does the network keep energy?

Extra 2's furnace: the share of light sent back, $\int \rho \cos\theta \, d\omega$

(red, green, blue; each must stay below 1)

eye at	the paint			the network			one lobe		
$0°$	0.123	0.342	0.453	0.131	0.344	0.463	0.232	0.167	0.448
$45°$	0.269	0.226	0.447	0.261	0.236	0.464	0.232	0.167	0.450
$75°$	0.463	0.325	0.535	0.478	0.324	0.536	0.226	0.161	0.442

- Nothing in a network keeps it below 1.
- Here it follows the paint, at every angle.
- The lobe: the wrong colours.

Worked: does the network keep energy?

Extra 2's furnace: the share of light sent back, $\int \rho \cos\theta \, d\omega$

(red, green, blue; each must stay below 1)

eye at	the paint			the network			one lobe		
$0°$	0.123	0.342	0.453	0.131	0.344	0.463	0.232	0.167	0.448
$45°$	0.269	0.226	0.447	0.261	0.236	0.464	0.232	0.167	0.450
$75°$	0.463	0.325	0.535	0.478	0.324	0.536	0.226	0.161	0.442

Check every fitted material this way before it ships.

- Nothing in a network keeps it below 1.
- Here it follows the paint, at every angle.
- The lobe: the wrong colours.

A network has no inverse: pick by a lobe

Extra 2: GGX

random numbers $\rightarrow \mathbf{l}$
with a known $p(\mathbf{l})$

- A path needs directions picked by ρ.

A network has no inverse: pick by a lobe

Extra 2: GGX

random numbers $\rightarrow \mathbf{l}$
with a known $p(\mathbf{l})$



$(\mathbf{l}, \mathbf{v}) \rightarrow \rho$
no way back

- A path needs directions picked by ρ.
- A network answers ρ, but cannot pick.

A network has no inverse: pick by a lobe

Extra 2: GGX

random numbers $\to \mathbf{l}$
with a known $p(\mathbf{l})$



$(\mathbf{l}, \mathbf{v}) \to \rho$
no way back

- A path needs directions picked by ρ.
- A network answers ρ, but cannot pick.
- Pick by a lobe fitted to it; divide by its p.

pick $\mathbf{l}$ by C12's lobe (70%) or the cosine (30%);

$$\text{weight} = \frac{\rho_{\text{net}}(\mathbf{l}, \mathbf{v}) \cos\theta}{p_{\text{lobe+cos}}(\mathbf{l})}$$

A network has no inverse: pick by a lobe

Extra 2: GGX

random numbers $\to \mathbf{l}$
with a known $p(\mathbf{l})$



$(\mathbf{l}, \mathbf{v}) \to \rho$
no way back

- A path needs directions picked by ρ.
- A network answers ρ, but cannot pick.
- Pick by a lobe fitted to it; divide by its p.

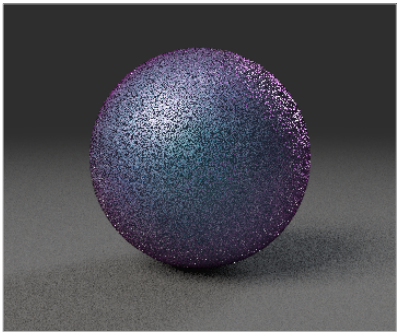
pick $\mathbf{l}$ by C12's lobe (70%) or the cosine (30%);

$$\text{weight} = \frac{\rho_{\text{net}}(\mathbf{l}, \mathbf{v}) \cos\theta}{p_{\text{lobe+cos}}(\mathbf{l})}$$

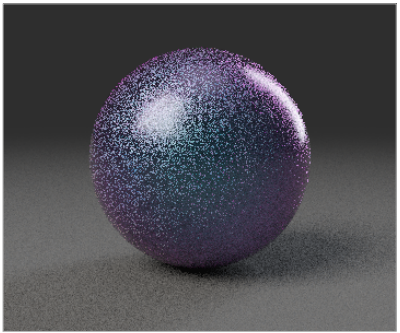
the lobe only picks; the network still says how much

Rendered: cosine picks against lobe picks

the network, 16 paths a pixel, no softbox picks



picked by the cosine

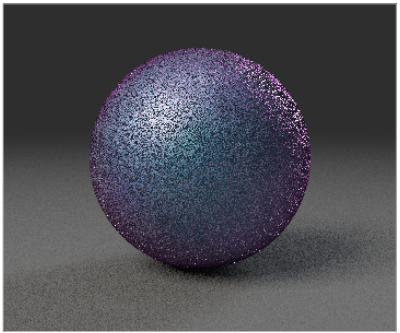


picked by the lobe and the cosine

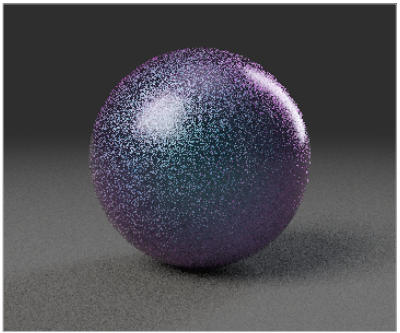
- The same network, the same 16 paths a pixel.

Rendered: cosine picks against lobe picks

the network, 16 paths a pixel, no softbox picks



picked by the cosine

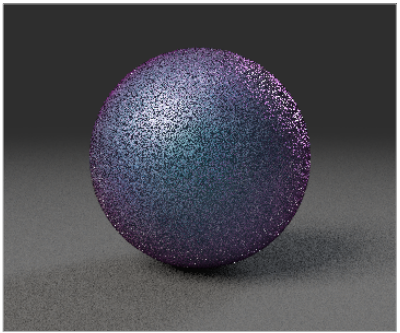


picked by the lobe and the cosine

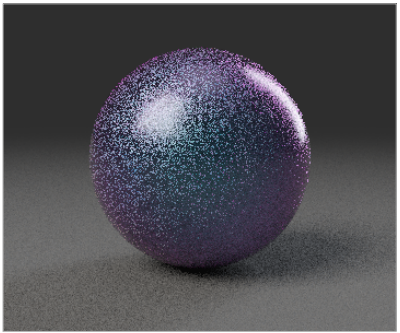
- The same network, the same 16 paths a pixel.
- Cosine picks rarely find the highlights.

Rendered: cosine picks against lobe picks

the network, 16 paths a pixel, no softbox picks



picked by the cosine

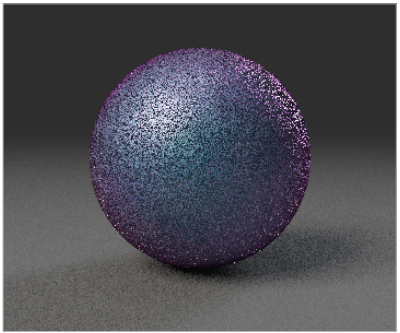


picked by the lobe and the cosine

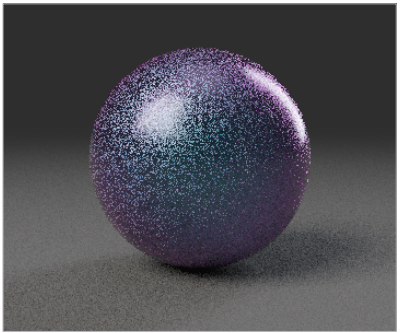
- The same network, the same 16 paths a pixel.
- Cosine picks rarely find the highlights.
- Lobe picks find them: far less noise.

Rendered: cosine picks against lobe picks

the network, 16 paths a pixel, no softbox picks



picked by the cosine



picked by the lobe and the cosine

- The same network, the same 16 paths a pixel.
- Cosine picks rarely find the highlights.
- Lobe picks find them: far less noise.

Both are right on average; only the noise differs.

Quick check 1: a measured material

1. Lamp and eye are each $45°$ from $\mathbf{n}$, on opposite sides, in one plane. What are θ_h and θ_d?

2. What is the network's first input, $\sqrt{\theta_h/90°}$, for $\theta_h = 22.5°$?

3. How many numbers has a network $3 \to 16 \to 16 \to 3$?

4. A measured ρ is 99 and the network says 89. How far apart are they in $\log(1+\rho)$?

Quick check 1: a measured material

1. Lamp and eye are each $45°$ from $\mathbf{n}$, on opposite sides, in one plane. What are θ_h and θ_d?

2. What is the network's first input, $\sqrt{\theta_h/90°}$, for $\theta_h = 22.5°$?

3. How many numbers has a network $3 \to 16 \to 16 \to 3$?

4. A measured ρ is 99 and the network says 89. How far apart are they in $\log(1+\rho)$?

1: $\theta_h = 0$, $\theta_d = 45°$ (a mirror pair). 2: $\sqrt{0.25} = 0.5$. 3: $64 + 272 + 51 = 387$. 4: $\ln(100/90) = 0.105$.

Recap: the BRDF, as a network

network

the BRDF $\rho(\mathbf{l}, \mathbf{v})$	the textures colour, normal, ... at (u, v)	the rest of the path light from there
a formula: too simple	a table: too big	a recursion: too slow

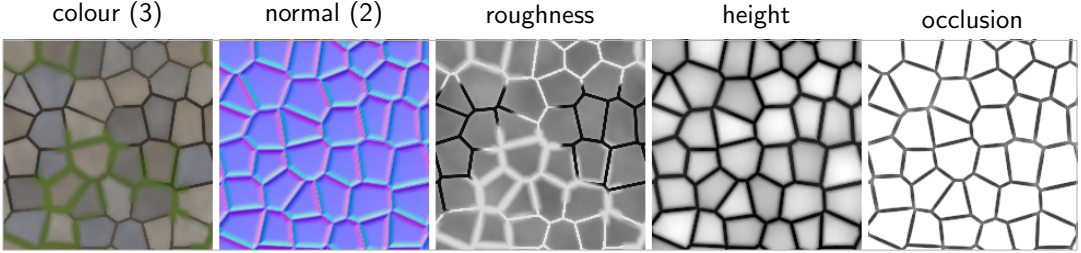
network for	trained on	when	ours
2 the BRDF	measurements	once a material	612 numbers

Part 2: a material no formula fits, in 612 numbers.

Neural textures

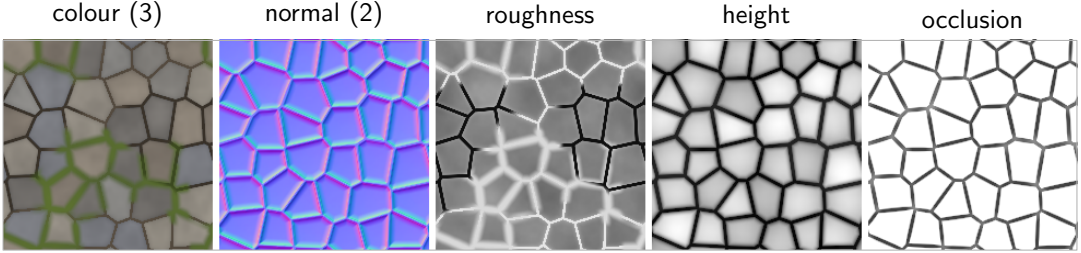
Memory, traded for arithmetic

A material is a stack of textures



our cobblestones: 192×192 texels, 8 numbers each, made by the script

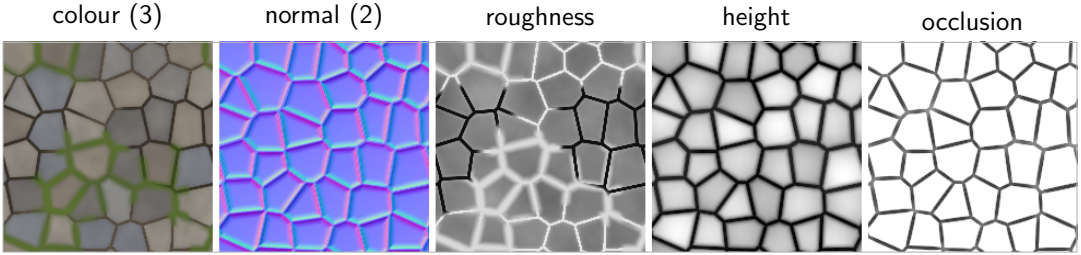
A material is a stack of textures



our cobblestones: 192×192 texels, 8 numbers each, made by the script

- Eight numbers a texel, read at every hit.

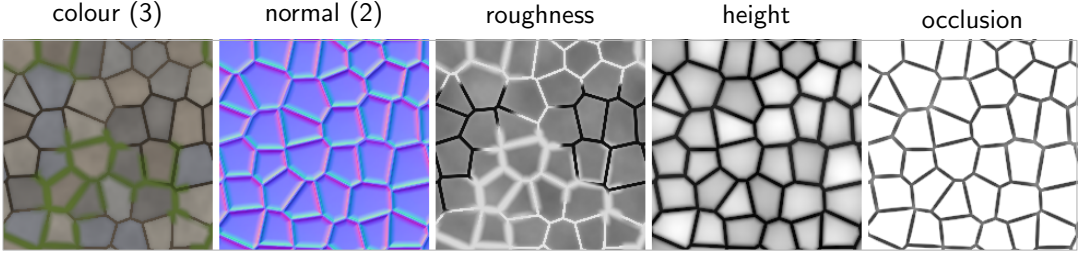
A material is a stack of textures



our cobblestones: 192×192 texels, 8 numbers each, made by the script

- Eight numbers a texel, read at every hit.
- Not independent: every map shows the same stones.

A material is a stack of textures



our cobblestones: 192×192 texels, 8 numbers each, made by the script

- Eight numbers a texel, read at every hit.
- Not independent: every map shows the same stones.
- Stored apart, they repeat that shape eight times.

Worked: one material's memory

One material at 4096×4096 texels, 8 numbers a texel, a byte each

Try first: how many megabytes?

Worked: one material's memory

One material at 4096×4096 texels, 8 numbers a texel, a byte each

Try first: how many megabytes?

$4096 \times 4096 \times 8 =$ **134 MB** (a third more with its mipmaps)

Worked: one material's memory

One material at 4096×4096 texels, 8 numbers a texel, a byte each

Try first: how many megabytes?

$4096 \times 4096 \times 8 =$ **134 MB** (a third more with its mipmaps)

a scene with 100 such materials: **13 GB**

a graphics card holds 8 to 24 GB, for everything (from memory)

- Textures fill most of a game's memory.

Worked: one material's memory

One material at 4096×4096 texels, 8 numbers a texel, a byte each

Try first: how many megabytes?

$4096 \times 4096 \times 8 =$ **134 MB** (a third more with its mipmaps)

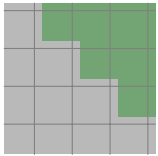
a scene with 100 such materials: **13 GB**

a graphics card holds 8 to 24 GB, for everything (from memory)

so textures are kept compressed, in a form the card reads directly

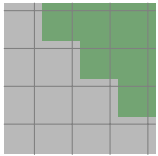
- Textures fill most of a game's memory.

Block compression: sixteen texels, two ends

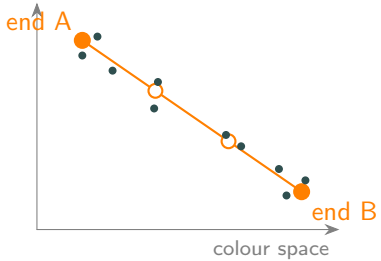


16 texels

Block compression: sixteen texels, two ends

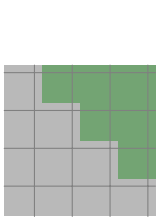


16 texels

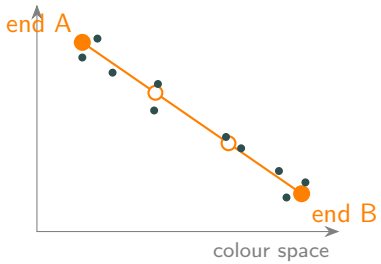


- Each block: a line between two colours.

Block compression: sixteen texels, two ends



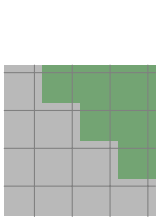
16 texels



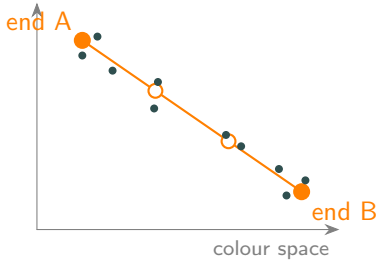
- Each block: a line between two colours.
- Each texel: which of four points on it.

BC1: two ends 2×16 bits, 16 texels $\times$ 2 bits: 64 bits, **4 bits a texel**

Block compression: sixteen texels, two ends



16 texels

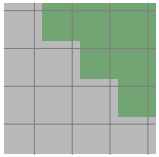


- Each block: a line between two colours.
- Each texel: which of four points on it.

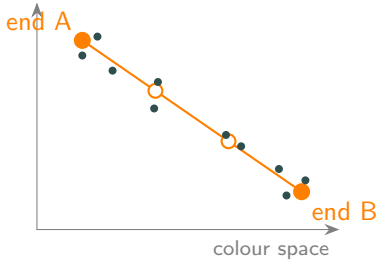
BC1: two ends 2×16 bits, 16 texels $\times$ 2 bits: 64 bits, **4 bits a texel**

BC4, one number: ends 2×8, 16 $\times$ 3 bits: also 4 bits a texel

Block compression: sixteen texels, two ends



16 texels



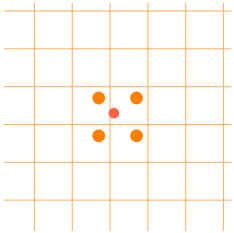
- Each block: a line between two colours.
- Each texel: which of four points on it.
- Decoded by the card for free.

BC1: two ends 2×16 bits, 16 texels $\times$ 2 bits: 64 bits, **4 bits a texel**

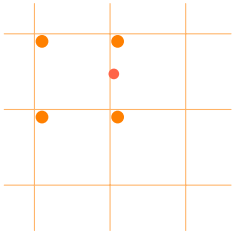
BC4, one number: ends 2×8, 16×3 bits: also 4 bits a texel

our set: BC1 + BC5 (normal) + $3\times$ BC4 $= 4 + 8 + 12 =$ **24 bits** a texel

Grids of learned numbers, and a decoder



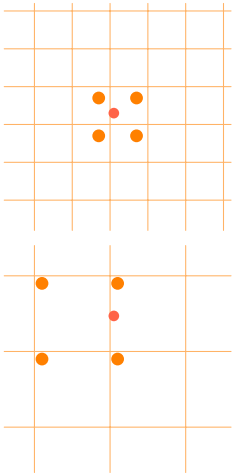
48×48 points,
12 numbers each

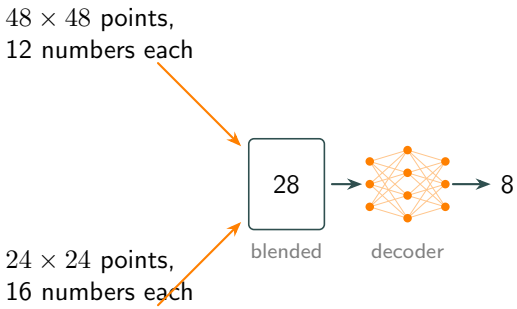


24×24 points,
16 numbers each

- Grids of numbers nobody painted.

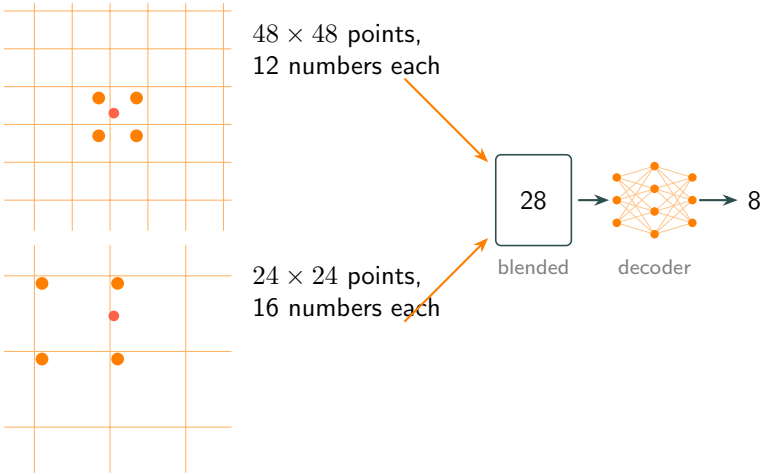
Grids of learned numbers, and a decoder





- Grids of numbers nobody painted.
- Blend the 4 nearest points of each; decode.

Grids of learned numbers, and a decoder



- Grids of numbers nobody painted.
- Blend the 4 nearest points of each; decode.
- Grids and decoder fitted together.

Worked: bits a texel

Our set: 192×192 texels; every grid number kept in 4 bits

Try first: bits a texel for the two grids?

Worked: bits a texel

Our set: 192×192 texels; every grid number kept in 4 bits

Try first: bits a texel for the two grids?

fine grid: a point for $4 \times 4 = 16$ texels: $12 \times 4/16 =$ **3 bits**

Worked: bits a texel

Our set: 192×192 texels; every grid number kept in 4 bits

Try first: bits a texel for the two grids?

fine grid: a point for $4 \times 4 = 16$ texels: $12 \times 4/16 =$ **3 bits**

coarse grid: a point for $8 \times 8 = 64$ texels: $16 \times 4/64 =$ **1 bit**

Worked: bits a texel

Our set: 192×192 texels; every grid number kept in 4 bits

Try first: bits a texel for the two grids?

fine grid: a point for $4 \times 4 = 16$ texels: $12 \times 4/16 =$ **3 bits**

coarse grid: a point for $8 \times 8 = 64$ texels: $16 \times 4/64 =$ **1 bit**

the decoder: 6,536 weights $\times$ 16 bits, shared by every texel:

2.8 bits a texel here; at 4096×4096, **0.006**

Worked: bits a texel

Our set: 192×192 texels; every grid number kept in 4 bits

Try first: bits a texel for the two grids?

fine grid: a point for $4 \times 4 = 16$ texels: $12 \times 4/16 =$ **3 bits**

coarse grid: a point for $8 \times 8 = 64$ texels: $16 \times 4/64 =$ **1 bit**

the decoder: 6,536 weights $\times$ 16 bits, shared by every texel:

2.8 bits a texel here; at 4096×4096, **0.006**

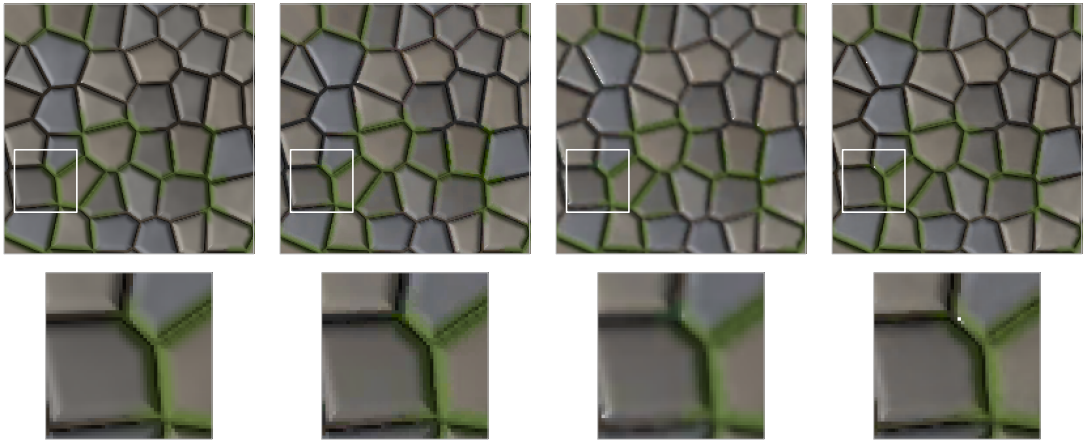
$3 + 1 =$ **4 bits** a texel, against block compression's 24

- A sixth of the memory.

Rendered: the same memory, two ways

the maps: 64 bits

blocks: 24 bits



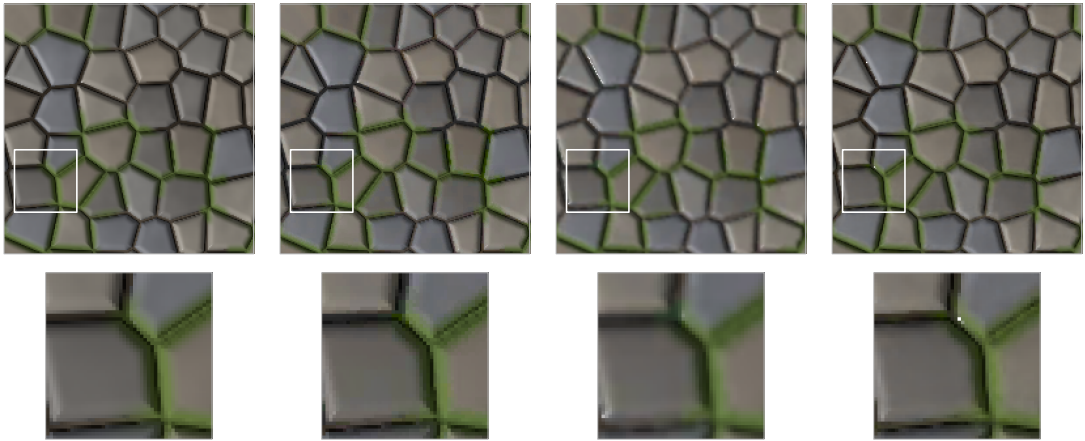
40.0 dB

Rendered: the same memory, two ways

the maps: 64 bits

blocks: 24 bits

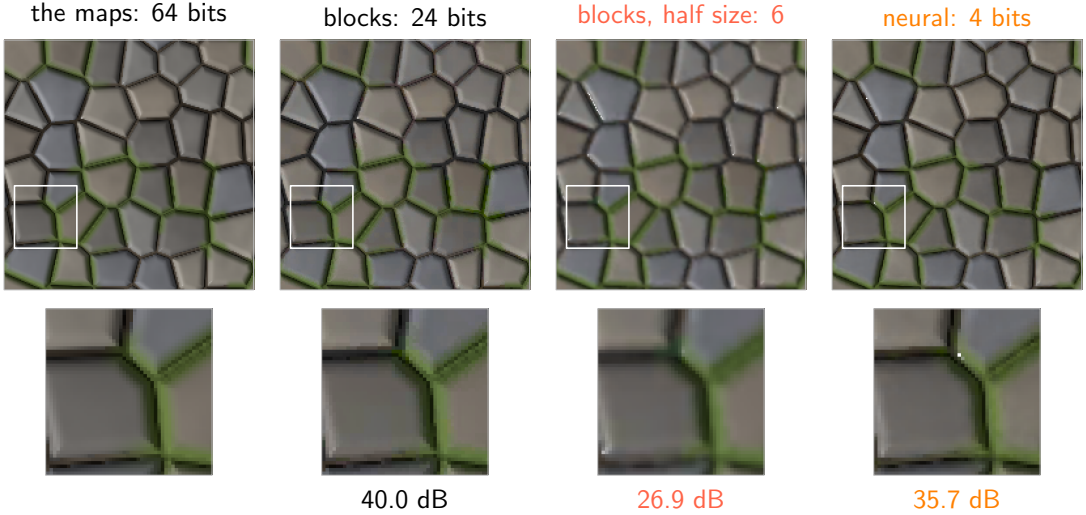
blocks, half size: 6



40.0 dB

26.9 dB

Rendered: the same memory, two ways



Decoding in the shader: arithmetic for memory

blocks

5 reads (one a map)
decoded by fixed
hardware: free

neural

Decoding in the shader: arithmetic for memory

blocks

5 reads (one a map)
decoded by fixed
hardware: free

neural

8 grid reads, then
$28 \times 64 + 64 \times 64 + 64 \times 8$
$=$ **6,400** multiply-adds

Decoding in the shader: arithmetic for memory

blocks

5 reads (one a map)
decoded by fixed
hardware: free

neural

8 grid reads, then
$28{\times}64 + 64{\times}64 + 64{\times}8$
$=$ **6,400** multiply-adds

- Memory saved, arithmetic spent.

a frame at 1920×1080, one read a pixel: 13 billion multiply-adds

Decoding in the shader: arithmetic for memory

blocks

5 reads (one a map)
decoded by fixed
hardware: free

neural

8 grid reads, then
$28{\times}64 + 64{\times}64 + 64{\times}8$
$=$ **6,400** multiply-adds

- Memory saved, arithmetic spent.
- Random access, as a shader needs.

a frame at 1920×1080, one read a pixel: 13 billion multiply-adds

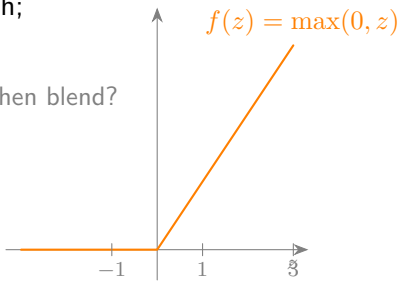
any texel decoded alone: no neighbours, no order

Worked: filtering breaks

A far pixel covers two texels, half each;

their grid numbers: -1 and 3.

Try first: blend, then decode? Decode, then blend?



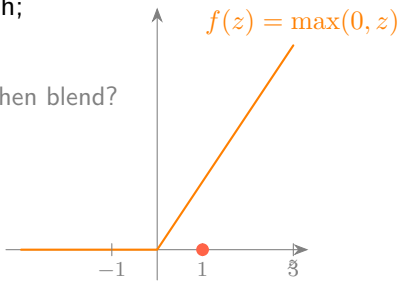
Worked: filtering breaks

A far pixel covers two texels, half each;

their grid numbers: -1 and 3.

Try first: blend, then decode? Decode, then blend?

blend, then decode: $f(1) = \mathbf{1}$



Worked: filtering breaks

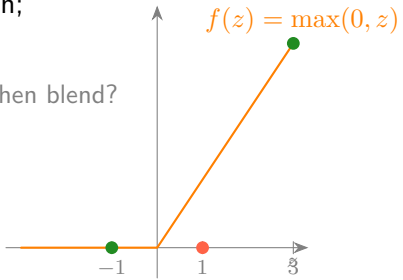
A far pixel covers two texels, half each;

their grid numbers: -1 and 3.

Try first: blend, then decode? Decode, then blend?

blend, then decode: $f(1) = \mathbf{1}$

decode, then blend: $\frac{0+3}{2} = \mathbf{1.5}$



Worked: filtering breaks

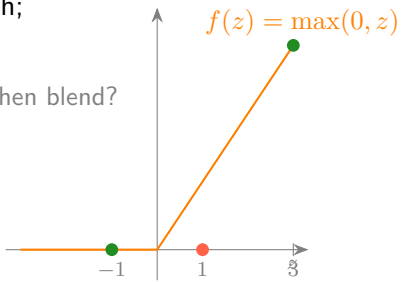
A far pixel covers two texels, half each;

their grid numbers: -1 and 3.

Try first: blend, then decode? Decode, then blend?

blend, then decode: $f(1) = \mathbf{1}$

decode, then blend: $\frac{0+3}{2} = \mathbf{1.5}$



the right answer is 1.5: what the texels look like, blended

- Blending the grids is not blending the texels.

Worked: filtering breaks

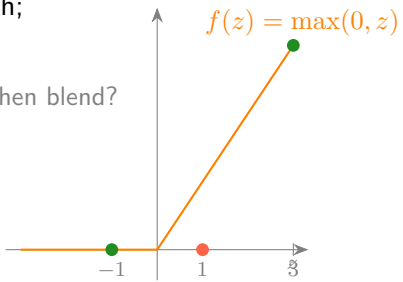
A far pixel covers two texels, half each;

their grid numbers: -1 and 3.

Try first: blend, then decode? Decode, then blend?

blend, then decode: $f(1) = \mathbf{1}$

decode, then blend: $\frac{0+3}{2} = \mathbf{1.5}$

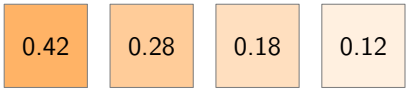


the right answer is 1.5: what the texels look like, blended

decoding every texel under a far pixel: too slow

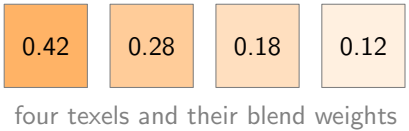
- Blending the grids is not blending the texels.

Stochastic filtering: one texel a frame



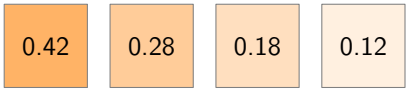
four texels and their blend weights

Stochastic filtering: one texel a frame

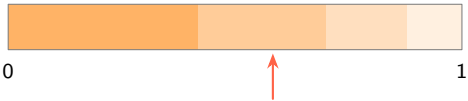


- Pick one texel, with its weight as its chance.

Stochastic filtering: one texel a frame



four texels and their blend weights

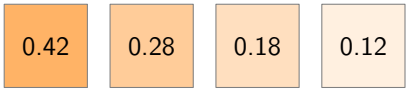


a random number: decode the second only

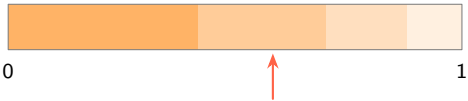
on average over frames: $\sum w_k \, f(z_k)$, the right blend

- Pick one texel, with its weight as its chance.
- Decode only it: one decode a pixel.

Stochastic filtering: one texel a frame



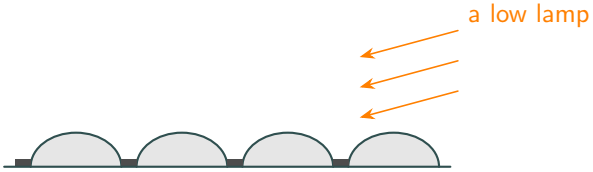
four texels and their blend weights



on average over frames: $\sum w_k f(z_k)$, the right blend

- Pick one texel, with its weight as its chance.
- Decode only it: one decode a pixel.
- Frames average it: Extra 6's accumulation.

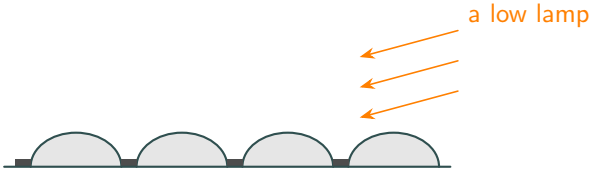
NeuMIP: the decoder is told the light



real bumps: hollows in shadow

- Bumps shade each other.

NeuMIP: the decoder is told the light



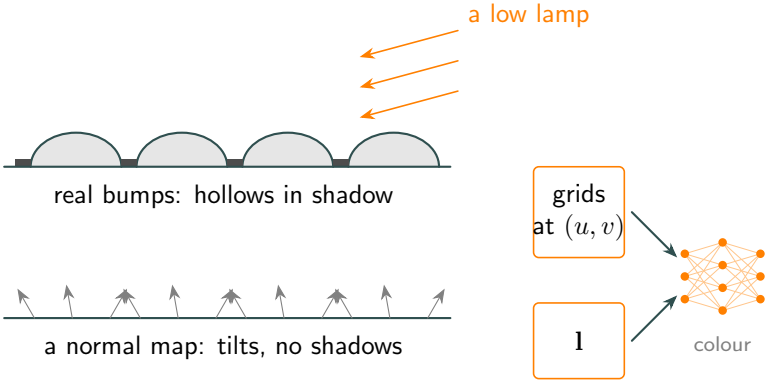
real bumps: hollows in shadow



a normal map: tilts, no shadows

- Bumps shade each other.
- A normal map only tilts.

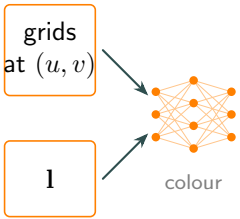
NeuMIP: the decoder is told the light



- Bumps shade each other.
- A normal map only tilts.
- Give the decoder the lamp's direction.

NeuMIP: the decoder is told the light





- Bumps shade each other.
- A normal map only tilts.
- Give the decoder the lamp's direction.
- Train it on renders of the real bumps.

Rendered: true bumps, a normal map, the network

from above; a lamp 14° up, from the left, never in training



real bumps | normal map | neural material

Rendered: true bumps, a normal map, the network

from above; a lamp 14° up, from the left, never in training



real bumps | normal map | neural material

- Normal map: no shadows in the gaps.

Rendered: true bumps, a normal map, the network

from above; a lamp 14° up, from the left, never in training



real bumps | normal map | neural material

- Normal map: no shadows in the gaps.
- Network: the shadows, a little soft.

squared error: normal map 0.0012, network **0.00014**: about 9 times less

Quick check 2: textures

1. BC1 keeps 4 bits a texel. How many bits is one 4×4 block?

2. A grid point serves 8×8 texels and holds 16 numbers of 4 bits. Bits a texel?

3. Decoder $f(z) = \max(0, z)$; two texels, half each, with -2 and 4. Blend then decode? Decode then blend?

4. Stochastic filtering, weights 0.75 and 0.25: in how many frames is the second texel decoded?

Quick check 2: textures

1. BC1 keeps 4 bits a texel. How many bits is one 4×4 block?

2. A grid point serves 8×8 texels and holds 16 numbers of 4 bits. Bits a texel?

3. Decoder $f(z) = \max(0, z)$; two texels, half each, with -2 and 4. Blend then decode? Decode then blend?

4. Stochastic filtering, weights 0.75 and 0.25: in how many frames is the second texel decoded?

1: $16 \times 4 = 64$. 2: $16 \times 4/64 = 1$. 3: $f(1) = 1$; $(0 + 4)/2 = 2$ (right). 4: one in four, on average.

Recap: the textures, as a network

network

the BRDF
$\rho(\mathbf{l}, \mathbf{v})$

a formula: too simple

network

the textures
colour, normal, ... at (u, v)

a table: too big

the rest of the path
light from there

a recursion: too slow

network for	trained on	when	ours
2 the BRDF	measurements	once a material	612 numbers
3 the textures	the textures	once, at build	4 bits a texel

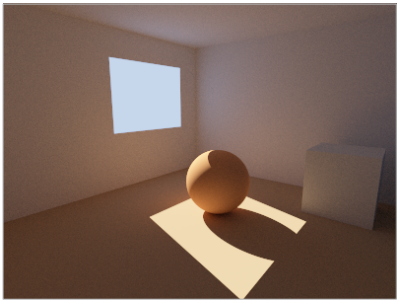
Part 3: a texture set in a sixth of block compression's memory.

The learned bounce

A cache trained while it renders

A room lit through one window

2,048 paths a pixel; every surface chalk

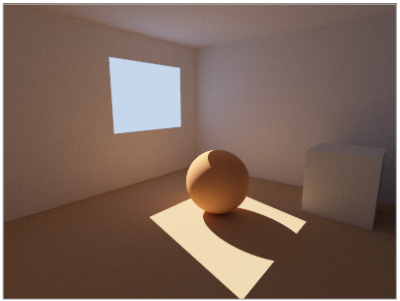


all the light

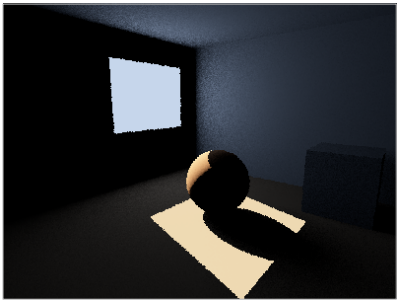
- Lit only by the sun and the sky, through one window.

A room lit through one window

2,048 paths a pixel; every surface chalk



all the light

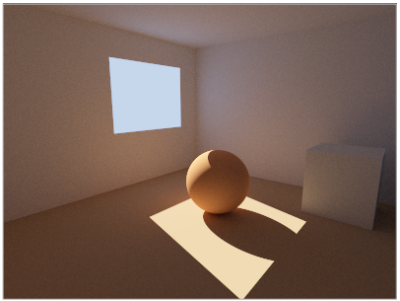


only the light straight from the window

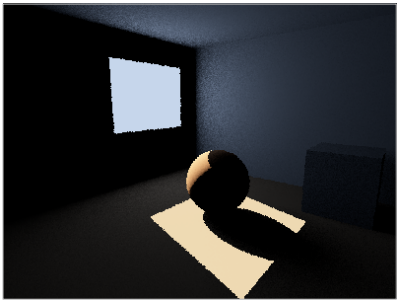
- Lit only by the sun and the sky, through one window.
- Straight from the window: the patch, little else.

A room lit through one window

2,048 paths a pixel; every surface chalk



all the light



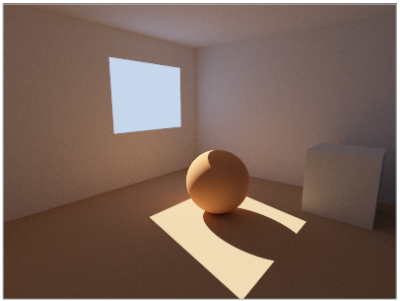
only the light straight from the window

- Lit only by the sun and the sky, through one window.
- Straight from the window: the patch, little else.
- In 91% of the pixels, most of the light has bounced.

Rendered: one path a pixel



1 path a pixel



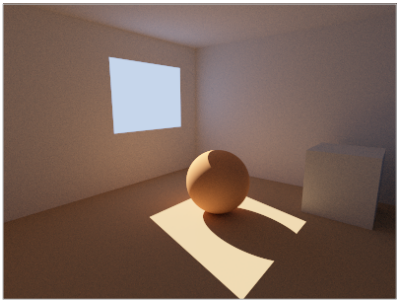
2,048 paths a pixel

- Lecture 20's tracer, at a game's budget.

Rendered: one path a pixel



1 path a pixel



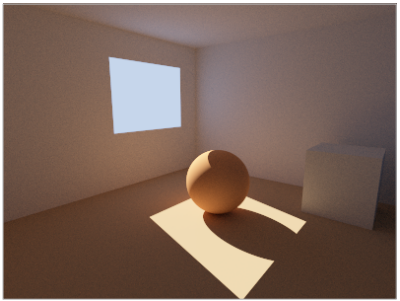
2,048 paths a pixel

- Lecture 20's tracer, at a game's budget.
- Sparkles: a bounce that happens to find the sun patch.

Rendered: one path a pixel



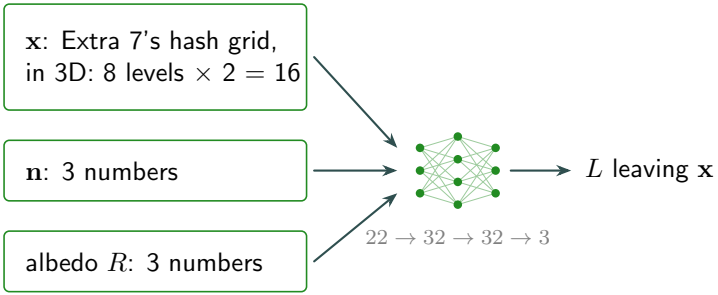
1 path a pixel



2,048 paths a pixel

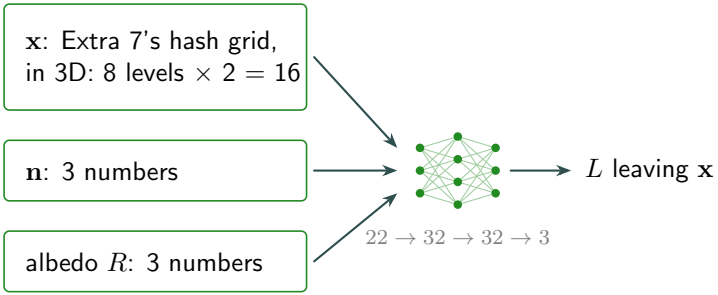
- Lecture 20's tracer, at a game's budget.
- Sparkles: a bounce that happens to find the sun patch.
- A path: 4.2 hits on average.

A cache: where, facing, colour in; light out



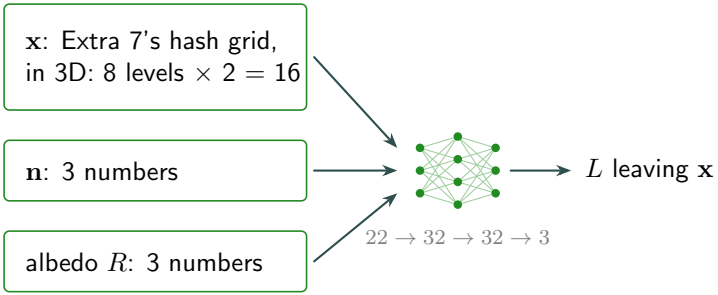
- Asked: how much light leaves $\mathbf{x}$?

A cache: where, facing, colour in; light out



- Asked: how much light leaves $\mathbf{x}$?
- Our room is all chalk: the same in every direction.

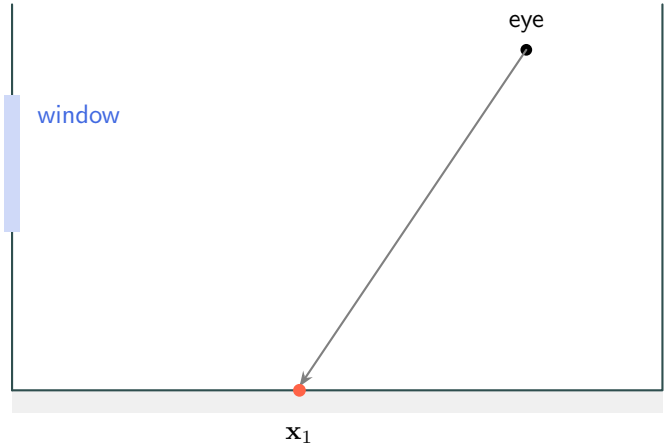
A cache: where, facing, colour in; light out



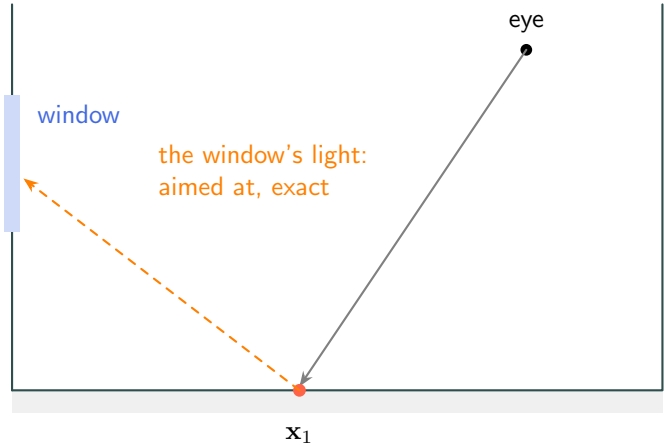
262,144 numbers in the tables, 1,891 weights

- Asked: how much light leaves $\mathbf{x}$?
- Our room is all chalk: the same in every direction.
- Knows the whole room, in one small store.

Stop after one bounce and ask

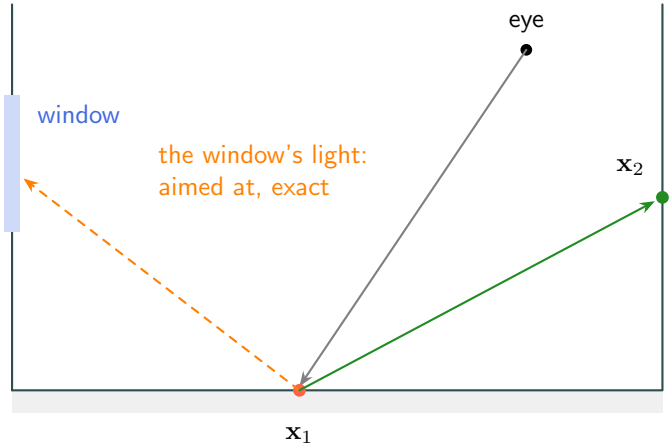


Stop after one bounce and ask



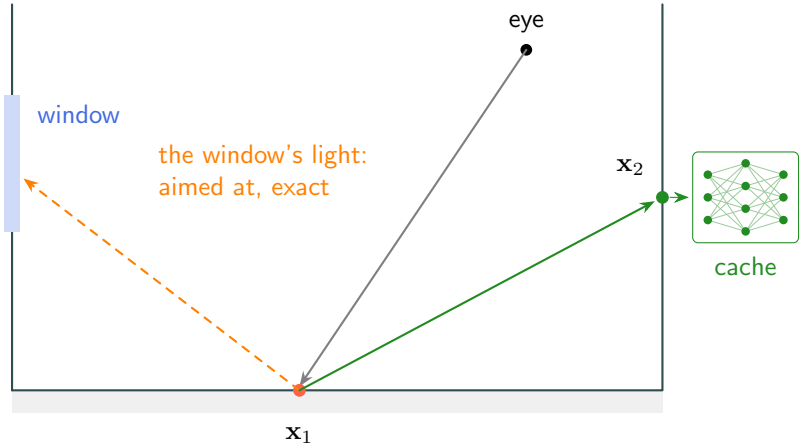
- Light straight from the window: traced.

Stop after one bounce and ask



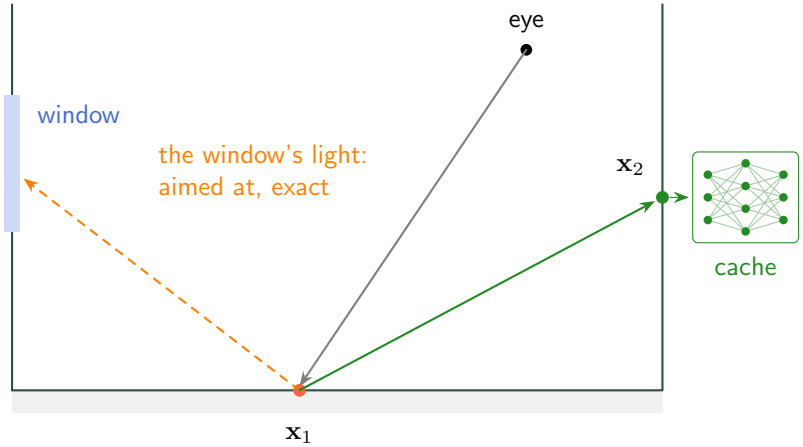
- Light straight from the window: traced.

Stop after one bounce and ask



- Light straight from the window: traced.
- The rest: one bounce, then ask.

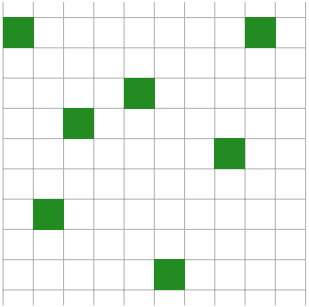
Stop after one bounce and ask



- Light straight from the window: traced.
- The rest: one bounce, then ask.

$L = R_1$ (window's light at $\mathbf{x}_1 +$ cache$(\mathbf{x}_2)$): two hits, not 4.2

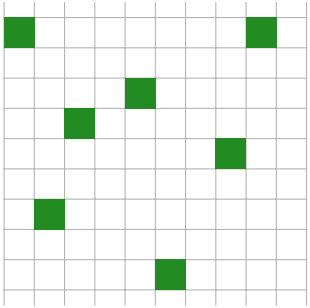
Training paths, while the frame renders



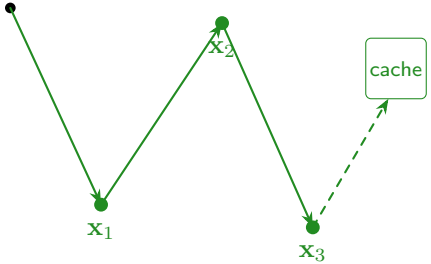
a frame's pixels

- A few pixels' paths go on.

Training paths, while the frame renders

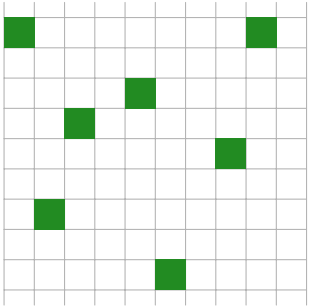


a frame's pixels

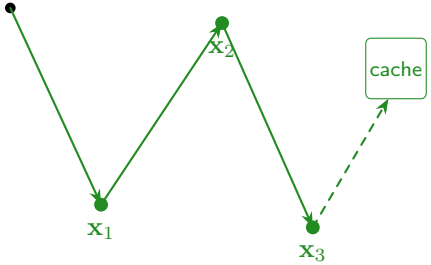


- A few pixels' paths go on.

Training paths, while the frame renders



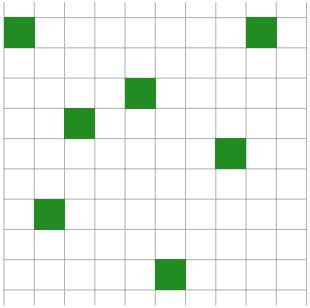
a frame's pixels



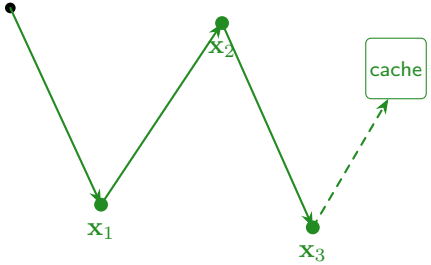
- A few pixels' paths go on.
- Every hit: a training example.

a target at every hit: the light that path found leaving it

Training paths, while the frame renders



a frame's pixels

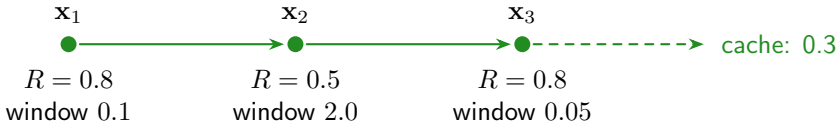


- A few pixels' paths go on.
- Every hit: a training example.

a target at every hit: the light that path found leaving it

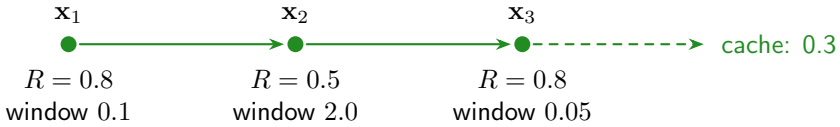
ours: 2,048 such paths a frame, about 5,800 targets, four Adam steps

Worked: one training path's targets



Try first: the three targets, from the back. $T = R\,(\text{window} + \text{next } T)$

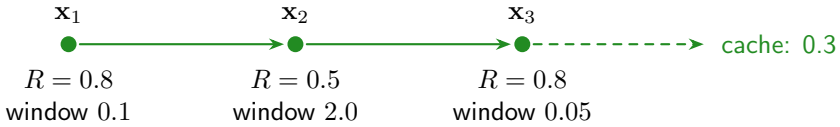
Worked: one training path's targets



Try first: the three targets, from the back. $T = R\,(\text{window} + \text{next } T)$

$T_3 = 0.8\,(0.05 + 0.3) = \mathbf{0.28}$

Worked: one training path's targets

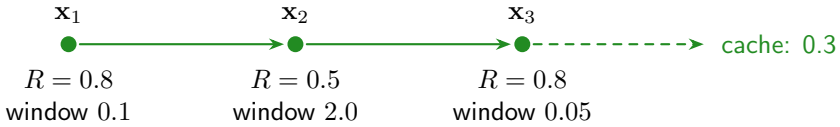


Try first: the three targets, from the back. $T = R\,(\text{window} + \text{next } T)$

$T_3 = 0.8\,(0.05 + 0.3) = \mathbf{0.28}$

$T_2 = 0.5\,(2.0 + 0.28) = \mathbf{1.14}$

Worked: one training path's targets



Try first: the three targets, from the back. $T = R\,(\text{window} + \text{next } T)$

$T_3 = 0.8\,(0.05 + 0.3) = \mathbf{0.28}$

$T_2 = 0.5\,(2.0 + 0.28) = \mathbf{1.14}$

$T_1 = 0.8\,(0.1 + 1.14) = \mathbf{0.992}$

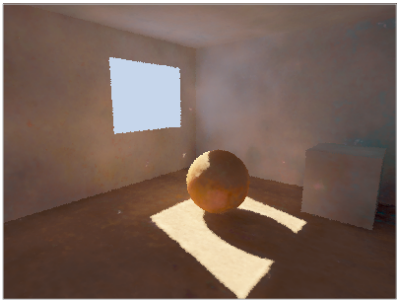
- One path, three examples.

Self-training: the cache at the end

the cache asked at the first hit, after 64 frames



the end of a training path: 0



the end: the cache's own answer

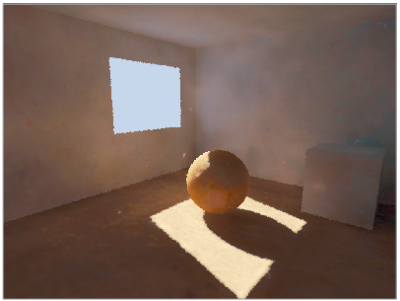
- Without: light after three hits is lost.

Self-training: the cache at the end

the cache asked at the first hit, after 64 frames



the end of a training path: 0



the end: the cache's own answer

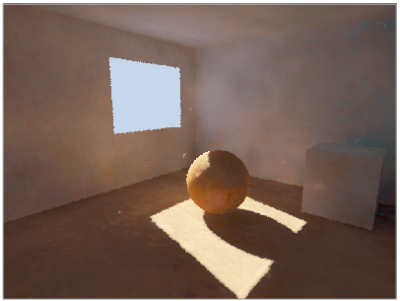
- Without: light after three hits is lost.
- With: the cache answers for the rest.

Self-training: the cache at the end

the cache asked at the first hit, after 64 frames



the end of a training path: 0



the end: the cache's own answer

- Without: light after three hits is lost.
- With: the cache answers for the rest.
- Each frame adds a bounce.

squared error against the reference: 0.0038 without, **0.0020** with

Worked: a loss for light from 0.02 to 8

relative loss: $\dfrac{(y-t)^2}{y^2+0.01}$ (y: the cache's answer, t: the target)

target	cache	squared	relative
sun patch, $t = 8$	7		
dark corner, $t = 0.02$	0.03		

Worked: a loss for light from 0.02 to 8

relative loss: $\dfrac{(y-t)^2}{y^2+0.01}$ (y: the cache's answer, t: the target)

target	cache	squared	relative
sun patch, $t = 8$	7	1	
dark corner, $t = 0.02$	0.03	0.0001	

- Plain: the patch counts 10,000 times more.

Worked: a loss for light from 0.02 to 8

relative loss: $\dfrac{(y-t)^2}{y^2+0.01}$ (y: the cache's answer, t: the target)

target	cache	squared	relative
sun patch, $t = 8$	7	1	$1/49.01 =$ **0.020**
dark corner, $t = 0.02$	0.03	0.0001	$0.0001/0.0109 =$ **0.0092**

- Plain: the patch counts 10,000 times more.
- Relative: about twice. Both are fitted.

Worked: a loss for light from 0.02 to 8

relative loss: $\dfrac{(y-t)^2}{y^2+0.01}$ (y: the cache's answer, t: the target)

target	cache	squared	relative
sun patch, $t = 8$	7	1	$1/49.01 =$ **0.020**
dark corner, $t = 0.02$	0.03	0.0001	$0.0001/0.0109 =$ **0.0092**

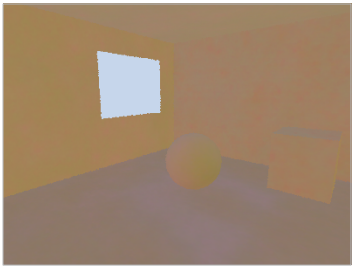
- Plain: the patch counts 10,000 times more.
- Relative: about twice. Both are fitted.

the y below the line is held fixed: only the top is nudged

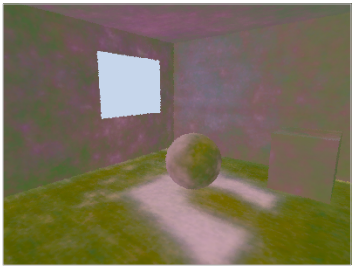
C13's logarithm did the same job for ρ

Rendered: the cache after 1, 4 and 64 frames

the cache asked at the first hit; squared error 0.040, 0.022, 0.0020



after 1 frame



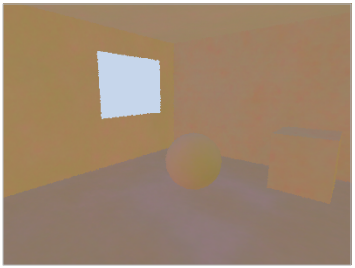
after 4



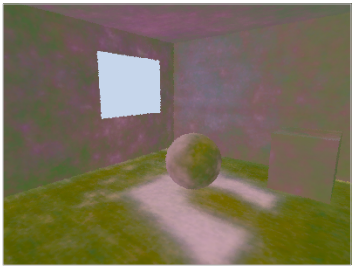
after 64

Rendered: the cache after 1, 4 and 64 frames

the cache asked at the first hit; squared error 0.040, 0.022, 0.0020



after 1 frame



after 4

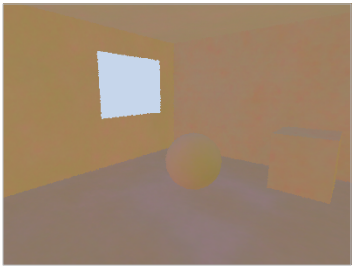


after 64

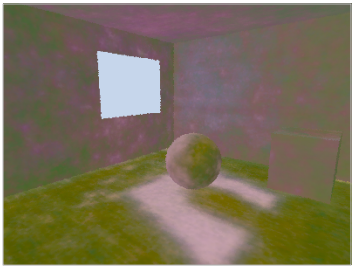
- From a guess to the room in about a second of frames.

Rendered: the cache after 1, 4 and 64 frames

the cache asked at the first hit; squared error 0.040, 0.022, 0.0020



after 1 frame



after 4



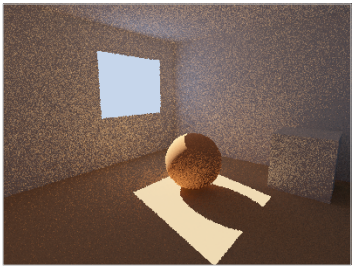
after 64

- From a guess to the room in about a second of frames.
- Move the sun, and it follows within a few frames.

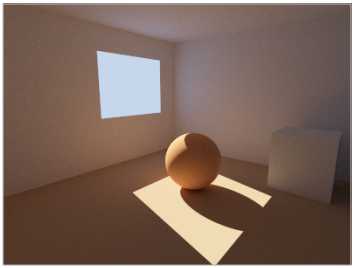
Why one bounce before asking



asked at the first hit



one bounce, then asked; 16 paths

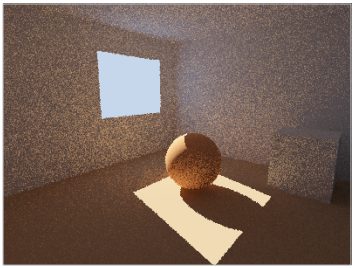


reference

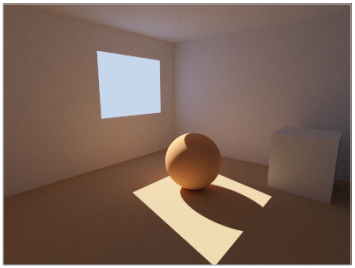
Why one bounce before asking



asked at the first hit



one bounce, then asked; 16 paths



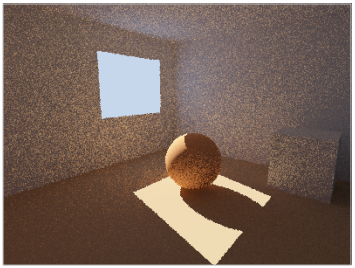
reference

- First hit: the cache's blotches are the picture, and they stay.

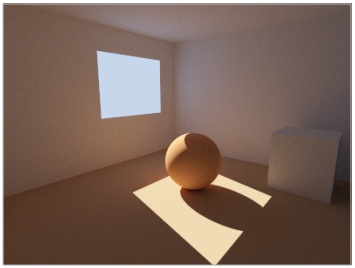
Why one bounce before asking



asked at the first hit



one bounce, then asked; 16 paths



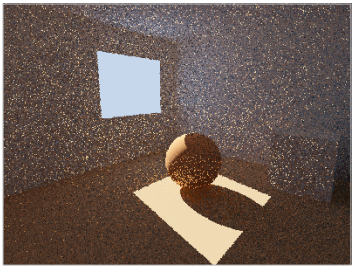
reference

- First hit: the cache's blotches are the picture, and they stay.
- After a bounce: its errors blur into noise, which averages away.

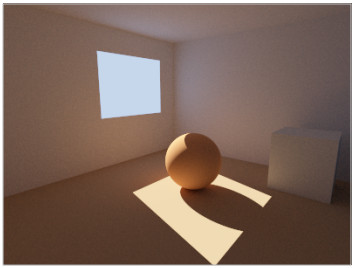
Rendered: one path a pixel, plain and cached



plain, 1 path



cached, 1 path

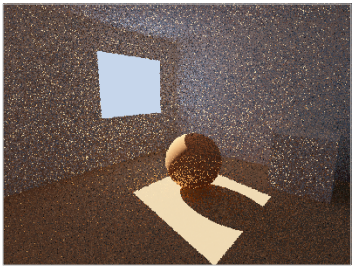


reference

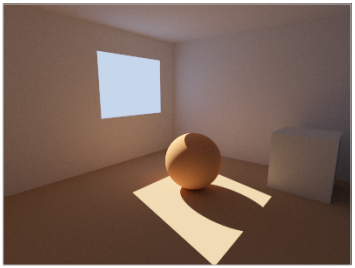
Rendered: one path a pixel, plain and cached



plain, 1 path



cached, 1 path



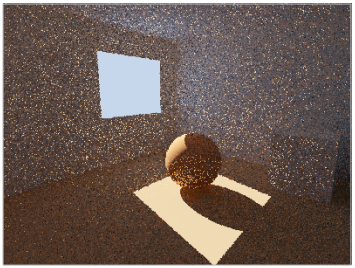
reference

squared error: plain **0.053**, cached **0.021**, two and a half times less

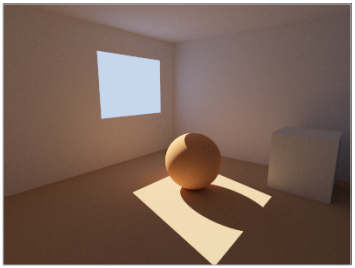
Rendered: one path a pixel, plain and cached



plain, 1 path



cached, 1 path

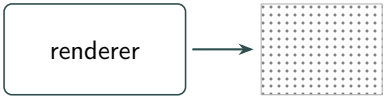


reference

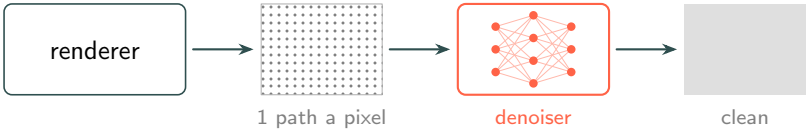
squared error: plain **0.053**, cached **0.021**, two and a half times less

hits a path: plain 4.2, cached 2 (and one call to the cache)

Networks after the renderer

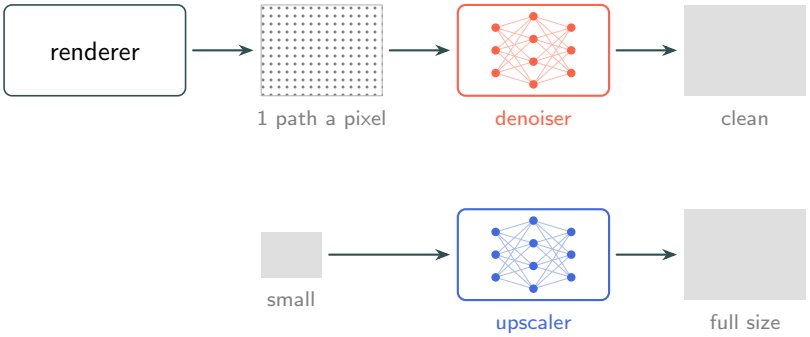


Networks after the renderer



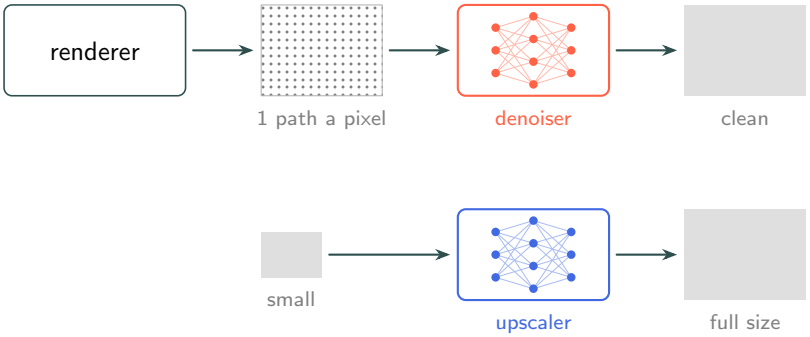
- Denoisers: trained on noisy and clean pairs.

Networks after the renderer



- Denoisers: trained on noisy and clean pairs.
- Upscalers: render small, show large.

Networks after the renderer



- Denoisers: trained on noisy and clean pairs.
- Upscalers: render small, show large.

Inside the renderer and after it, the same tool: a network fitted to right answers.

Recap: the rest of the path, as a network

network

the BRDF
$\rho(\mathbf{l}, \mathbf{v})$

a formula: too simple

network

the textures
colour, normal, ... at (u, v)

a table: too big

network

the rest of the path
light from there

a recursion: too slow

network for	trained on	when	ours
2 the BRDF	measurements	once a material	612 numbers
3 the textures	the textures	once, at build	4 bits a texel
4 the rest of the path	its own paths	every frame	stops after a bounce

Part 4: light from the rest of the path, learned while the frame renders.

Summary

1 A network the renderer calls

numbers in $\rightarrow$ numbers out

cheap arithmetic, if its weights stay on the chip

2 A measured material

$$(\theta_h, \theta_d, \phi_d) \rightarrow \log(1 + \rho)$$

612 numbers; directions picked by a fitted lobe

3 Neural textures

grids + decoder: 4 bits a texel

decode, then filter: one texel by its weight

4 The learned bounce

$$T = R\,(\text{window} + \text{next } T)$$

self-trained each frame; ask after one bounce

Reading

Matusik, Pfister, Brand and McMillan, "A data-driven reflectance model", SIGGRAPH 2003; **Rusinkiewicz**, "A new change of variables for efficient BRDF representation", 1998

Sztrajman, Rainer, Ritschel and Weyrich, "Neural BRDF representation and importance sampling", Computer Graphics Forum 2021

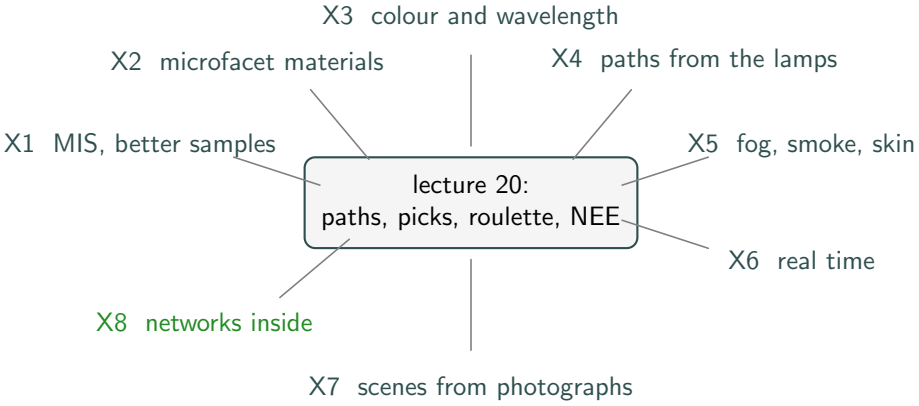
Vaidyanathan, Salvi, Wronski, Akenine-Möller, Ebelin and Lefohn, "Random-access neural compression of material textures", SIGGRAPH 2023

Kuznetsov, Mullia, Xu, Hasan and Ramamoorthi, "NeuMIP: multi-resolution neural materials", SIGGRAPH 2021; **Pharr, Wronski, Salvi and Fajardo**, "Filtering after shading with stochastic texture filtering", 2024

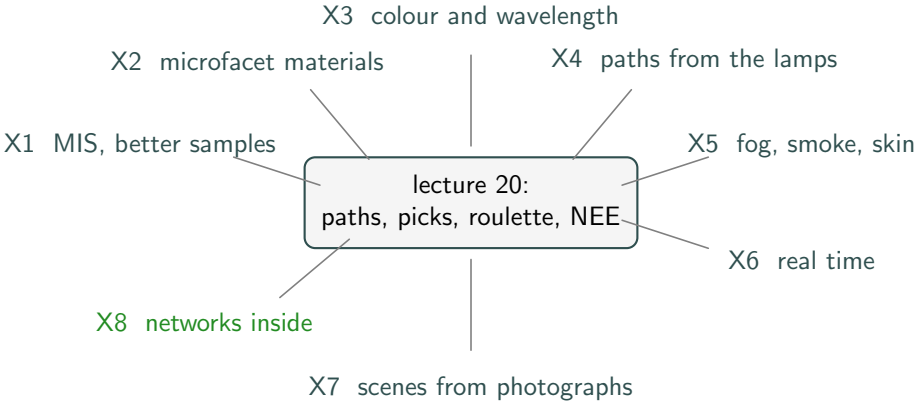
Müller, Rousselle, Novák and Keller, "Real-time neural radiance caching for path tracing", SIGGRAPH 2021

Chaitanya et al., "Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder", SIGGRAPH 2017

The end of the track



The end of the track



Next: PBRT (free online), Ray Tracing Gems I and II, and each year's SIGGRAPH papers.