Extra 6: Real Time

Path Tracing in Sixteen Milliseconds

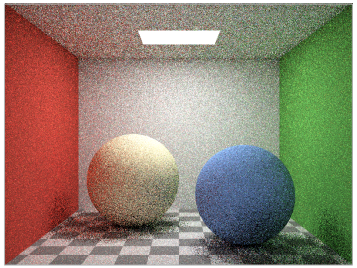
CS 453 — Computer Graphics

Fakhir Shaheen

Forman Christian University

Today: a path tracer at sixty frames a second

Extra 6: self-study, not on the exams

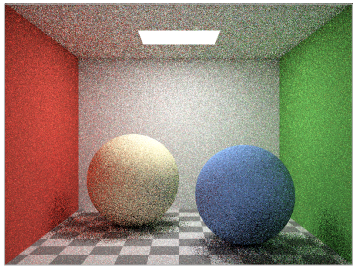


one path per pixel

- Our pictures took minutes. A game has 16.7 ms.

Today: a path tracer at sixty frames a second

Extra 6: self-study, not on the exams

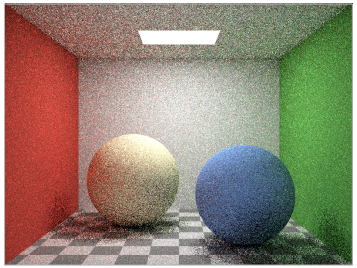


one path per pixel

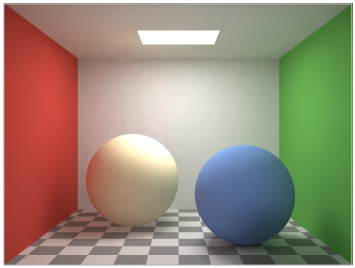
- Our pictures took minutes. A game has 16.7 ms.
- So: one path per pixel, and noise.

Today: a path tracer at sixty frames a second

Extra 6: self-study, not on the exams



one path per pixel

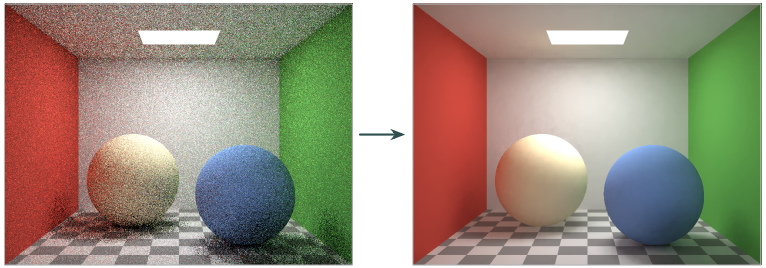


the same, after a filter

- Our pictures took minutes. A game has 16.7 ms.
- So: one path per pixel, and noise.
- A filter turns it into a usable picture.

Today: a path tracer at sixty frames a second

Extra 6: self-study, not on the exams



one path per pixel — the same, after a filter

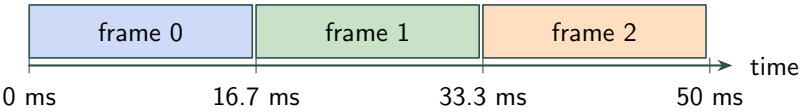
Two million pixels, sixty times a second, every path from scratch.

- Our pictures took minutes. A game has 16.7 ms.
- So: one path per pixel, and noise.
- A filter turns it into a usable picture.
- Today: the filter, shared lamp picks, the hardware.

One path per pixel

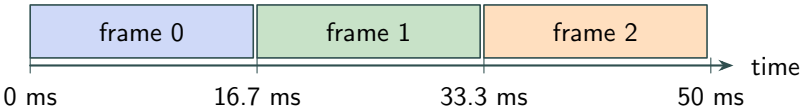
The budget, and a first hit for free

Sixteen milliseconds for a frame



- 60 frames a second: 16.7 ms each.

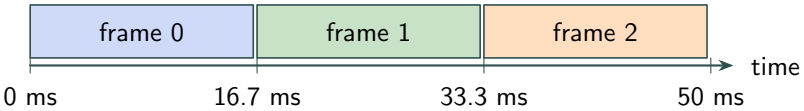
Sixteen milliseconds for a frame



$1920 \times 1080 = 2{,}073{,}600$ pixels full HD

- 60 frames a second: 16.7 ms each.
- Two million pixels in each.

Sixteen milliseconds for a frame

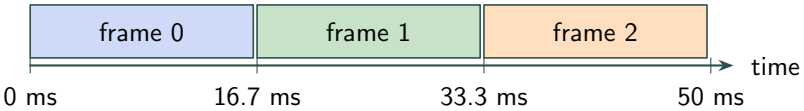


$1920 \times 1080 = 2{,}073{,}600$ pixels full HD

$\times$ 60 frames $= 124$ million pixels a second

- 60 frames a second: 16.7 ms each.
- Two million pixels in each.

Sixteen milliseconds for a frame



$1920 \times 1080 = 2{,}073{,}600$ pixels full HD

$\times$ 60 frames $= 124$ million pixels a second

$1\text{ s} \,/\, 124$ million $= 8$ nanoseconds per pixel

for everything: the first hit, the rays, the shading, the filter

- 60 frames a second: 16.7 ms each.
- Two million pixels in each.
- 8 ns per pixel, all in.

Worked: how many rays can a pixel have?

A ray tracing card: about 10^{10} rays a second (the 2018 figure for the first RTX card)

Worked: how many rays can a pixel have?

A ray tracing card: about 10^{10} rays a second (the 2018 figure for the first RTX card)

$\div$ 60 frames: about 167 million rays a frame

Worked: how many rays can a pixel have?

A ray tracing card: about 10^{10} rays a second (the 2018 figure for the first RTX card)

$\div$ 60 frames: about 167 million rays a frame

$\div$ 2.07 million pixels: about **80 rays per pixel**, if the card did nothing else

Worked: how many rays can a pixel have?

A ray tracing card: about 10^{10} rays a second (the 2018 figure for the first RTX card)

$\div$ 60 frames: about 167 million rays a frame

$\div$ 2.07 million pixels: about **80 rays per pixel**, if the card did nothing else

Our reference picture: 1024 paths $\times$ 8.6 rays $\approx$ **8,800 rays per pixel**

Worked: how many rays can a pixel have?

A ray tracing card: about 10^{10} rays a second (the 2018 figure for the first RTX card)

$\div$ 60 frames: about 167 million rays a frame

$\div$ 2.07 million pixels: about **80 rays per pixel**, if the card did nothing else

Our reference picture: 1024 paths $\times$ 8.6 rays $\approx$ **8,800 rays per pixel**

80: the whole card's budget

8,800: what our reference used, 110 times more

Worked: how many rays can a pixel have?

A ray tracing card: about 10^{10} rays a second (the 2018 figure for the first RTX card)

$\div$ 60 frames: about 167 million rays a frame

$\div$ 2.07 million pixels: about **80 rays per pixel**, if the card did nothing else

Our reference picture: 1024 paths $\times$ 8.6 rays $\approx$ **8,800 rays per pixel**

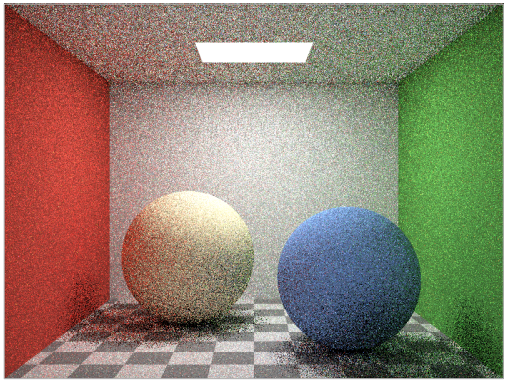
80: the whole card's budget

8,800: what our reference used, 110 times more

- So a game gets one path per pixel, a handful of rays, and noise.

Rendered: one path per pixel, against 1024

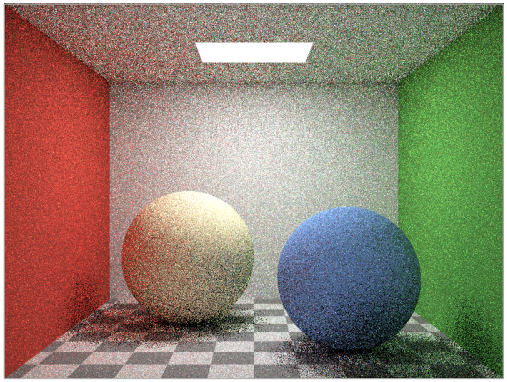
Extra 1's room: lecture 20's tracer with its shadow ray to the panel



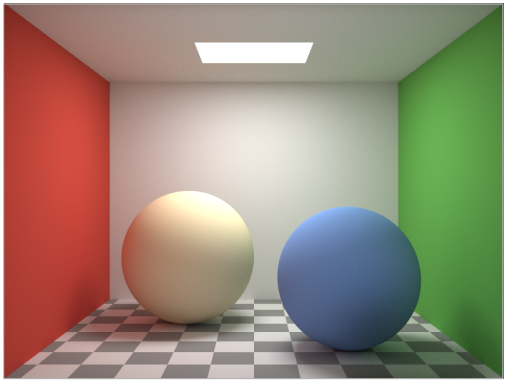
one path per pixel, through its centre

Rendered: one path per pixel, against 1024

Extra 1's room: lecture 20's tracer with its shadow ray to the panel



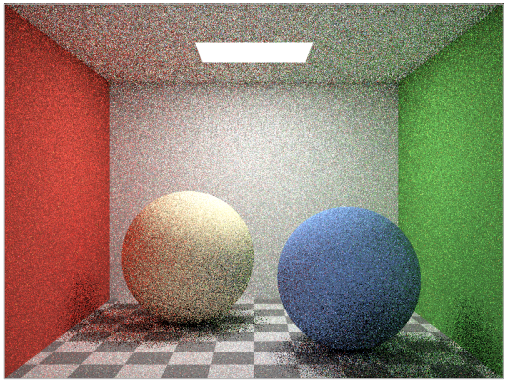
one path per pixel, through its centre



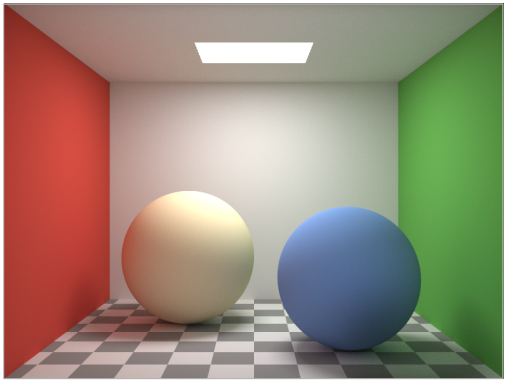
1024 paths per pixel

Rendered: one path per pixel, against 1024

Extra 1's room: lecture 20's tracer with its shadow ray to the panel



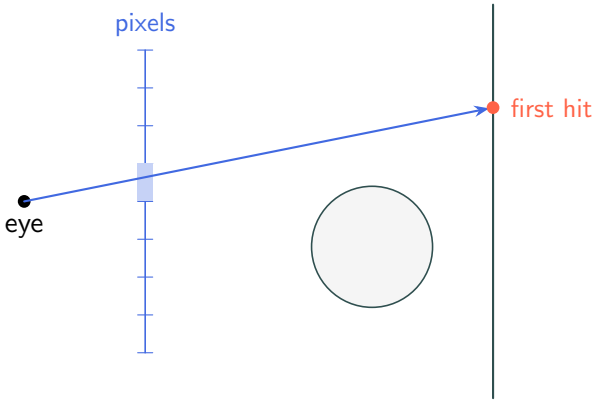
one path per pixel, through its centre



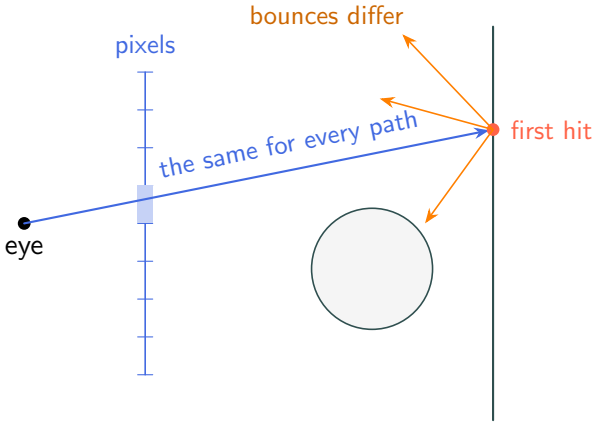
1024 paths per pixel

- The same tracer and the same room: only the number of paths differs.

Every path begins with the same first hit

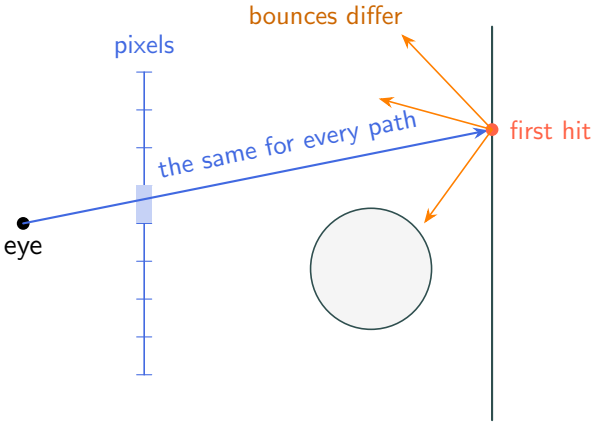


Every path begins with the same first hit

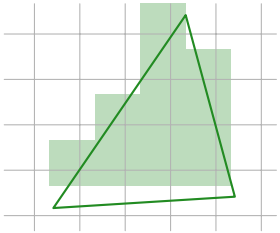


Through the pixel's centre, the first ray of every path is the same ray.

Every path begins with the same first hit



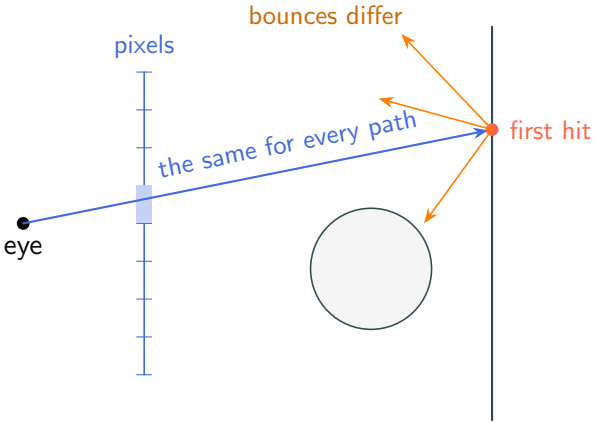
the rasteriser:
triangle by triangle



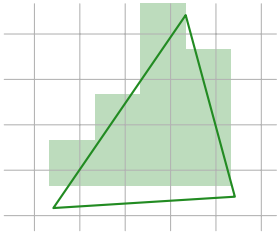
a triangle covers pixels

Through the pixel's centre, the first ray of every path is the same ray.

Every path begins with the same first hit



the rasteriser:
triangle by triangle

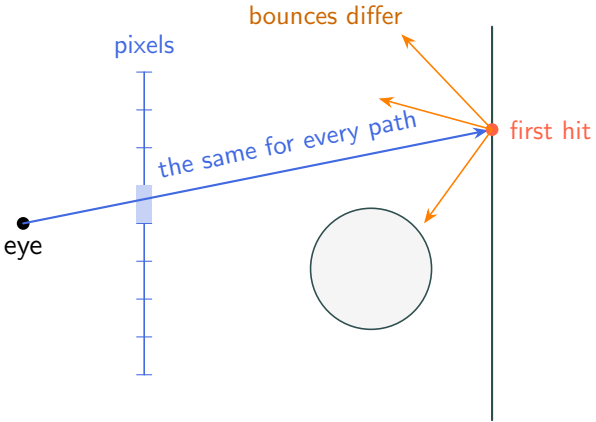


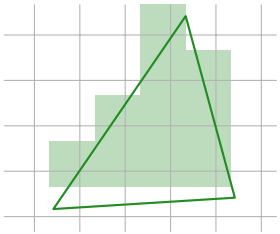
a triangle covers pixels

- Each pixel keeps the nearest: the depth buffer.

Through the pixel's centre, the first ray of every path is the same ray.

Every path begins with the same first hit

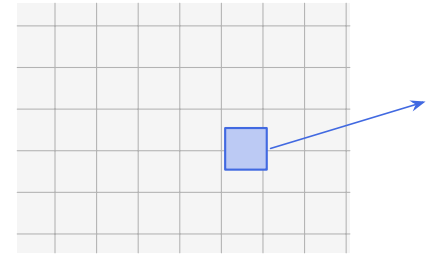




- Each pixel keeps the nearest: the depth buffer.
- Same first hits, far faster.

Through the pixel's centre, the first ray of every path is the same ray.

The G-buffer: what the first hit leaves in each pixel



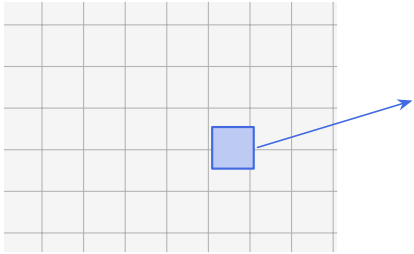
the window, one record per pixel

one pixel's record

position $\mathbf{x}$, depth

- G for geometry: the first hit, stored per pixel.

The G-buffer: what the first hit leaves in each pixel



the window, one record per pixel

one pixel's record

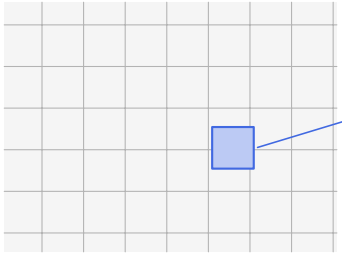
position $\mathbf{x}$, depth

normal $\mathbf{n}$

albedo R, material

- G for geometry: the first hit, stored per pixel.
- Enough to aim rays from there, and to shade.

The G-buffer: what the first hit leaves in each pixel



the window, one record per pixel

one pixel's record

position $\mathbf{x}$, depth

normal $\mathbf{n}$

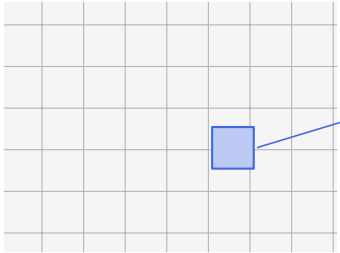
albedo R, material

motion since last frame

(part two)

- G for geometry: the first hit, stored per pixel.
- Enough to aim rays from there, and to shade.

The G-buffer: what the first hit leaves in each pixel



the window, one record per pixel

one pixel's record

position $\mathbf{x}$, depth

normal $\mathbf{n}$

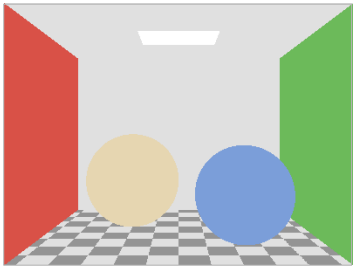
albedo R, material

motion since last frame

(part two)

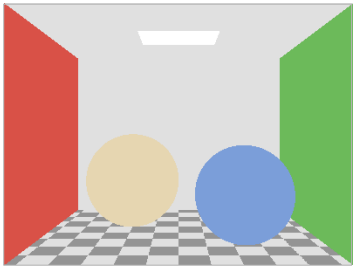
- G for geometry: the first hit, stored per pixel.
- Enough to aim rays from there, and to shade.
- Exact and free of noise: the filter's guide.

Rendered: our room's G-buffer

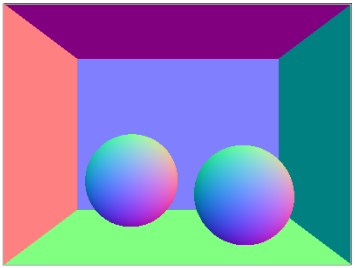


albedo R

Rendered: our room's G-buffer

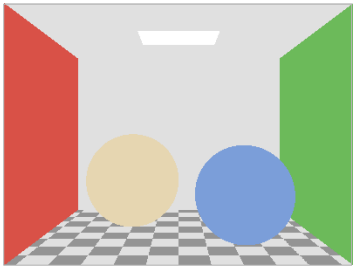


albedo R

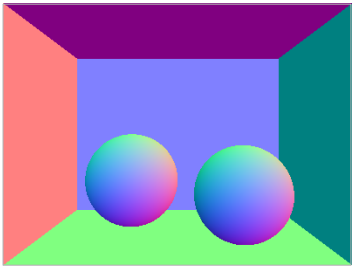


normal: x red, y green, z blue

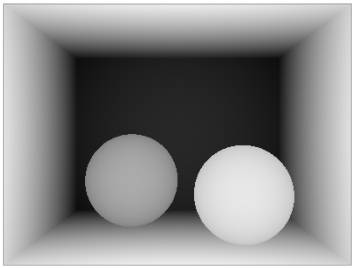
Rendered: our room's G-buffer



albedo R

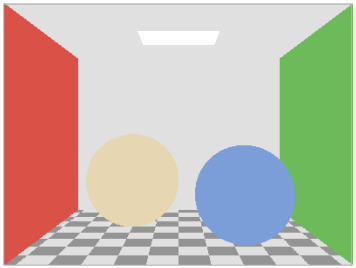


normal: x red, y green, z blue

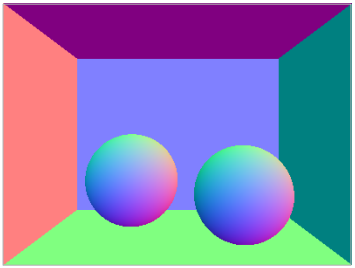


depth: near light, far dark

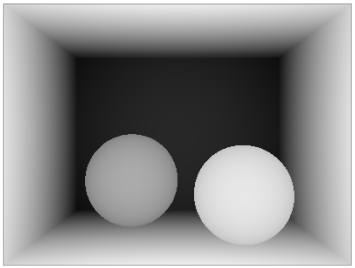
Rendered: our room's G-buffer



albedo R



normal: x red, y green, z blue



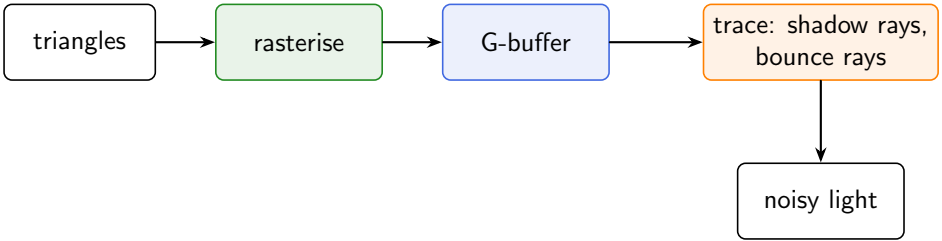
depth: near light, far dark

- No light in any of them: only what is where.

The hybrid frame: rasterise first, trace after

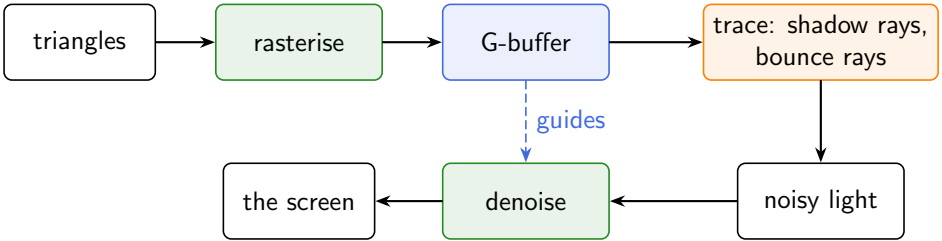


The hybrid frame: rasterise first, trace after



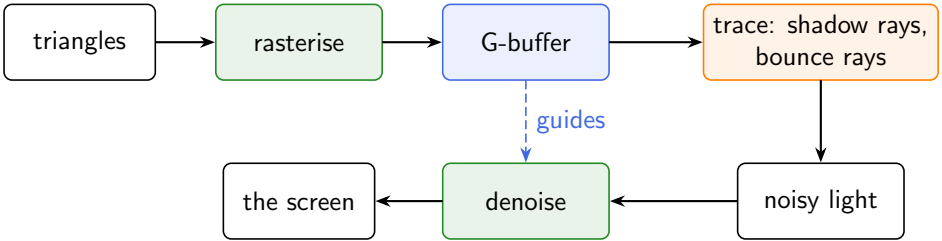
- Rays start from the G-buffer: the first ray is never traced.

The hybrid frame: rasterise first, trace after



- Rays start from the G-buffer: the first ray is never traced.
- The G-buffer guides the filter, too.

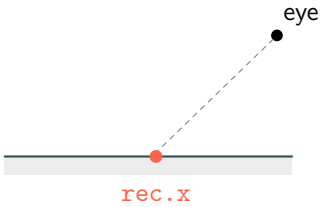
The hybrid frame: rasterise first, trace after



- Rays start from the G-buffer: the first ray is never traced.
- The G-buffer guides the filter, too.
- First games traced one effect: shadows, or reflections. Now, whole paths.

The path from the G-buffer, in Python

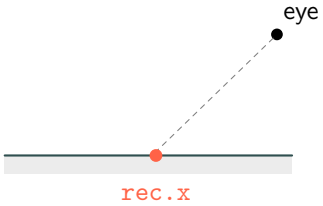
```
for (i, j), rec in gbuffer.items():
```



- No first ray: `rec` comes from the G-buffer (the dashed line is not traced).

The path from the G-buffer, in Python

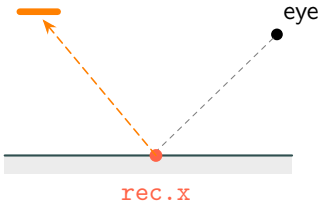
```
for (i, j), rec in gbuffer.items():
    if rec.glow:  image[j][i] = rec.glow; continue
```



- No first ray: `rec` comes from the G-buffer (the dashed line is not traced).

The path from the G-buffer, in Python

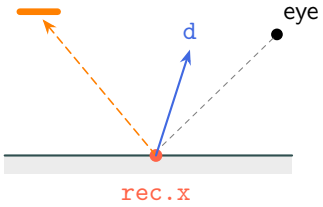
```
for (i, j), rec in gbuffer.items():
    if rec.glow:  image[j][i] = rec.glow; continue
    c = Vec3(0, 0, 0)
    for light in lights:
        c = c + light.illuminate(rec)
```



- No first ray: `rec` comes from the G-buffer (the dashed line is not traced).

The path from the G-buffer, in Python

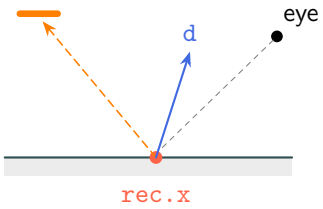
```
for (i, j), rec in gbuffer.items():
    if rec.glow:  image[j][i] = rec.glow; continue
    c = Vec3(0, 0, 0)
    for light in lights:
        c = c + light.illuminate(rec)
    d = rec.normal + random_unit_vector()
    image[j][i] = c + rec.albedo * trace(Ray(rec.x, d), depth - 1)
```



- No first ray: `rec` comes from the G-buffer (the dashed line is not traced).
- From the first bounce on, lecture 20's `trace`, unchanged.

The path from the G-buffer, in Python

```
for (i, j), rec in gbuffer.items():
    if rec.glow:  image[j][i] = rec.glow; continue
    c = Vec3(0, 0, 0)
    for light in lights:
        c = c + light.illuminate(rec)
    d = rec.normal + random_unit_vector()
    image[j][i] = c + rec.albedo * trace(Ray(rec.x, d), depth - 1)
image = denoise(image, gbuffer)
```



- No first ray: `rec` comes from the G-buffer (the dashed line is not traced).
- From the first bounce on, lecture 20's `trace`, unchanged.
- Then one call cleans the whole picture: part two.

Recap: one frame, 16.7 ms

One frame at 60 frames a second: 16.7 ms (an example budget)

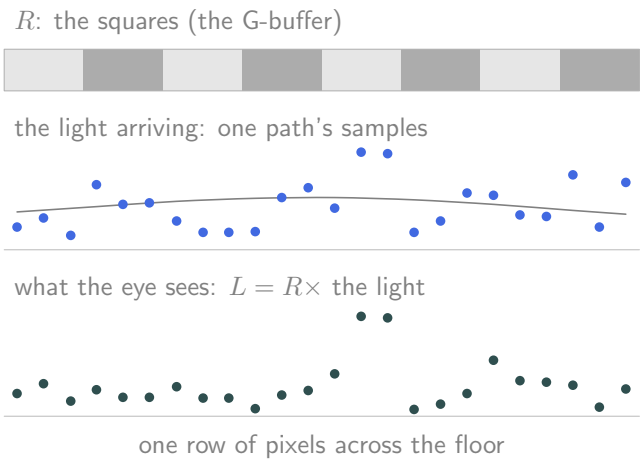


1. Rasterise the first hit into a G-buffer; trace one path per pixel

Denoising

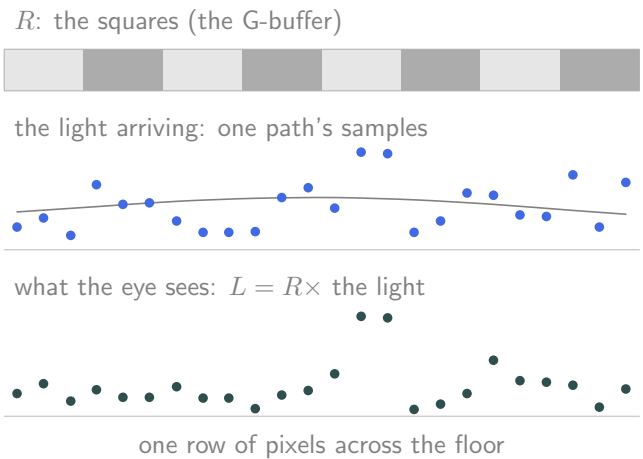
One noisy path per pixel, a clean picture

Keep the texture out of the filter



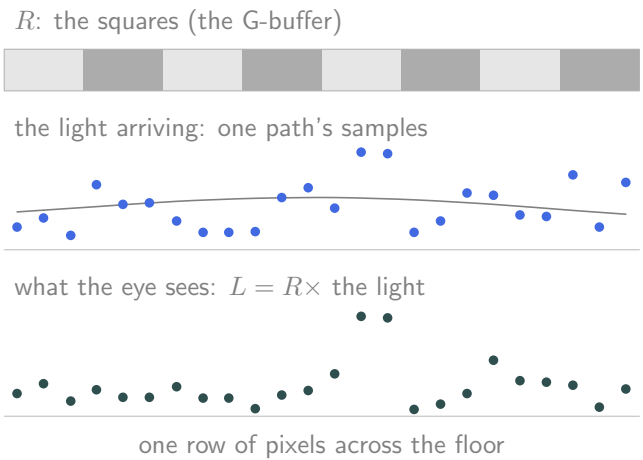
- What the eye sees: R times the light arriving.

Keep the texture out of the filter



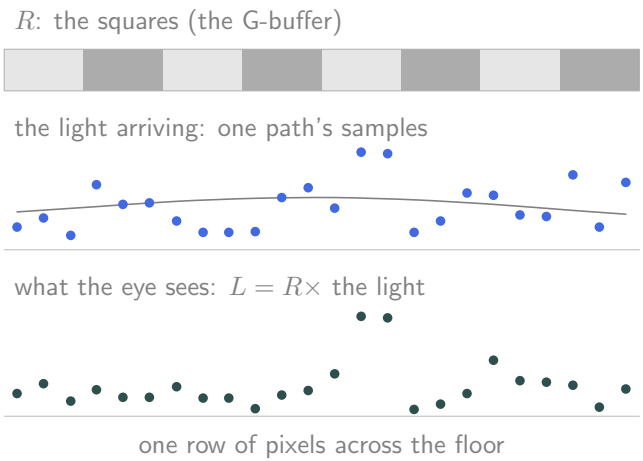
- What the eye sees: R times the light arriving.
- The noise is all in the light. R is exact.

Keep the texture out of the filter



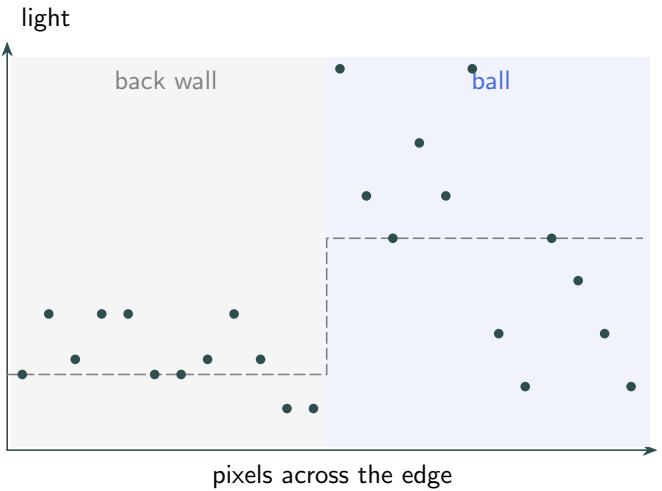
- What the eye sees: R times the light arriving.
- The noise is all in the light. R is exact.
- So divide by R, filter the light alone, multiply R back.

Keep the texture out of the filter



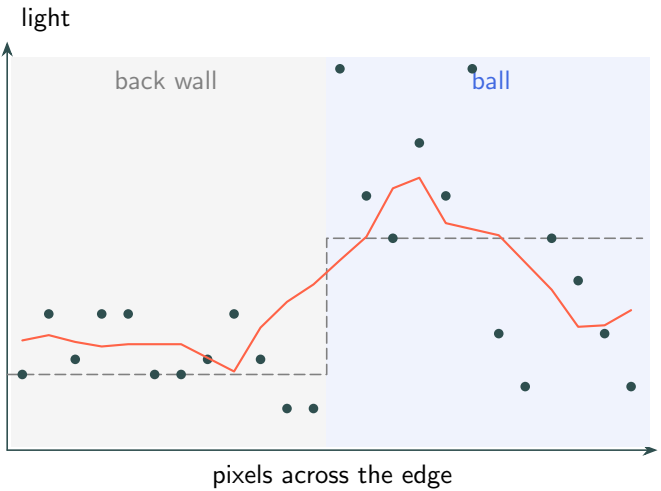
- What the eye sees: R times the light arriving.
- The noise is all in the light. R is exact.
- So divide by R, filter the light alone, multiply R back.
- The squares stay sharp; only the light is smoothed.

A plain blur: the noise goes, and so does the edge



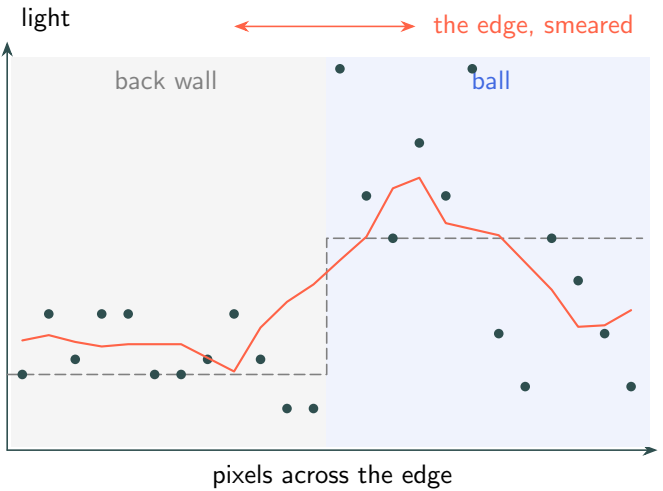
- Dots: one path per pixel.
  Dashed: the truth.

A plain blur: the noise goes, and so does the edge



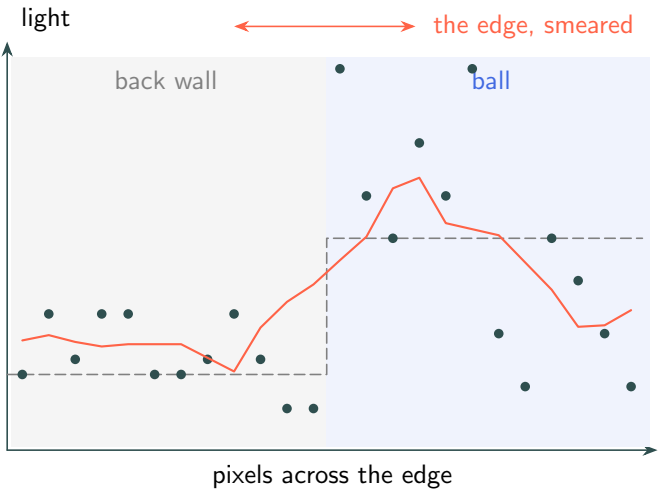
- Dots: one path per pixel. Dashed: the truth.
- Average each pixel with 3 on each side: calmer.

A plain blur: the noise goes, and so does the edge



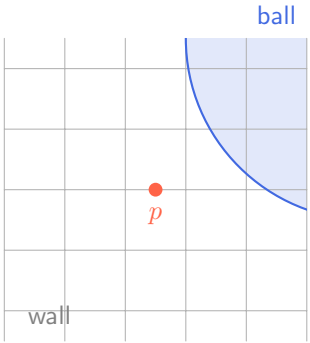
- Dots: one path per pixel. Dashed: the truth.
- Average each pixel with 3 on each side: calmer.
- But it mixes wall into ball, and ball into wall.

A plain blur: the noise goes, and so does the edge



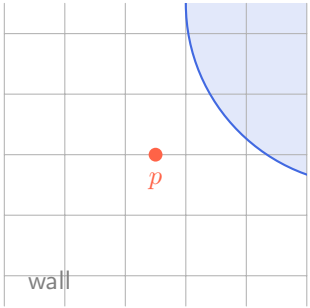
- Dots: one path per pixel. Dashed: the truth.
- Average each pixel with 3 on each side: calmer.
- But it mixes wall into ball, and ball into wall.
- The fix: only average a neighbour on the same surface.

Ask the G-buffer before averaging



p's 5×5 neighbours

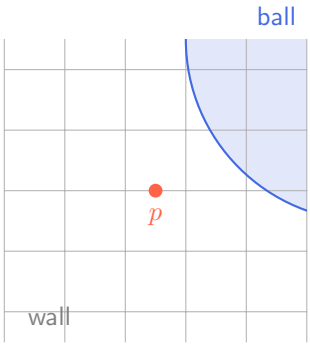
Ask the G-buffer before averaging



p's 5 × 5 neighbours

$w = h \cdot w_z \cdot w_n$ h: the kernel, $\frac{1}{16}, \frac{1}{4}, \frac{3}{8}, \frac{1}{4}, \frac{1}{16}$ each way

Ask the G-buffer before averaging



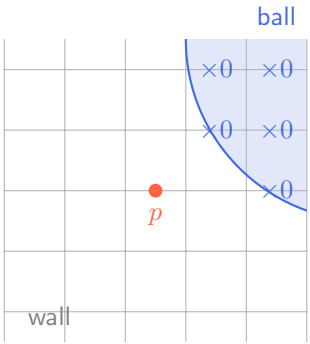
p's 5 × 5 neighbours

$w = h \cdot w_z \cdot w_n$ h: the kernel, $\frac{1}{16}, \frac{1}{4}, \frac{3}{8}, \frac{1}{4}, \frac{1}{16}$ each way

$w_z = e^{-|z_p - z_q|/\sigma_z}$ depth z: as far from the eye?

$w_n = \max(0,\ \mathbf{n}_p \cdot \mathbf{n}_q)^{128}$ normal: facing the same way?

Ask the G-buffer before averaging



p's 5 × 5 neighbours

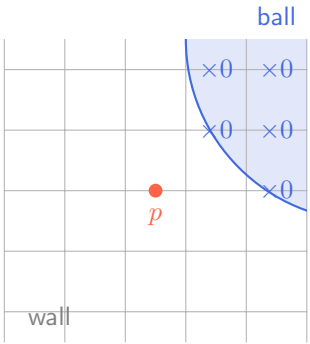
$w = h \cdot w_z \cdot w_n$ h: the kernel, $\frac{1}{16}, \frac{1}{4}, \frac{3}{8}, \frac{1}{4}, \frac{1}{16}$ each way

$w_z = e^{-|z_p - z_q|/\sigma_z}$ depth z: as far from the eye?

$w_n = \max(0,\ \mathbf{n}_p \cdot \mathbf{n}_q)^{128}$ normal: facing the same way?

- On the ball: wrong depth, wrong normal, weight near 0.

Ask the G-buffer before averaging



p's 5×5 neighbours

$w = h \cdot w_z \cdot w_n$ h: the kernel, $\frac{1}{16}, \frac{1}{4}, \frac{3}{8}, \frac{1}{4}, \frac{1}{16}$ each way

$w_z = e^{-|z_p - z_q|/\sigma_z}$ depth z: as far from the eye?

$w_n = \max(0,\ \mathbf{n}_p \cdot \mathbf{n}_q)^{128}$ normal: facing the same way?

- On the ball: wrong depth, wrong normal, weight near 0.
- Then divide by the sum of the weights.
- The edge stays: the G-buffer has it exactly.

Worked: one pixel at the ball's edge

Try first: five pixels in a row, p in the middle. $\sigma_z = 0.1$.

	q_{-2}	q_{-1}	p	q_1	q_2
on	wall	wall	wall	ball	ball
light (noisy)	1.6	0.4	1.3	3.0	2.6
kernel h	$\frac{1}{16}$	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{1}{4}$	$\frac{1}{16}$
depth z	5.21	5.21	5.20	4.10	4.10
$\mathbf{n}_p \cdot \mathbf{n}_q$	1	1	1	0.6	0.6

Worked: one pixel at the ball's edge

Try first: five pixels in a row, p in the middle. $\sigma_z = 0.1$.

	q_{-2}	q_{-1}	p	q_1	q_2
on	wall	wall	wall	ball	ball
light (noisy)	1.6	0.4	1.3	3.0	2.6
kernel h	$\frac{1}{16}$	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{1}{4}$	$\frac{1}{16}$
depth z	5.21	5.21	5.20	4.10	4.10
$\mathbf{n}_p \cdot \mathbf{n}_q$	1	1	1	0.6	0.6
$w_z\, w_n$	0.905	0.905	1	≈ 0	≈ 0

Worked: one pixel at the ball's edge

Try first: five pixels in a row, p in the middle. $\sigma_z = 0.1$.

	q_{-2}	q_{-1}	p	q_1	q_2
on	wall	wall	wall	ball	ball
light (noisy)	1.6	0.4	1.3	3.0	2.6
kernel h	$\frac{1}{16}$	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{1}{4}$	$\frac{1}{16}$
depth z	5.21	5.21	5.20	4.10	4.10
$\mathbf{n}_p \cdot \mathbf{n}_q$	1	1	1	0.6	0.6
$w_z\, w_n$	0.905	0.905	1	≈ 0	≈ 0

plain: $\frac{1}{16}1.6 + \frac{1}{4}0.4 + \frac{3}{8}1.3 + \frac{1}{4}3.0 + \frac{1}{16}2.6 = \mathbf{1.60}$

Worked: one pixel at the ball's edge

Try first: five pixels in a row, p in the middle. $\sigma_z = 0.1$.

	q_{-2}	q_{-1}	p	q_1	q_2
on	wall	wall	wall	ball	ball
light (noisy)	1.6	0.4	1.3	3.0	2.6
kernel h	$\frac{1}{16}$	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{1}{4}$	$\frac{1}{16}$
depth z	5.21	5.21	5.20	4.10	4.10
$\mathbf{n}_p \cdot \mathbf{n}_q$	1	1	1	0.6	0.6
$w_z\, w_n$	0.905	0.905	1	≈ 0	≈ 0

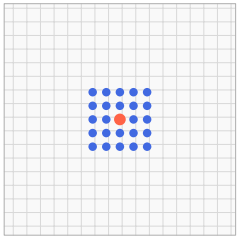
plain: $\frac{1}{16}1.6 + \frac{1}{4}0.4 + \frac{3}{8}1.3 + \frac{1}{4}3.0 + \frac{1}{16}2.6 = \mathbf{1.60}$

with the tests: $\dfrac{0.0905 + 0.0905 + 0.4875}{0.0566 + 0.2263 + 0.375} = \mathbf{1.02}$

the wall's truth: 1.0

À-trous: a wide filter from five small ones

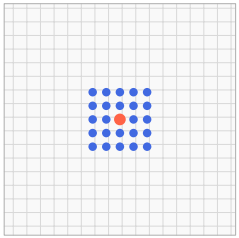
step 1



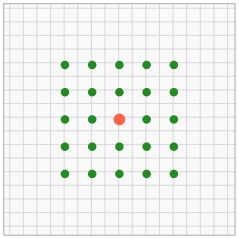
- Each pass: the same 25 taps, spread wider; each smooths the last one's output.

À-trous: a wide filter from five small ones

step 1

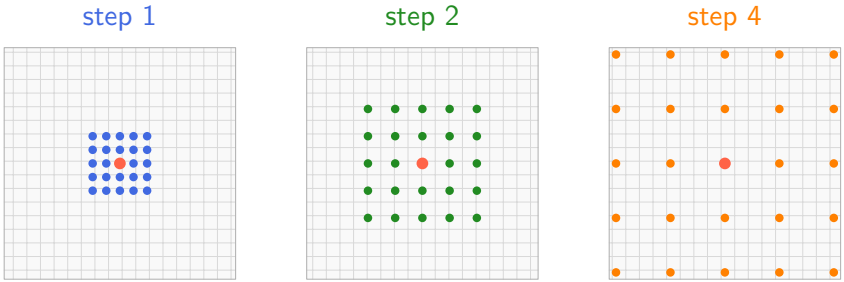


step 2



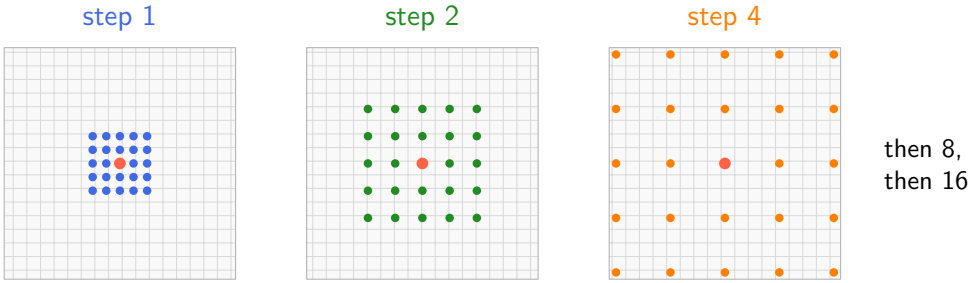
- Each pass: the same 25 taps, spread wider; each smooths the last one's output.

À-trous: a wide filter from five small ones



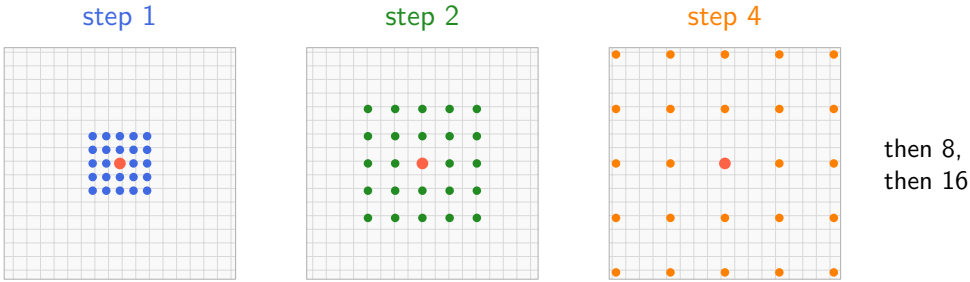
- Each pass: the same 25 taps, spread wider; each smooths the last one's output.

À-trous: a wide filter from five small ones



- Each pass: the same 25 taps, spread wider; each smooths the last one's output.
- Steps 1, 2, 4, 8, 16: they reach 62 pixels each way, 125 across.

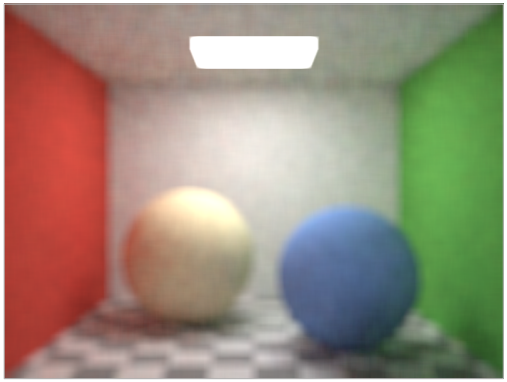
À-trous: a wide filter from five small ones



- Each pass: the same 25 taps, spread wider; each smooths the last one's output.
- Steps 1, 2, 4, 8, 16: they reach 62 pixels each way, 125 across.
- $5 \times 25 = 125$ taps a pixel, not $125 \times 125 = 15{,}625$.

Rendered: a plain blur against the edge-stopping filter

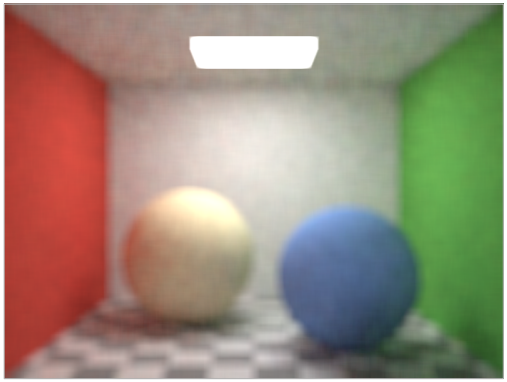
The same frame, one path per pixel (C4, left), filtered alone



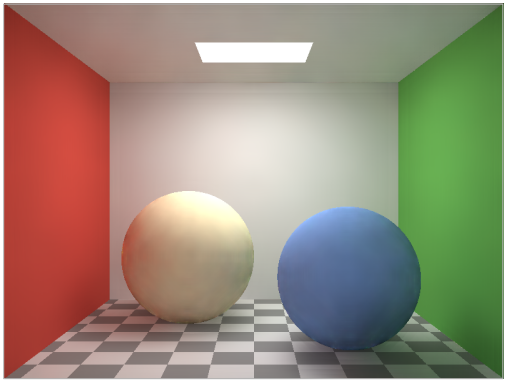
a 21×21 box, no G-buffer

Rendered: a plain blur against the edge-stopping filter

The same frame, one path per pixel (C4, left), filtered alone



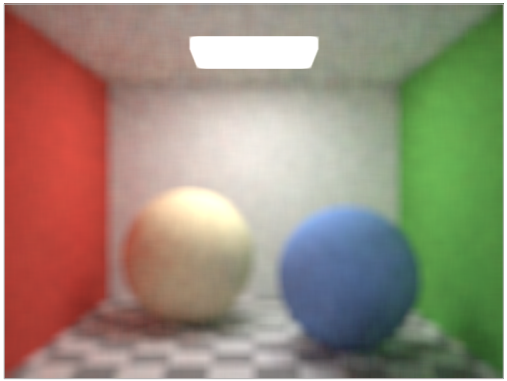
a 21×21 box, no G-buffer



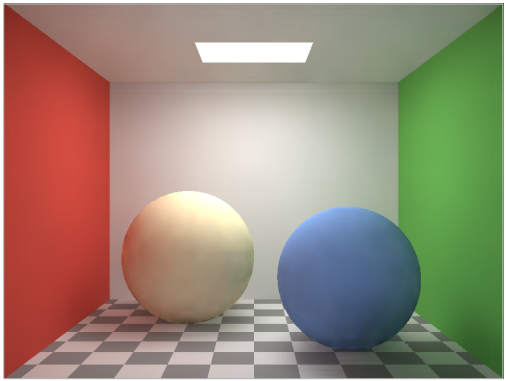
five à-trous passes, depth and normal tests

Rendered: a plain blur against the edge-stopping filter

The same frame, one path per pixel (C4, left), filtered alone



a 21×21 box, no G-buffer



five à-trous passes, depth and normal tests

- Edges and squares kept; blotches left where one frame had too few samples.

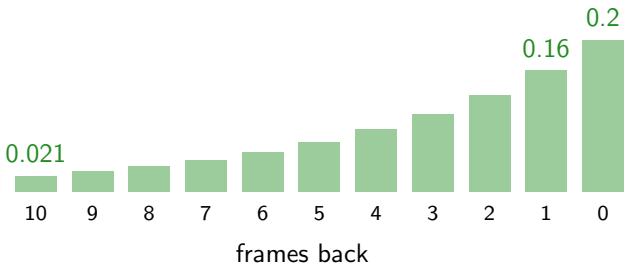
Average over time: blend into the history

history $\leftarrow 0.8\times$ history $+ 0.2\times$ this frame

- The camera hardly moves in 16.7 ms: reuse the last frames.

Average over time: blend into the history

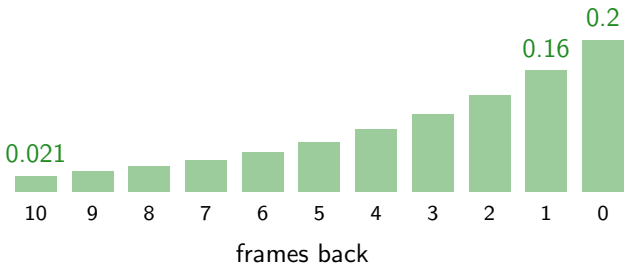
history $\leftarrow 0.8\times$ history $+ 0.2\times$ this frame



- The camera hardly moves in 16.7 ms: reuse the last frames.
- Old frames fade: 10 frames back, 2%.

Average over time: blend into the history

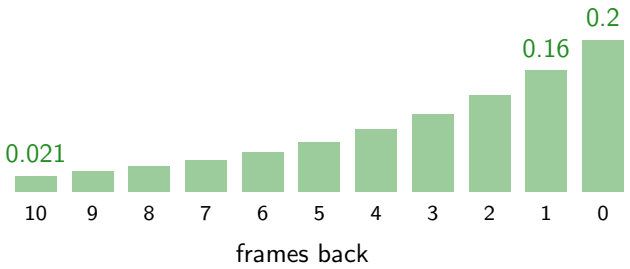
history $\leftarrow 0.8\times$ history $+ 0.2\times$ this frame



- The camera hardly moves in 16.7 ms: reuse the last frames.
- Old frames fade: 10 frames back, 2%.
- In the long run, like 9 frames averaged: noise $\div$ 3.

Average over time: blend into the history

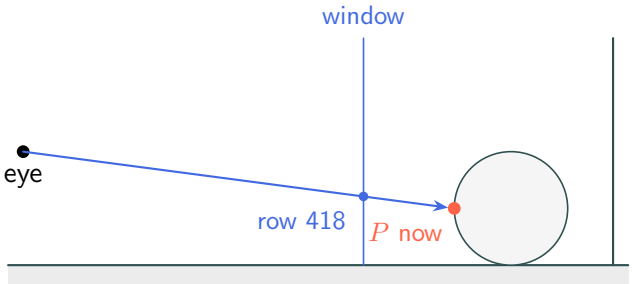
history $\leftarrow 0.8\times$ history $+ 0.2\times$ this frame



- The camera hardly moves in 16.7 ms: reuse the last frames.
- Old frames fade: 10 frames back, 2%.
- In the long run, like 9 frames averaged: noise $\div$ 3.
- The price: a change of light shows only slowly.

Reprojection: where was this point last frame?

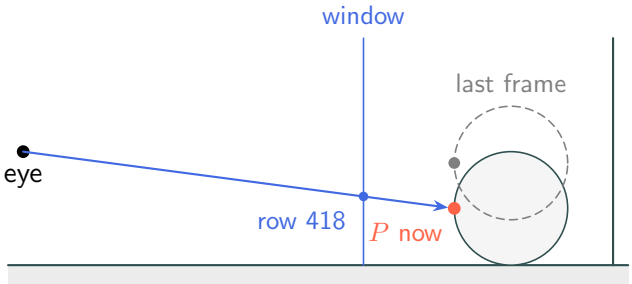
Side view; the fall drawn much larger than it is



- Each pixel follows its point back one frame.

Reprojection: where was this point last frame?

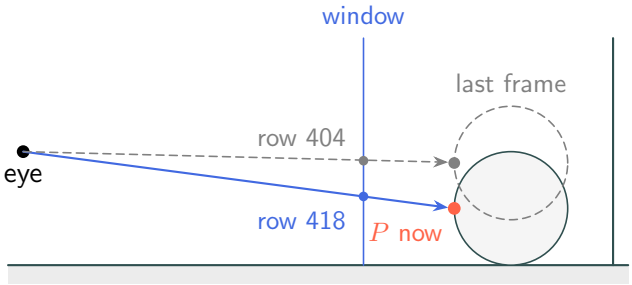
Side view; the fall drawn much larger than it is



- Each pixel follows its point back one frame.
- The game knows how everything moved: the G-buffer's motion.

Reprojection: where was this point last frame?

Side view; the fall drawn much larger than it is

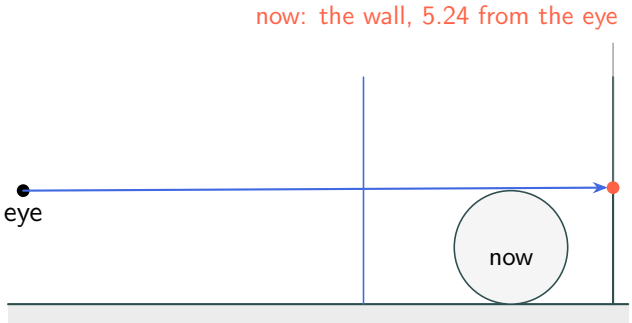


- Each pixel follows its point back one frame.
- The game knows how everything moved: the G-buffer's motion.
- Project the old point: that is where its history is.

$P = (-0.5, 0.5, -0.8)$: row 418. Last frame 0.06 higher: row 404. Read the history 14 rows up.

Disocclusion: a history that belongs to something else

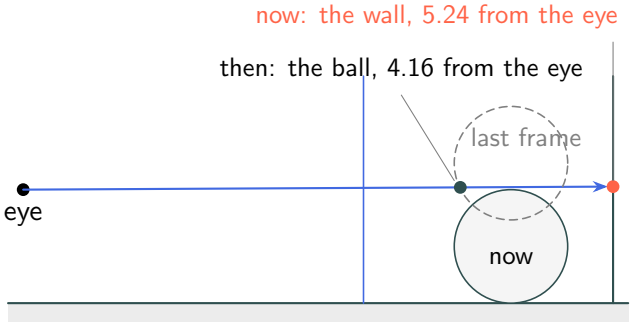
C17's side view; the fall drawn much larger than it is



- Just above the falling ball, the wall comes into view.

Disocclusion: a history that belongs to something else

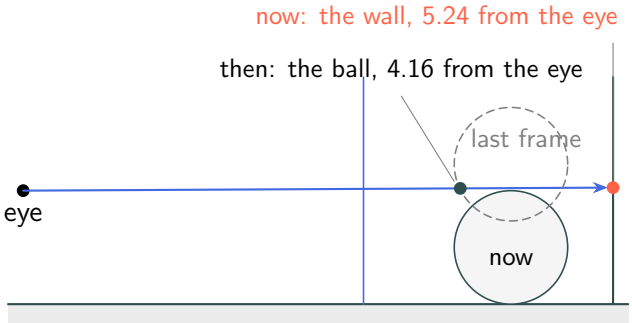
C17's side view; the fall drawn much larger than it is



- Just above the falling ball, the wall comes into view.
- Its history pixel saw the ball: the wrong light.

Disocclusion: a history that belongs to something else

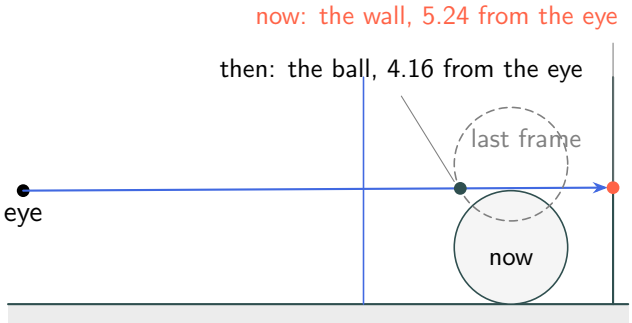
C17's side view; the fall drawn much larger than it is



- Just above the falling ball, the wall comes into view.
- Its history pixel saw the ball: the wrong light.
- Compare depth, normal, object: any mismatch, drop the history.

Disocclusion: a history that belongs to something else

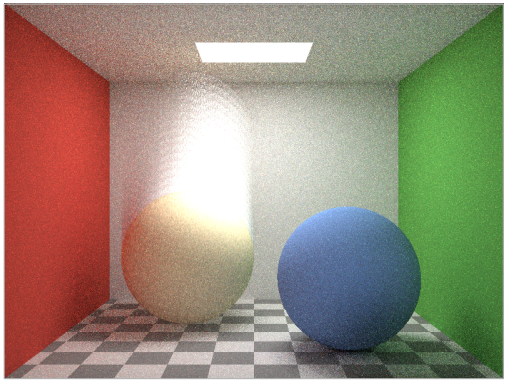
C17's side view; the fall drawn much larger than it is



- Just above the falling ball, the wall comes into view.
- Its history pixel saw the ball: the wrong light.
- Compare depth, normal, object: any mismatch, drop the history.
- The pixel starts again, noisy for a few frames.

Rendered: history kept against history dropped

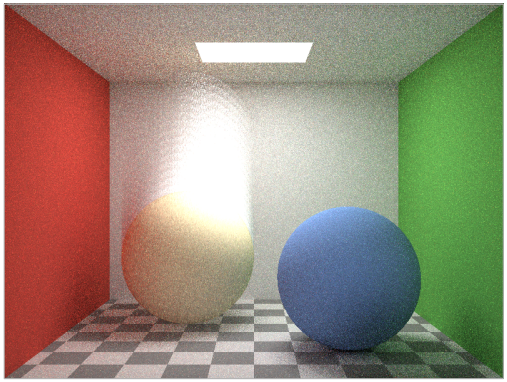
16 frames of the ball falling, 20% new light each frame, no filter



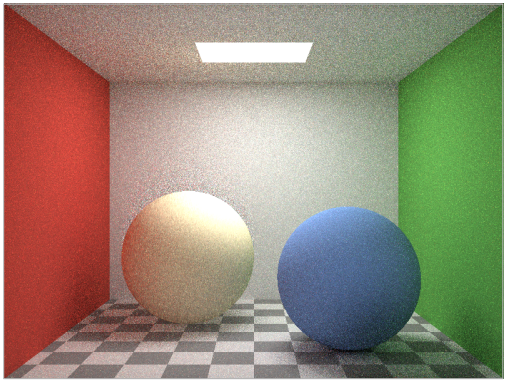
history always kept

Rendered: history kept against history dropped

16 frames of the ball falling, 20% new light each frame, no filter



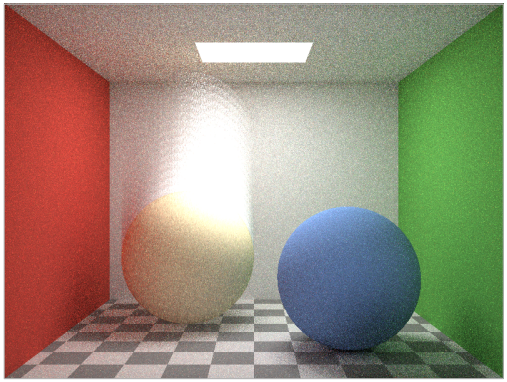
history always kept



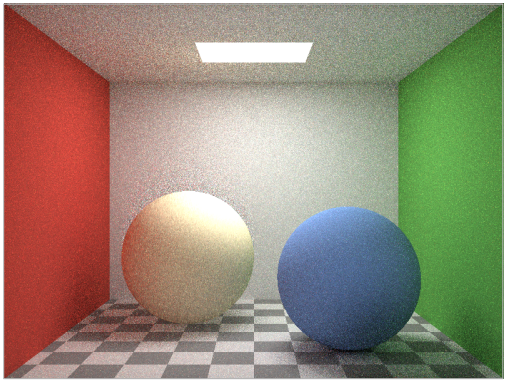
a mismatched history dropped

Rendered: history kept against history dropped

16 frames of the ball falling, 20% new light each frame, no filter



history always kept



a mismatched history dropped

- Kept: a bright ghost above the ball. Dropped: no ghost, a noisy rim.

SVGF: let the noise decide how hard to filter

The history also keeps the brightness l and l^2, averaged the same way:

$$\text{variance} = \text{average of } l^2 - (\text{average of } l)^2$$ how noisy this pixel is

SVGF: let the noise decide how hard to filter

The history also keeps the brightness l and l^2, averaged the same way:

$$\text{variance} = \text{average of } l^2 - (\text{average of } l)^2$$ how noisy this pixel is

A third test in every pass: $w_l = \exp\left(-\frac{|l_p - l_q|}{4\sqrt{\text{variance}_p}}\right)$

SVGF: let the noise decide how hard to filter

The history also keeps the brightness l and l^2, averaged the same way:

$$\text{variance} = \text{average of } l^2 - (\text{average of } l)^2$$ how noisy this pixel is

A third test in every pass: $w_l = \exp\left(-\dfrac{|l_p - l_q|}{4\sqrt{\text{variance}_p}}\right)$

A neighbour 0.5 brighter: variance 0.01: $w_l = e^{-0.5/0.4} = 0.29$; variance 0.25: $e^{-0.5/2} = 0.78$

SVGF: let the noise decide how hard to filter

The history also keeps the brightness l and l^2, averaged the same way:

$$\text{variance} = \text{average of } l^2 - (\text{average of } l)^2$$ how noisy this pixel is

A third test in every pass: $w_l = \exp\left(-\dfrac{|l_p - l_q|}{4\sqrt{\text{variance}_p}}\right)$

A neighbour 0.5 brighter: variance 0.01: $w_l = e^{-0.5/0.4} = 0.29$; variance 0.25: $e^{-0.5/2} = 0.78$



C16–C18

SVGF: let the noise decide how hard to filter

The history also keeps the brightness l and l^2, averaged the same way:

$$\text{variance} = \text{average of } l^2 - (\text{average of } l)^2$$ how noisy this pixel is

A third test in every pass: $w_l = \exp\left(-\dfrac{|l_p - l_q|}{4\sqrt{\text{variance}_p}}\right)$

A neighbour 0.5 brighter: variance 0.01: $w_l = e^{-0.5/0.4} = 0.29$; variance 0.25: $e^{-0.5/2} = 0.78$

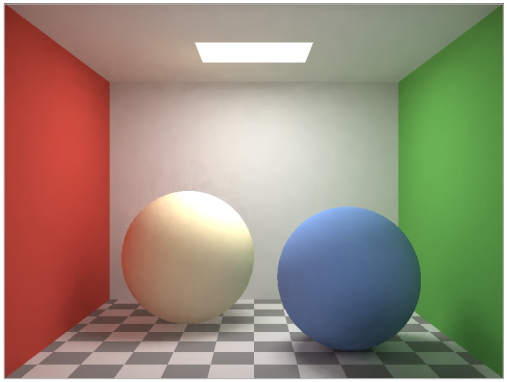


C16–C18

- Spatiotemporal variance-guided filtering: Schied et al., 2017.

Rendered: SVGF at one path per pixel, against 1024

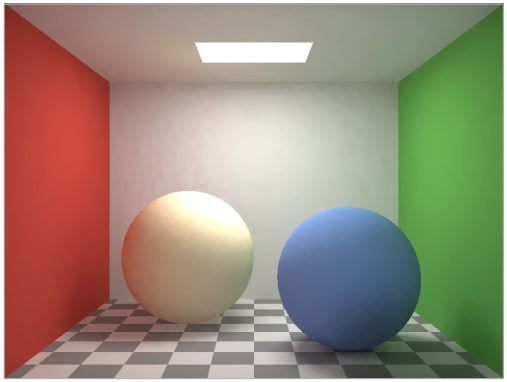
The 16th frame of the falling ball: C19's right picture through SVGF's filter



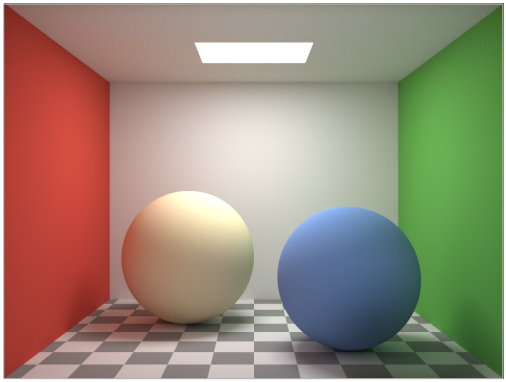
SVGF: one path per pixel per frame

Rendered: SVGF at one path per pixel, against 1024

The 16th frame of the falling ball: C19's right picture through SVGF's filter



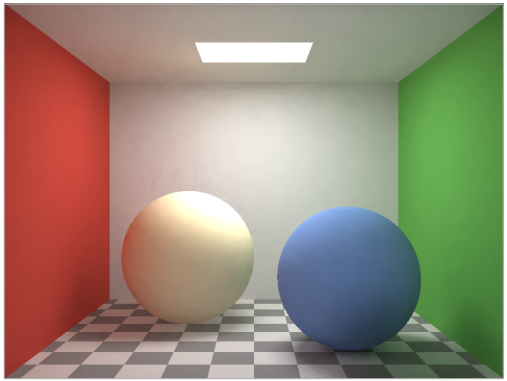
SVGF: one path per pixel per frame



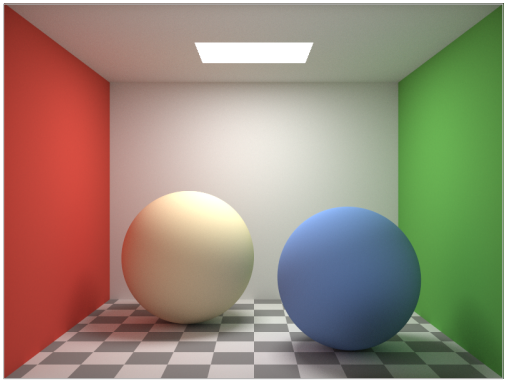
1024 paths per pixel

Rendered: SVGF at one path per pixel, against 1024

The 16th frame of the falling ball: C19's right picture through SVGF's filter



SVGF: one path per pixel per frame



1024 paths per pixel

- Close, at a thousandth of the paths; softer where light changes fast.

Quick check 1: budget and filters

$w_n = \max(0, \mathbf{n}_p \cdot \mathbf{n}_q)^{128}$; **history** $\leftarrow 0.8$ **history** $+0.2$ **new**

1. A game runs at 120 frames a second. Milliseconds per frame?
2. Neighbours' normals 10° apart ($\cos = 0.985$): their w_n?
3. What share of the history comes from the frame 3 back?
4. Why filter L/R and not L?

- The G-buffer guards the edges; the history supplies the samples.

Quick check 1: budget and filters

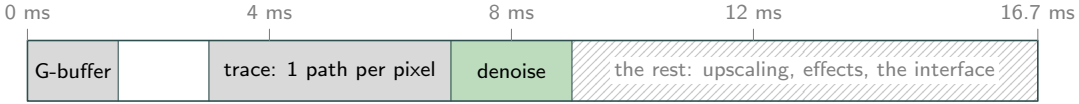
$w_n = \max(0, \mathbf{n}_p \cdot \mathbf{n}_q)^{128}$; **history** $\leftarrow 0.8\,$**history** $+0.2\,$**new**

1. A game runs at 120 frames a second. Milliseconds per frame? 8.3 ms
2. Neighbours' normals 10° apart ($\cos = 0.985$): their w_n? 0.14
3. What share of the history comes from the frame 3 back? $0.2 \times 0.8^3 = 0.10$
4. Why filter L/R and not L? keeps R sharp

- The G-buffer guards the edges; the history supplies the samples.

Recap: one frame, 16.7 ms

One frame at 60 frames a second: 16.7 ms (an example budget)

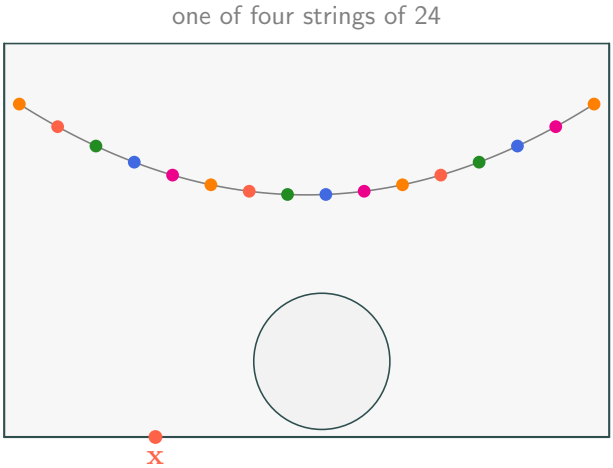


1. Rasterise the first hit into a G-buffer; trace one path per pixel
2. Denoise: G-buffer weights, history by reprojection, SVGF

Many lamps: ReSTIR

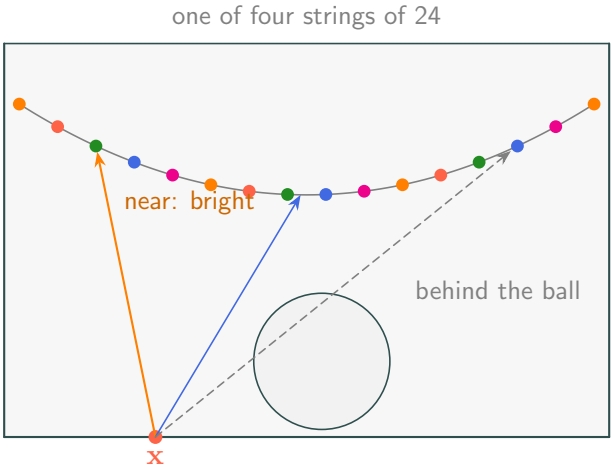
Reservoirs, and picks shared between pixels

A room lit by 96 lamps, and one shadow ray



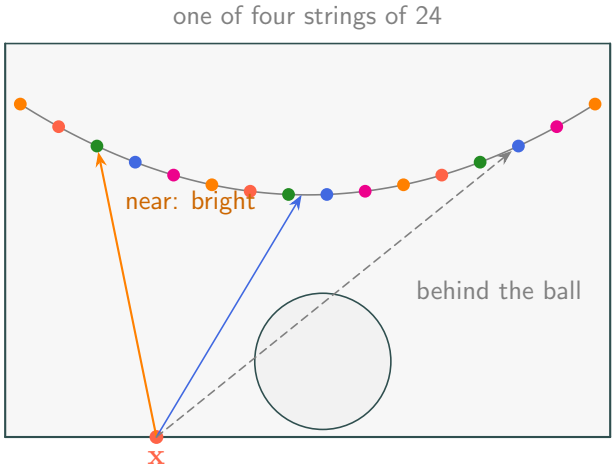
- Our night room: 96 small lamps on four strings.

A room lit by 96 lamps, and one shadow ray



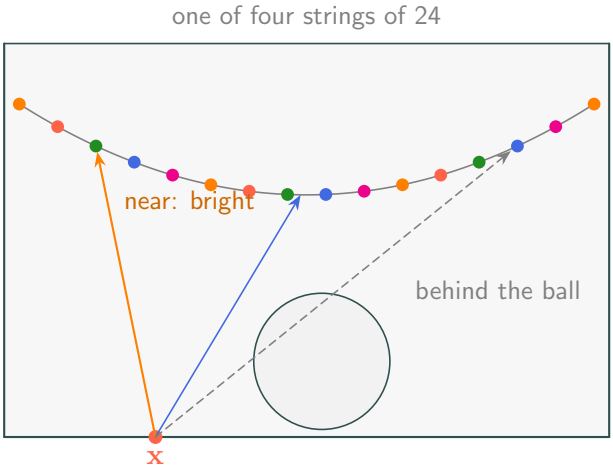
- Our night room: 96 small lamps on four strings.
- Near lamps give much, far ones little, hidden ones none.

A room lit by 96 lamps, and one shadow ray



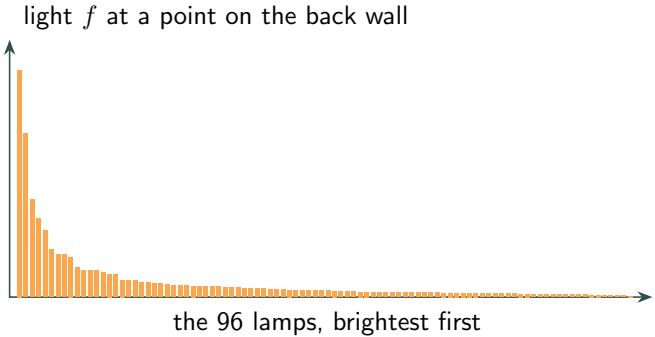
- Our night room: 96 small lamps on four strings.
- Near lamps give much, far ones little, hidden ones none.
- Budget: one shadow ray per pixel. To which lamp?

A room lit by 96 lamps, and one shadow ray

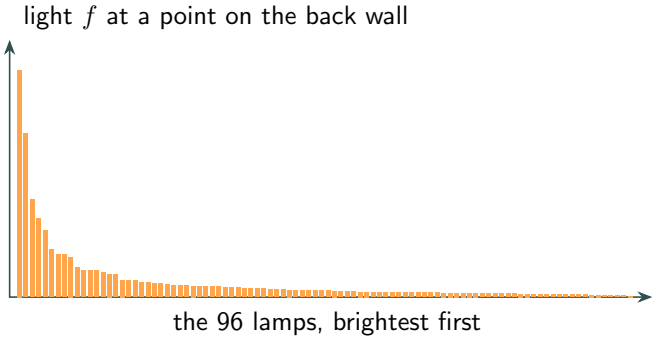


- Our night room: 96 small lamps on four strings.
- Near lamps give much, far ones little, hidden ones none.
- Budget: one shadow ray per pixel. To which lamp?
- Extra 1 picked a point on one lamp. Now: pick the lamp.

Pick a lamp by how much light it gives



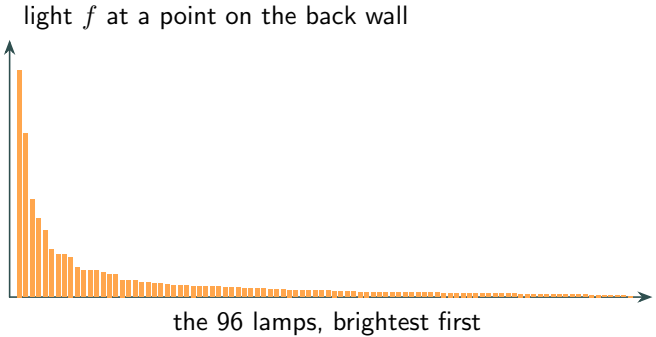
Pick a lamp by how much light it gives



- Picked evenly: usually too dim, now and then far too bright.

Evenly: $96 \times f$ of the lamp picked 0.07 to 14.6 times the truth

Pick a lamp by how much light it gives

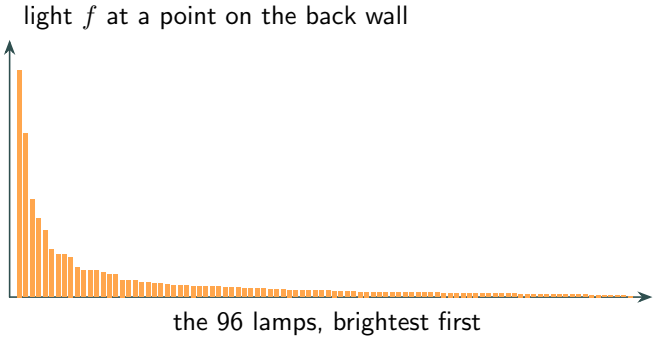


- Picked evenly: usually too dim, now and then far too bright.
- Picked in proportion to its light: no noise at all.

Evenly: $96 \times f$ of the lamp picked 0.07 to 14.6 times the truth

In proportion to f: $f/p = f/(f/\sum f) = \sum f$ the truth, every time

Pick a lamp by how much light it gives



- Picked evenly: usually too dim, now and then far too bright.
- Picked in proportion to its light: no noise at all.

- But that needs every lamp's light first: 96 shadow rays.

Evenly: $96 \times f$ of the lamp picked 0.07 to 14.6 times the truth

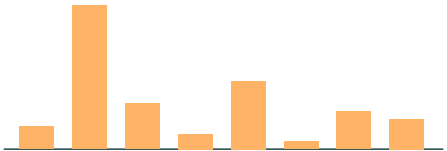
In proportion to f: $f/p = f/(f/\sum f) = \sum f$ the truth, every time

Resampled importance sampling: draw many, keep one

1. Draw M candidate lamps evenly ($p = 1/N$).

Resampled importance sampling: draw many, keep one

1. Draw M candidate lamps evenly ($p = 1/N$).
2. Weigh each by $w = \hat{p}/p$. $\hat{p}$: its light *without* the shadow ray.

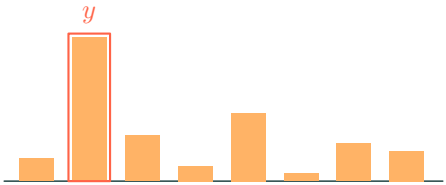


$M = 8$ candidates, their w

- $\hat{p}$ (p-hat) costs no ray: $I \cos /r^2$.

Resampled importance sampling: draw many, keep one

1. Draw M candidate lamps evenly ($p = 1/N$).
2. Weigh each by $w = \hat{p}/p$. $\hat{p}$: its light *without* the shadow ray.
3. Keep one, y, with chance $w/\sum w$.

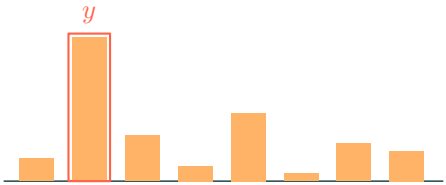


$M = 8$ candidates, their w

- $\hat{p}$ (p-hat) costs no ray: $I \cos /r^2$.

Resampled importance sampling: draw many, keep one

1. Draw M candidate lamps evenly ($p = 1/N$).
2. Weigh each by $w = \hat{p}/p$. $\hat{p}$: its light *without* the shadow ray.
3. Keep one, y, with chance $w/\sum w$.
4. Estimate: $f(y) \times W$, $W = \dfrac{\sum w}{M\,\hat{p}(y)}$. One shadow ray, to y.

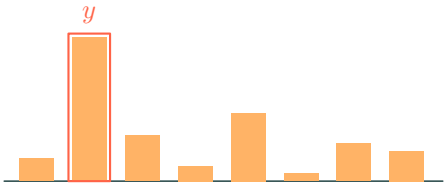


$M = 8$ candidates, their w

- $\hat{p}$ (p-hat) costs no ray: $I \cos /r^2$.
- W undoes the picking: right on average.

Resampled importance sampling: draw many, keep one

1. Draw M candidate lamps evenly ($p = 1/N$).
2. Weigh each by $w = \hat{p}/p$. $\hat{p}$: its light *without* the shadow ray.
3. Keep one, y, with chance $w/\sum w$.
4. Estimate: $f(y) \times W$, $W = \dfrac{\sum w}{M\,\hat{p}(y)}$. One shadow ray, to y.



$M = 8$ candidates, their w

- $\hat{p}$ (p-hat) costs no ray: $I \cos /r^2$.
- W undoes the picking: right on average.
- Many cheap guesses, one expensive ray.

Worked: four candidates, one kept

Try first: $N = 96$ lamps, $M = 4$ candidates. Fill in w, the chance of being kept, W, and $\hat{p}(y)\,W$.

	A	B	C	D
$\hat{p}$ (light, no shadow ray)	0.2	3.0	0.8	0

Worked: four candidates, one kept

Try first: $N = 96$ lamps, $M = 4$ candidates. Fill in w, the chance of being kept, W, and $\hat{p}(y)\,W$.

	A	B	C	D	
$\hat{p}$ (light, no shadow ray)	0.2	3.0	0.8	0	
$w = 96\,\hat{p}$	19.2	288	76.8	0	$\sum w = 384$

Worked: four candidates, one kept

Try first: $N = 96$ lamps, $M = 4$ candidates. Fill in w, the chance of being kept, W, and $\hat{p}(y)\,W$.

	A	B	C	D	
$\hat{p}$ (light, no shadow ray)	0.2	3.0	0.8	0	
$w = 96\,\hat{p}$	19.2	288	76.8	0	$\sum w = 384$
chance kept, $w/384$	0.05	0.75	0.20	0	

Worked: four candidates, one kept

Try first: $N = 96$ lamps, $M = 4$ candidates. Fill in w, the chance of being kept, W, and $\hat{p}(y)\,W$.

	A	B	C	D
$\hat{p}$ (light, no shadow ray)	0.2	3.0	0.8	0
$w = 96\,\hat{p}$	19.2	288	76.8	0
chance kept, $w/384$	0.05	0.75	0.20	0
$W = 384/(4\,\hat{p})$	480	32	120	–

$\sum w = 384$

Worked: four candidates, one kept

Try first: $N = 96$ lamps, $M = 4$ candidates. Fill in w, the chance of being kept, W, and $\hat{p}(y)\,W$.

	A	B	C	D	
$\hat{p}$ (light, no shadow ray)	0.2	3.0	0.8	0	
$w = 96\,\hat{p}$	19.2	288	76.8	0	$\sum w = 384$
chance kept, $w/384$	0.05	0.75	0.20	0	
$W = 384/(4\,\hat{p})$	480	32	120	–	
estimate, if lit: $\hat{p}\,W$	**96**	**96**	**96**	–	

Worked: four candidates, one kept

Try first: $N = 96$ lamps, $M = 4$ candidates. Fill in w, the chance of being kept, W, and $\hat{p}(y)\,W$.

	A	**B**	**C**	**D**
$\hat{p}$ (light, no shadow ray)	0.2	3.0	0.8	0
$w = 96\,\hat{p}$	19.2	288	76.8	0
chance kept, $w/384$	0.05	0.75	0.20	0
$W = 384/(4\,\hat{p})$	480	32	120	–
estimate, if lit: $\hat{p}\,W$	**96**	**96**	**96**	–

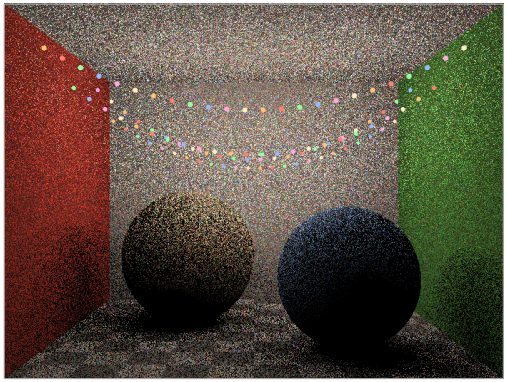
$\sum w = 384$

Evenly, one lamp: $96\,\hat{p} = 19.2$, 288, 76.8 or 0.

Resampled: 96 every time: the average of those four, for one shadow ray.

Rendered: one lamp evenly, against 32 candidates

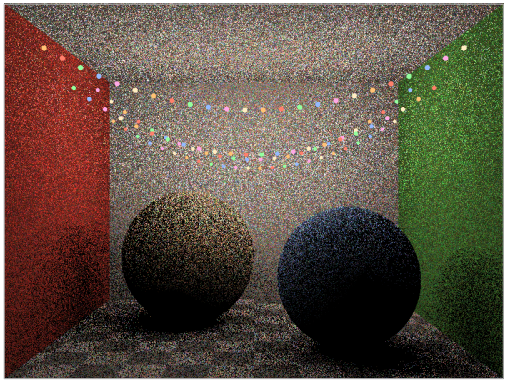
The night room, light straight from the lamps, one shadow ray per pixel



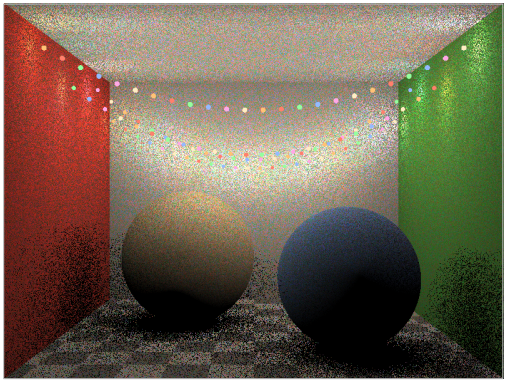
one lamp, picked evenly

Rendered: one lamp evenly, against 32 candidates

The night room, light straight from the lamps, one shadow ray per pixel



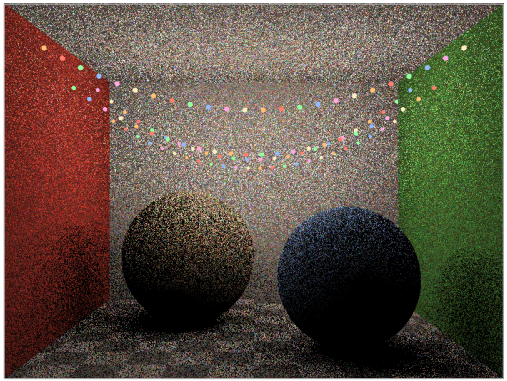
one lamp, picked evenly



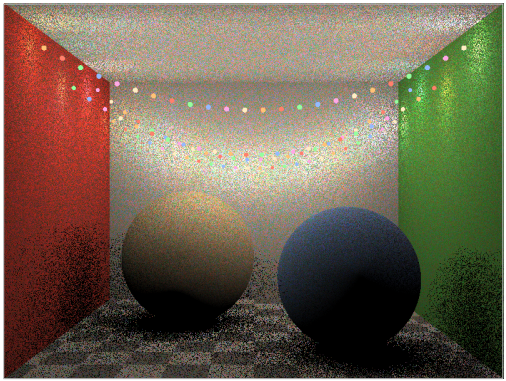
32 candidates, one kept

Rendered: one lamp evenly, against 32 candidates

The night room, light straight from the lamps, one shadow ray per pixel



one lamp, picked evenly



32 candidates, one kept

- The same rays, less noise; grain left next to the strings and in shadows.

The reservoir: keep one while the candidates stream past

C25's candidates, one at a time. Keep the new one with chance $w / \sum w$.

	arrives	w	$\sum w$ so far	chance to replace
kept: A	A	19.2	19.2	1

The reservoir: keep one while the candidates stream past

C25's candidates, one at a time. Keep the new one with chance $w / \sum w$.

	arrives	w	$\sum w$ so far	chance to replace
kept: A	A	19.2	19.2	1
kept: B	B	288	307.2	$288/307.2 = 0.9375$

The reservoir: keep one while the candidates stream past

C25's candidates, one at a time. Keep the new one with chance $w / \sum w$.

	arrives	w	$\sum w$ so far	chance to replace
kept: A	A	19.2	19.2	1
kept: B	B	288	307.2	$288/307.2 = 0.9375$
kept: B or C	C	76.8	384	$76.8/384 = 0.2$

The reservoir: keep one while the candidates stream past

C25's candidates, one at a time. Keep the new one with chance $w / \sum w$.

	arrives	w	$\sum w$ so far	chance to replace
kept: A	A	19.2	19.2	1
kept: B	B	288	307.2	$288/307.2 = 0.9375$
kept: B or C	C	76.8	384	$76.8/384 = 0.2$
kept: B or C	D	0	384	0

The reservoir: keep one while the candidates stream past

C25's candidates, one at a time. Keep the new one with chance $w / \sum w$.

	arrives	w	$\sum w$ so far	chance to replace
kept: A	A	19.2	19.2	1
kept: B	B	288	307.2	$288/307.2 = 0.9375$
kept: B or C	C	76.8	384	$76.8/384 = 0.2$
kept: B or C	D	0	384	0

B at the end: $0.9375 \times (1 - 0.2) = 0.75$ as in C25

The reservoir: keep one while the candidates stream past

C25's candidates, one at a time. Keep the new one with chance $w / \sum w$.

	arrives	w	$\sum w$ so far	chance to replace
kept: A	A	19.2	19.2	1
kept: B	B	288	307.2	$288/307.2 = 0.9375$
kept: B or C	C	76.8	384	$76.8/384 = 0.2$
kept: B or C	D	0	384	0

B at the end: $0.9375 \times (1 - 0.2) = 0.75$ as in C25

- No list: three numbers, y, $\sum w$ and M, the count so far.
- Same odds as choosing from the whole list at the end.

The reservoir, in Python

```
class Reservoir:
    def __init__(self):
        self.y, self.wsum, self.M = None, 0.0, 0
```

a reservoir

`y`: the lamp kept

`wsum`: $\sum w$

`M`: candidates seen

The reservoir, in Python

```
class Reservoir:
    def __init__(self):
        self.y, self.wsum, self.M = None, 0.0, 0
    def update(self, x, w, m=1):
        self.wsum += w;  self.M += m
        if random.random() * self.wsum < w:  self.y = x
```

a reservoir

`y`: the lamp kept

`wsum`: $\sum w$

`M`: candidates seen

- `m`: how many candidates this one stands for (1 now; many next slide).

The reservoir, in Python

```
class Reservoir:
    def __init__(self):
        self.y, self.wsum, self.M = None, 0.0, 0
    def update(self, x, w, m=1):
        self.wsum += w;  self.M += m
        if random.random() * self.wsum < w:  self.y = x

r = Reservoir()
for k in range(32):
    lamp = random.choice(lamps)
    r.update(lamp, p_hat(lamp, rec) * len(lamps))
```

a reservoir

y: the lamp kept

wsum: $\sum w$

M: candidates seen



candidates stream in

- M: how many candidates this one stands for (1 now; many next slide).

The reservoir, in Python

```
class Reservoir:
    def __init__(self):
        self.y, self.wsum, self.M = None, 0.0, 0
    def update(self, x, w, m=1):
        self.wsum += w;  self.M += m
        if random.random() * self.wsum < w:  self.y = x

r = Reservoir()
for k in range(32):
    lamp = random.choice(lamps)
    r.update(lamp, p_hat(lamp, rec) * len(lamps))

W = r.wsum / (r.M * p_hat(r.y, rec))
c = W * r.y.illuminate(rec)
```

a reservoir

y: the lamp kept

wsum: $\sum w$

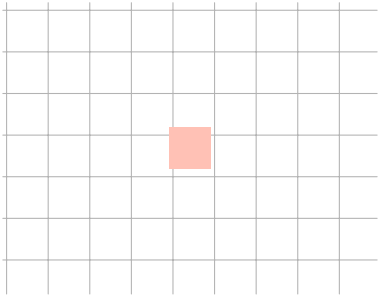
M: candidates seen



candidates stream in

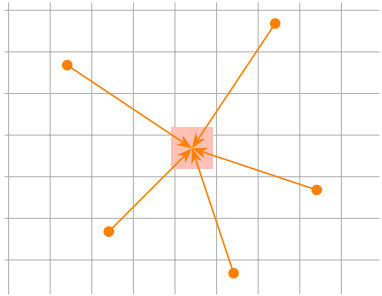
- m: how many candidates this one stands for (1 now; many next slide).
- One shadow ray, inside `illuminate`, for the lamp kept.

Reuse: a neighbour's reservoir is many candidates



this frame

Reuse: a neighbour's reservoir is many candidates



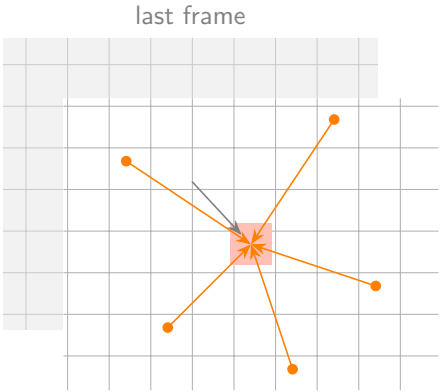
this frame

Merge neighbour q's reservoir as one candidate:

$$\texttt{r.update}(y_q,\ \hat{p}_{\text{here}}(y_q)\, W_q\, M_q,\ M_q)$$

- Neighbours see nearly the same lamps: take their picks, 5 of them.

Reuse: a neighbour's reservoir is many candidates

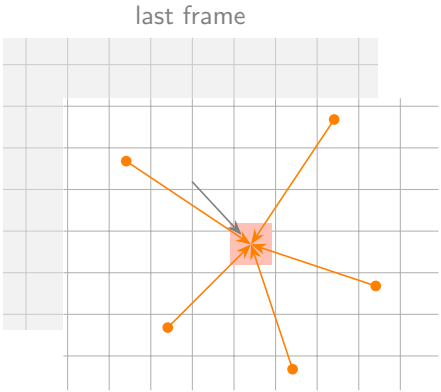


Merge neighbour q's reservoir as one candidate:

$$\texttt{r.update}(y_q,\ \hat{p}_{\text{here}}(y_q)\, W_q\, M_q,\ M_q)$$

- Neighbours see nearly the same lamps: take their picks, 5 of them.
- And last frame's, at the reprojected pixel (C17).

Reuse: a neighbour's reservoir is many candidates

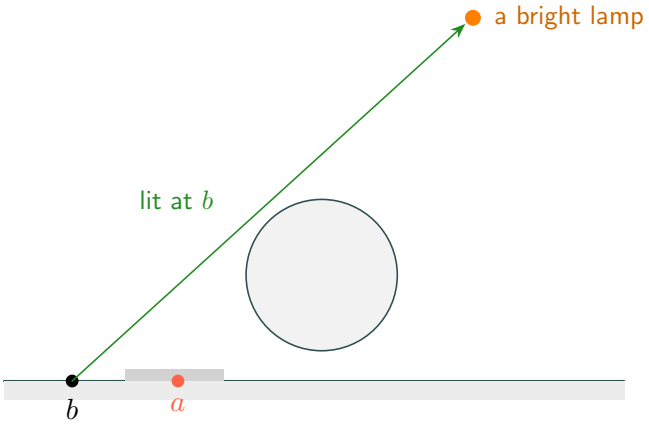


Merge neighbour q's reservoir as one candidate:

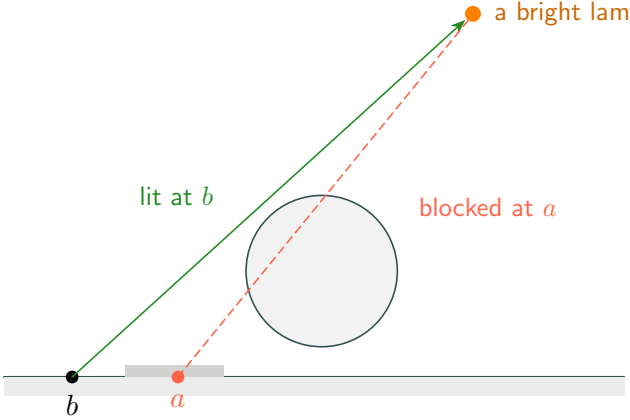
$$\texttt{r.update}(y_q,\ \hat{p}_{\text{here}}(y_q)\, W_q\, M_q,\ M_q)$$

- Neighbours see nearly the same lamps: take their picks, 5 of them.
- And last frame's, at the reprojected pixel (C17).
- The counts add up: after 8 frames our reservoirs stood for about 19,000 candidates each.

Shared picks have a price

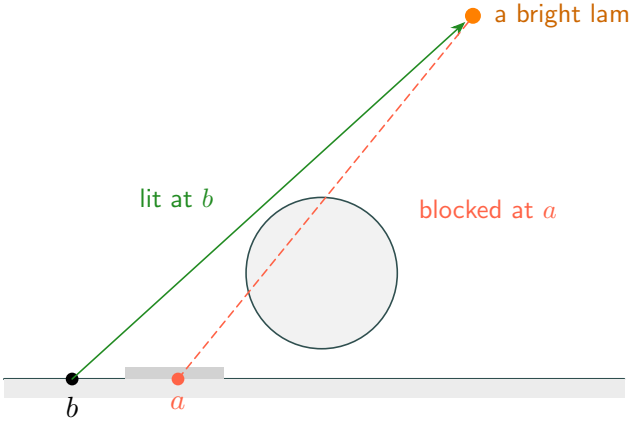


Shared picks have a price



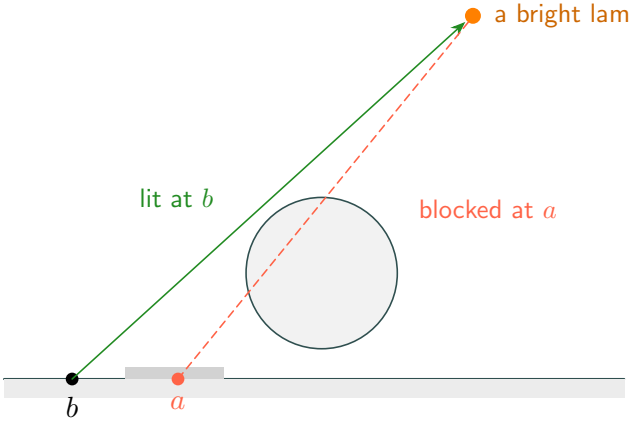
- A neighbour's good lamp can be hidden here.

Shared picks have a price



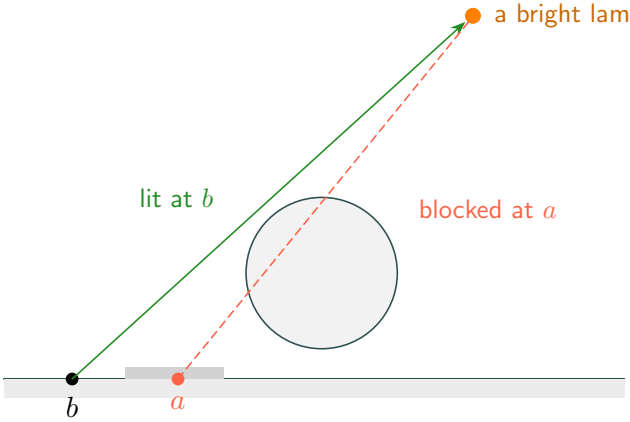
- A neighbour's good lamp can be hidden here.
- Fast version: a little too dark near shadows (biased).

Shared picks have a price



- A neighbour's good lamp can be hidden here.
- Fast version: a little too dark near shadows (biased).
- Neighbour on another surface: skip it (C12's tests).

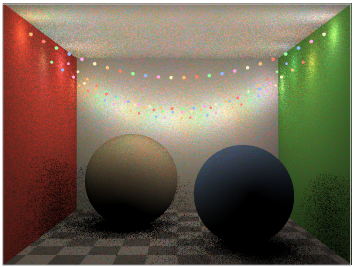
Shared picks have a price



- A neighbour's good lamp can be hidden here.
- Fast version: a little too dark near shadows (biased).
- Neighbour on another surface: skip it (C12's tests).
- Shared picks: noise in blotches, not grains.

Rendered: ReSTIR, then the filter, against the truth

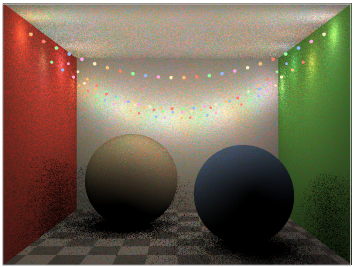
The night room: ReSTIR and SVGF at a shadow ray or two per pixel, against 96



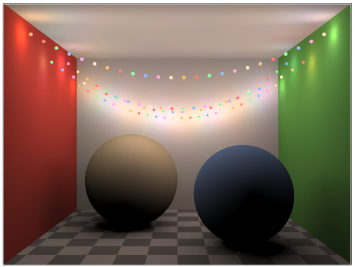
ReSTIR, frame 8

Rendered: ReSTIR, then the filter, against the truth

The night room: ReSTIR and SVGF at a shadow ray or two per pixel, against 96



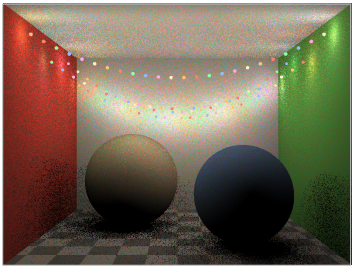
ReSTIR, frame 8



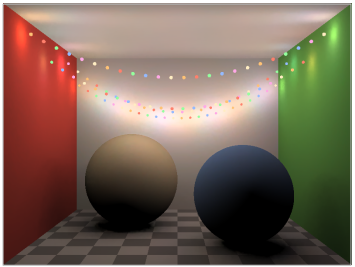
ReSTIR, then SVGF

Rendered: ReSTIR, then the filter, against the truth

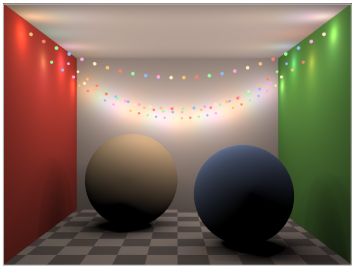
The night room: ReSTIR and SVGF at a shadow ray or two per pixel, against 96



ReSTIR, frame 8



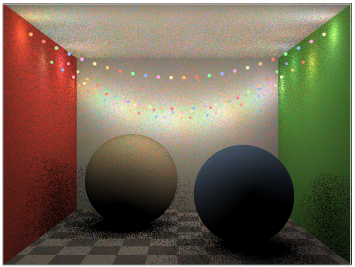
ReSTIR, then SVGF



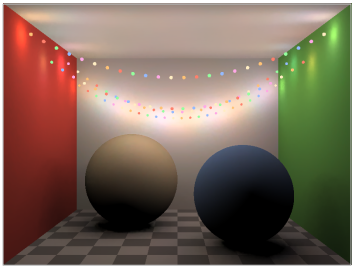
every lamp, its own ray

Rendered: ReSTIR, then the filter, against the truth

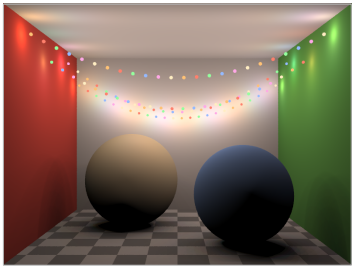
The night room: ReSTIR and SVGF at a shadow ray or two per pixel, against 96



ReSTIR, frame 8



ReSTIR, then SVGF



every lamp, its own ray

- Two shadow rays a pixel, not 96: the frame a game can afford.

Quick check 2: resampling

$w = \hat{p}/p$; **kept with chance** $w/\sum w$; $W = \sum w/(M\,\hat{p}(y))$

1. 50 lamps, one picked evenly; its light is 0.3. The estimate?
2. Two candidates, weights 1 and 3. The chance the second is kept?
3. A reservoir: $\sum w = 60$, $M = 4$, $\hat{p}(y) = 5$. Its W?
4. Why skip a neighbour on another surface?

- Guesses are cheap and shared; the shadow ray is the expensive part.

Quick check 2: resampling

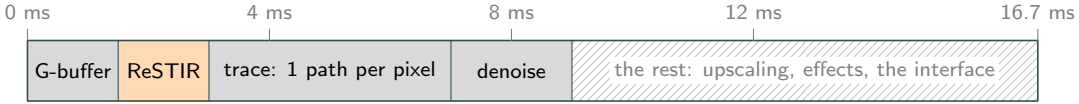
$w = \hat{p}/p$; **kept with chance** $w/\sum w$; $W = \sum w/(M\,\hat{p}(y))$

1. 50 lamps, one picked evenly; its light is 0.3. The estimate? $50 \times 0.3 = 15$
2. Two candidates, weights 1 and 3. The chance the second is kept? $3/4$
3. A reservoir: $\sum w = 60$, $M = 4$, $\hat{p}(y) = 5$. Its W? $60/20 = 3$
4. Why skip a neighbour on another surface? its lamps differ

- Guesses are cheap and shared; the shadow ray is the expensive part.

Recap: one frame, 16.7 ms

One frame at 60 frames a second: 16.7 ms (an example budget)

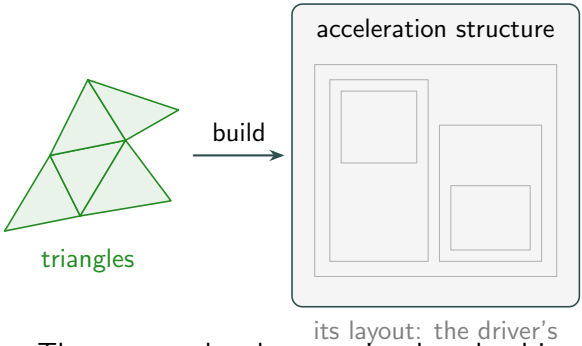


1. Rasterise the first hit into a G-buffer; trace one path per pixel
2. Denoise: G-buffer weights, history by reprojection, SVGF
3. Many lamps: candidates in a reservoir, shared by neighbours

Ray tracing in hardware

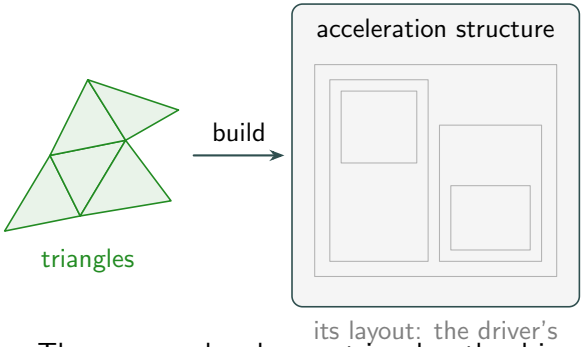
The w12 tree, built and walked by the card

Your triangles in, the driver's tree out



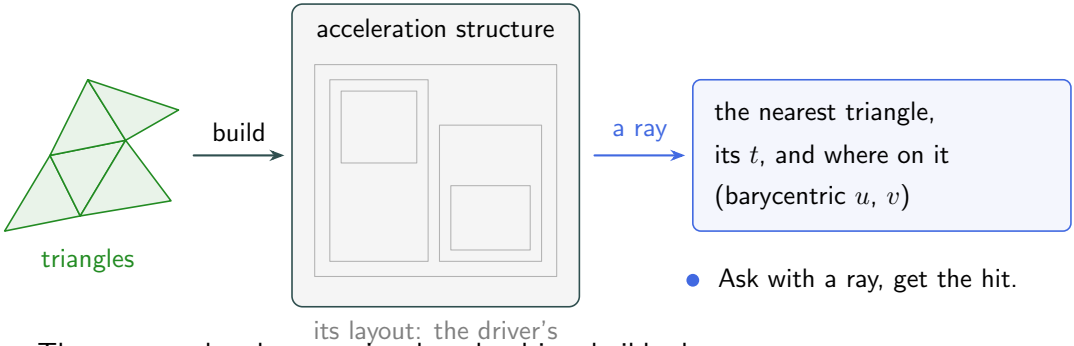
- The program hands over triangles; the driver builds the tree.

Your triangles in, the driver's tree out



- The program hands over triangles; the driver builds the tree.
- Inside: the w12 lectures' boxes in boxes, laid out for the card.

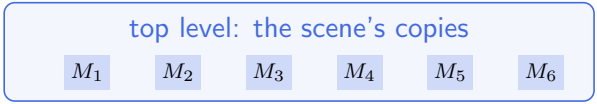
Your triangles in, the driver's tree out



- Ask with a ray, get the hit.

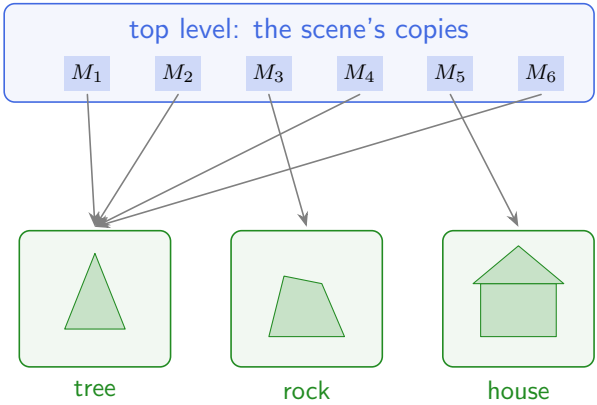
- The program hands over triangles; the driver builds the tree.
- Inside: the w12 lectures' boxes in boxes, laid out for the card.

Two levels: a tree per mesh, a tree of copies



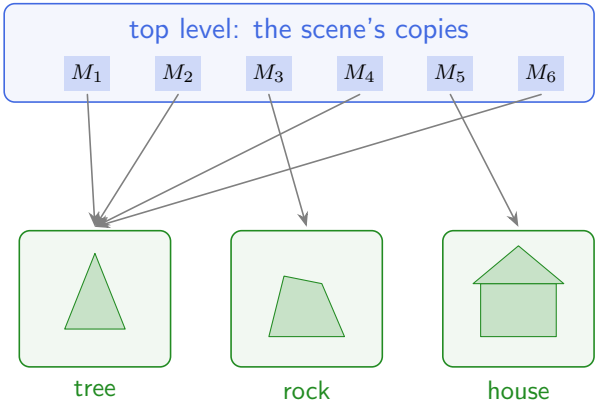
- Top: each copy is a mesh and a matrix (lecture 8).

Two levels: a tree per mesh, a tree of copies



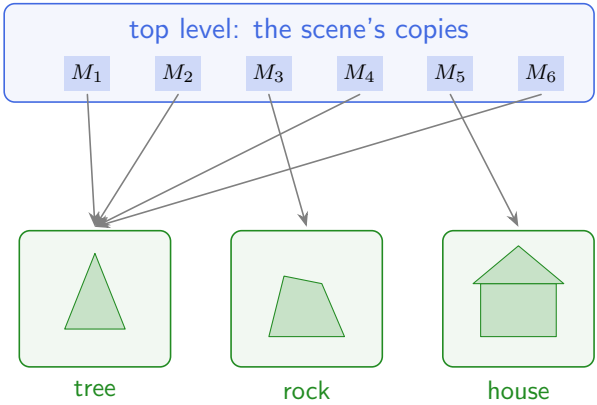
- Top: each copy is a mesh and a matrix (lecture 8).
- Bottom: one tree per mesh, built once. A forest: one tree, many matrices.

Two levels: a tree per mesh, a tree of copies



- Top: each copy is a mesh and a matrix (lecture 8).
- Bottom: one tree per mesh, built once. A forest: one tree, many matrices.
- Into a copy, the ray is moved by M^{-1} (lecture 9).

Two levels: a tree per mesh, a tree of copies



- Top: each copy is a mesh and a matrix (lecture 8).
- Bottom: one tree per mesh, built once. A forest: one tree, many matrices.
- Into a copy, the ray is moved by M^{-1} (lecture 9).
- Things move: rebuild the top each frame, refit the bottom.

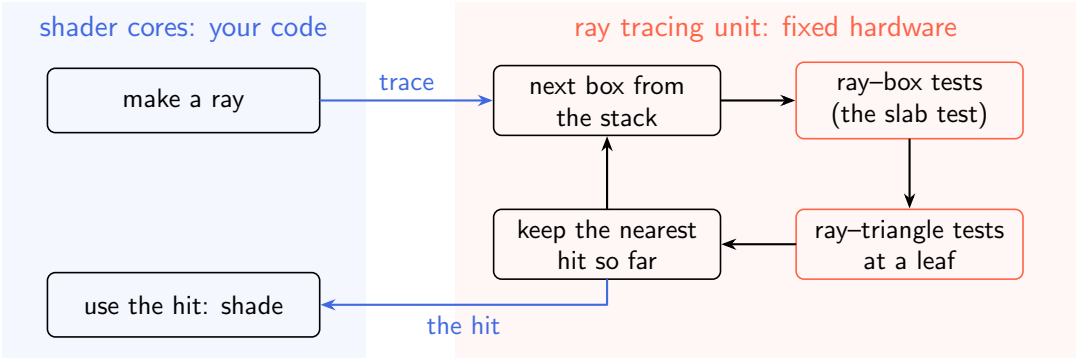
What the hardware does: boxes and triangles

shader cores: your code

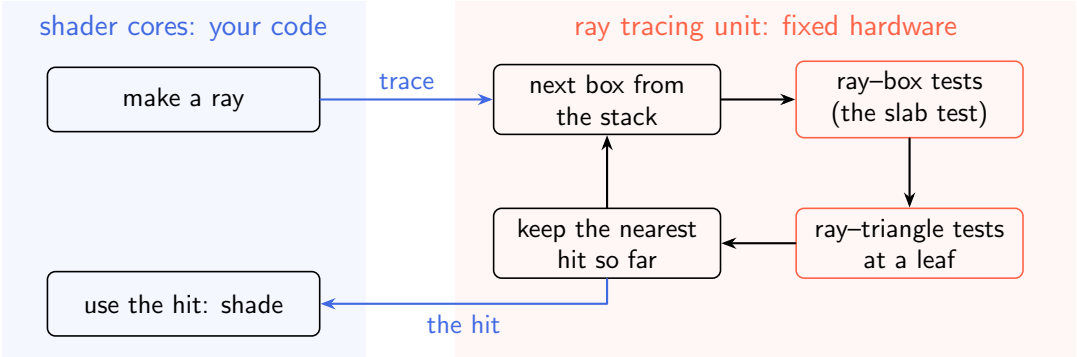
make a ray

use the hit: shade

What the hardware does: boxes and triangles

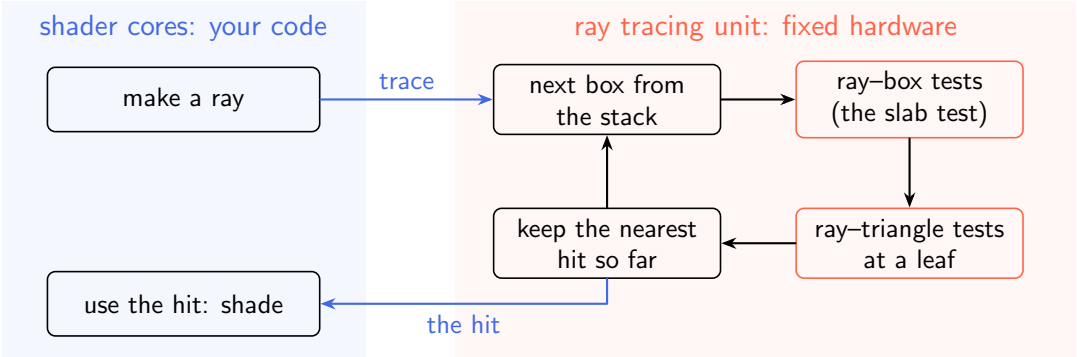


What the hardware does: boxes and triangles



- The w12 lectures' slab test, and the triangle test, in silicon.

What the hardware does: boxes and triangles



- The w12 lectures' slab test, and the triangle test, in silicon.
- The shader cores work on other pixels while a ray is walked.

Five kinds of shader, and the tracer you know

shader	runs	in lecture 20's tracer
ray generation	once per pixel	the loop over pixels; makes the rays

Five kinds of shader, and the tracer you know

shader	runs	in lecture 20's tracer
ray generation	once per pixel	the loop over pixels; makes the rays
closest hit	at the nearest hit	the material's branch (`isinstance`)
miss	when nothing is hit	the sky's colour

Five kinds of shader, and the tracer you know

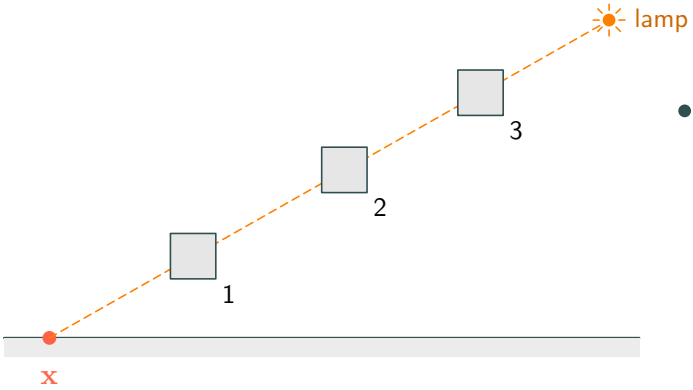
shader	runs	in lecture 20's tracer
ray generation	once per pixel	the loop over pixels; makes the rays
closest hit	at the nearest hit	the material's branch (`isinstance`)
miss	when nothing is hit	the sky's colour
any hit	at each hit on the way	(new) a leaf with holes: ignore the hit
intersection	at a leaf of non-triangles	lecture 14's ray–sphere test

Five kinds of shader, and the tracer you know

shader	runs	in lecture 20's tracer
ray generation	once per pixel	the loop over pixels; makes the rays
closest hit	at the nearest hit	the material's branch (`isinstance`)
miss	when nothing is hit	the sky's colour
any hit	at each hit on the way	(new) a leaf with holes: ignore the hit
intersection	at a leaf of non-triangles	lecture 14's ray–sphere test

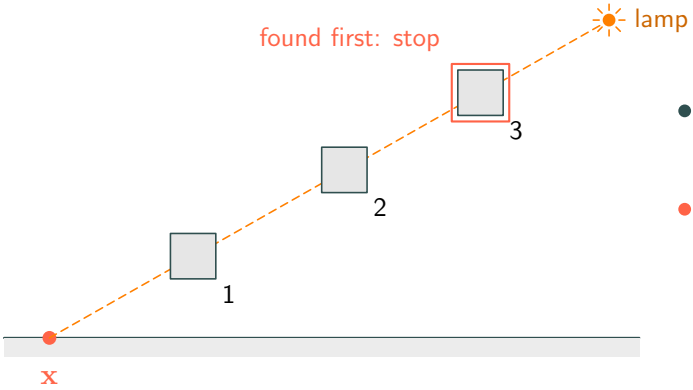
- `TraceRay(...)` takes the place of `scene.hit(ray)`; the card calls the others.
- Lecture 20's `trace`, cut into pieces the card can schedule.

A shadow ray asks “anything?”, not “what first?”



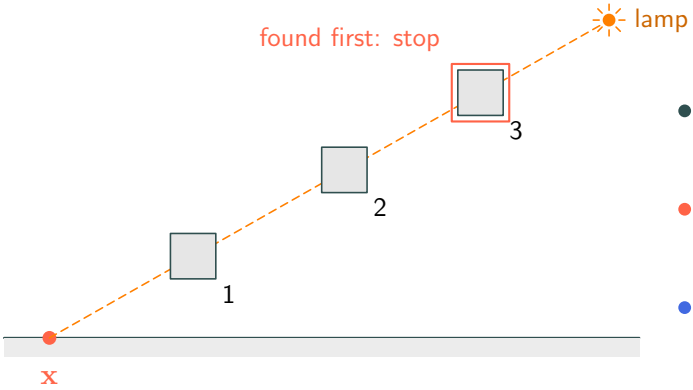
- Lecture 17’s question: is anything in the way?

A shadow ray asks “anything?”, not “what first?”



- Lecture 17’s question: is anything in the way?
- Any hit will do: stop the search at the first one found.

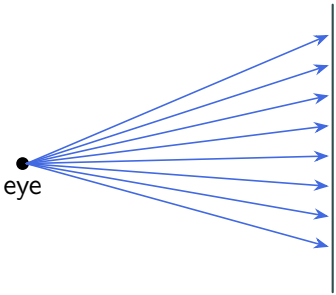
A shadow ray asks “anything?”, not “what first?”



The miss shader alone marks the point lit.

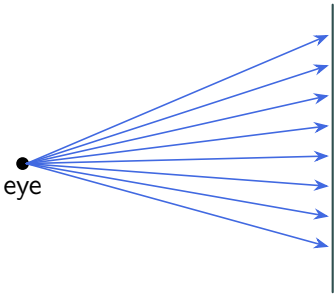
- Lecture 17’s question: is anything in the way?
- Any hit will do: stop the search at the first one found.
- Two flags: end at the first hit, skip the closest-hit shader.

Rays that go different ways

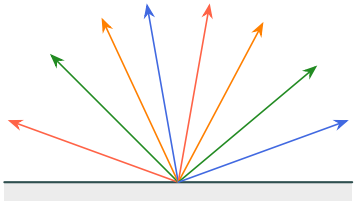


first rays: together

Rays that go different ways

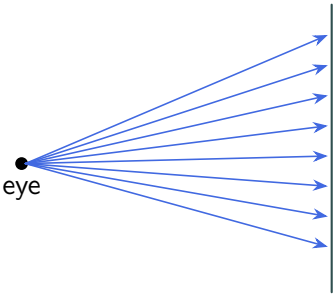


first rays: together

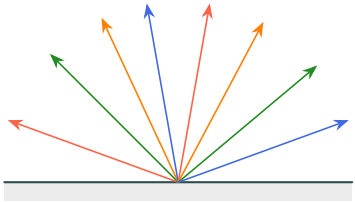


bounce rays: scattered

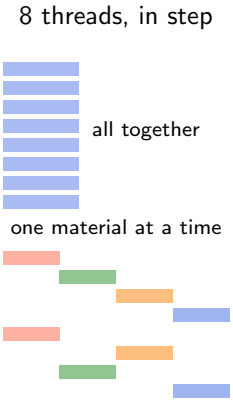
Rays that go different ways



first rays: together

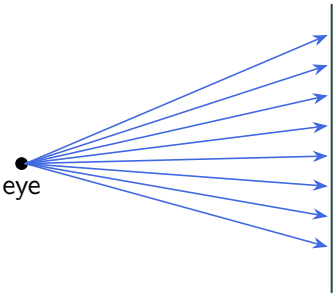


bounce rays: scattered

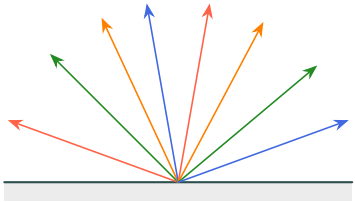


- A GPU runs 32 threads in step: one instruction for all.

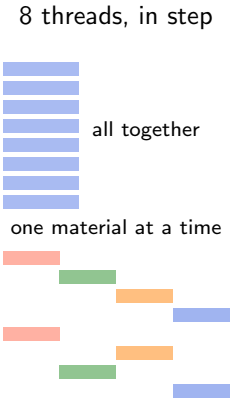
Rays that go different ways



first rays: together

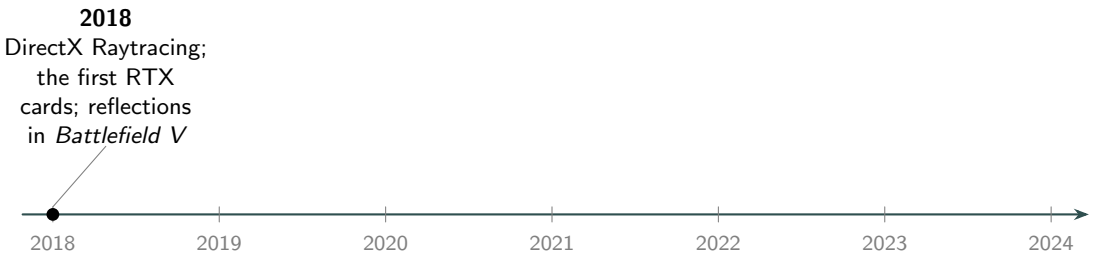


bounce rays: scattered

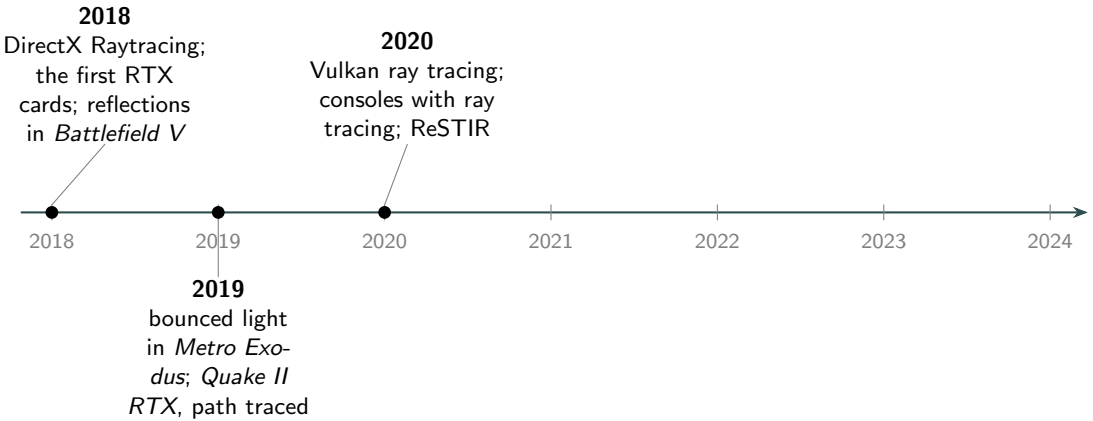


- A GPU runs 32 threads in step: one instruction for all.
- Scattered rays: threads wait for each other. Sorting them helps.

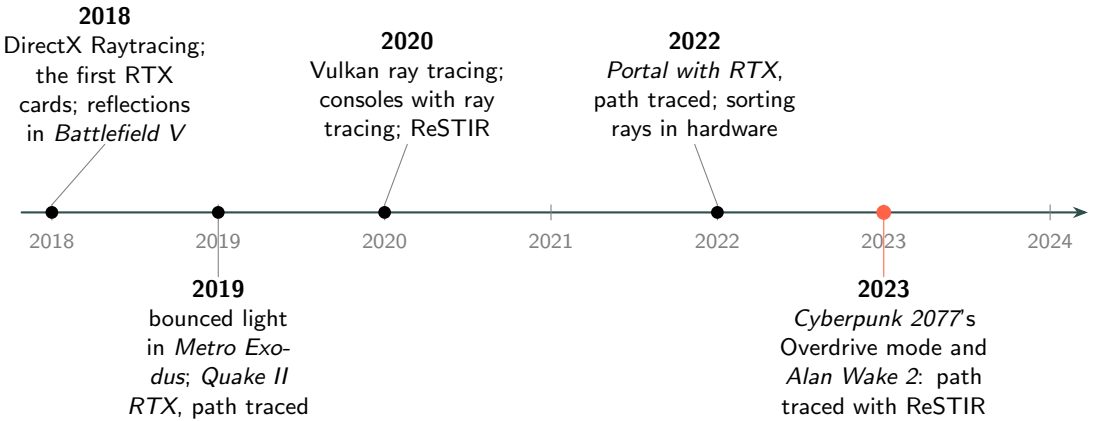
Where games are now



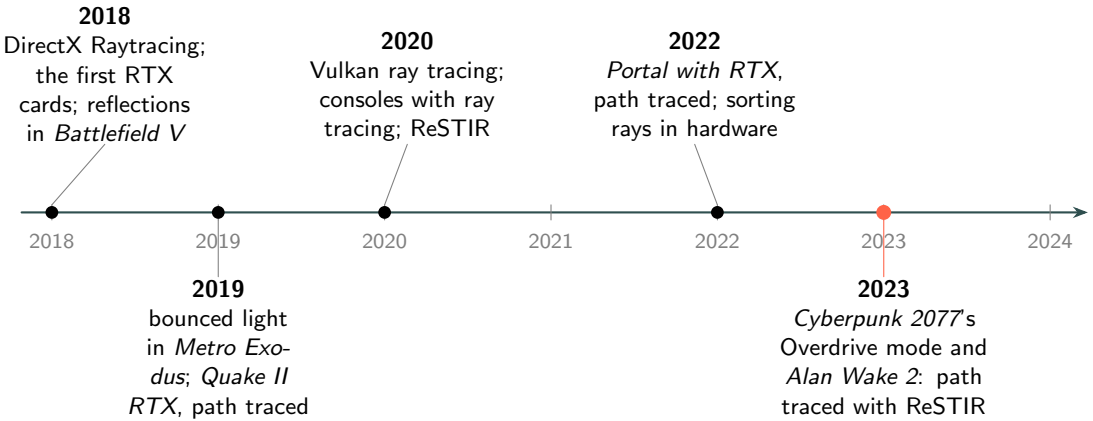
Where games are now



Where games are now



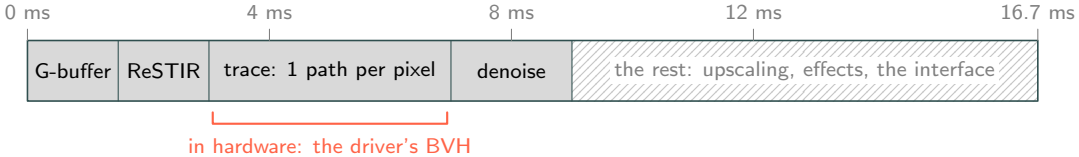
Where games are now



- Five years from one effect to lecture 20's whole path, every frame.

Recap: one frame, 16.7 ms

One frame at 60 frames a second: 16.7 ms (an example budget)



1. Rasterise the first hit into a G-buffer; trace one path per pixel
2. Denoise: G-buffer weights, history by reprojection, SVGF
3. Many lamps: candidates in a reservoir, shared by neighbours
4. The trace in hardware: the driver's BVH, five kinds of shader

Summary

1 One path per pixel

16.7 ms: about 80 rays a pixel, at most

rasterise the first hit: the G-buffer

2 Denoising

$$\text{history} \leftarrow 0.8\,\text{history} + 0.2\,\text{new}$$

filter L/R; G-buffer weights; reprojection; SVGF

3 Many lamps: ReSTIR

$$f(y)\,W, \quad W = \frac{\sum w}{M\,\hat{p}(y)}$$

reservoirs shared by neighbours and frames

4 Ray tracing in hardware

`TraceRay` for `scene.hit`

the driver's two-level BVH; five kinds of shader

Reading

Schied et al., "Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination", HPG 2017

Dammertz, Sewtz, Hanika and Lensch, "Edge-avoiding À-Trous wavelet transform for fast global illumination filtering", HPG 2010

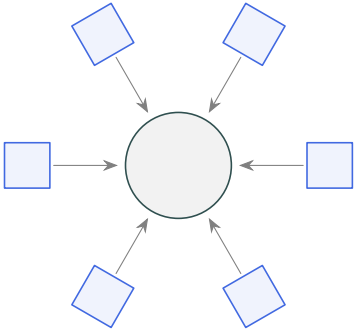
Bitterli, Wyman, Pharr, Shirley, Lefohn and Jarosz, "Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting", SIGGRAPH 2020

Talbot, Cline and Egbert, "Importance resampling for global illumination", EGSR 2005

Wyman et al., "A gentle introduction to ReSTIR", SIGGRAPH 2023 course

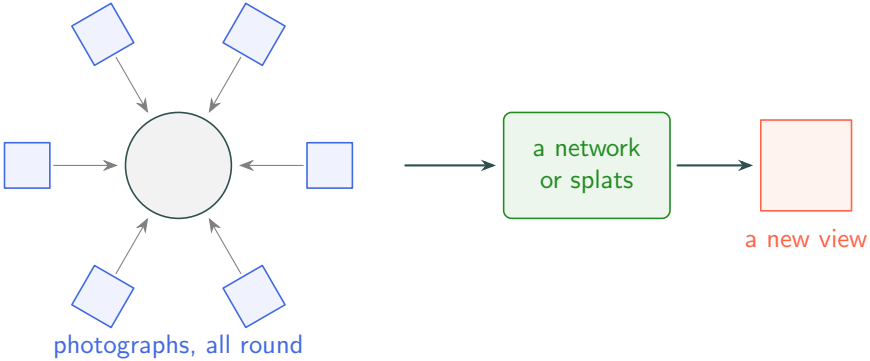
Haines and Akenine-Möller, eds., *Ray Tracing Gems* (2019); **Marrs, Shirley and Wald**, eds., *Ray Tracing Gems II* (2021); both free online

Next: Extra 7, learned scenes



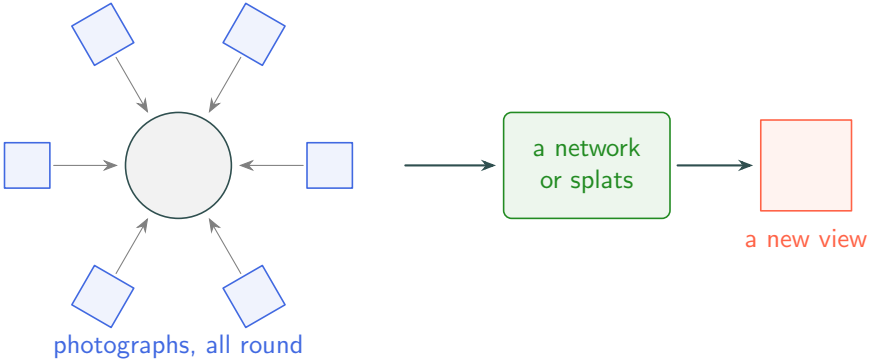
photographs, all round

Next: Extra 7, learned scenes



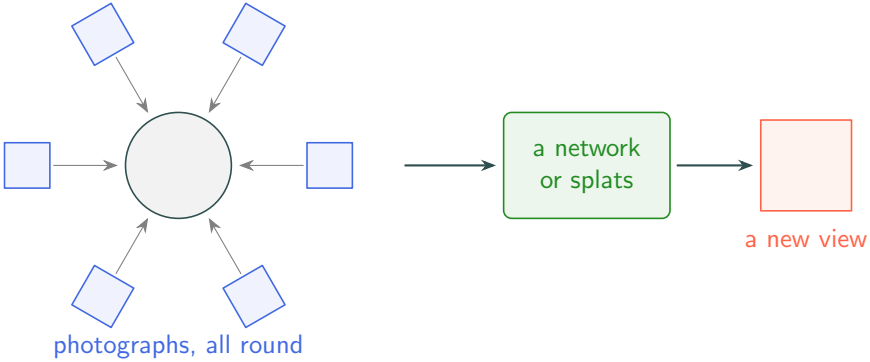
- Photographs in, a scene out.

Next: Extra 7, learned scenes



- Photographs in, a scene out.
- Drawn with Extra 5's volume equation.

Next: Extra 7, learned scenes



- Photographs in, a scene out.
- Drawn with Extra 5's volume equation.
- Extra 7: NeRF and Gaussian splatting.