
GDC

Temporal Reprojection Anti-Aliasing in INSIDE

Lasse Jon Fuglsang Pedersen
Programmer // **PLAYDEAD**

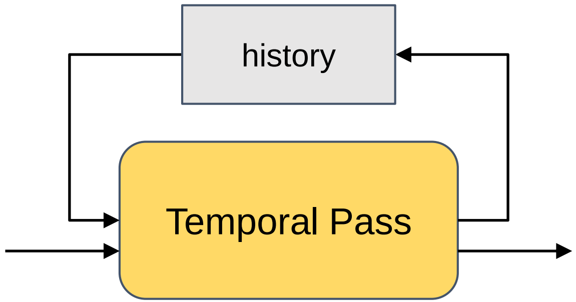
@codeverses

Background

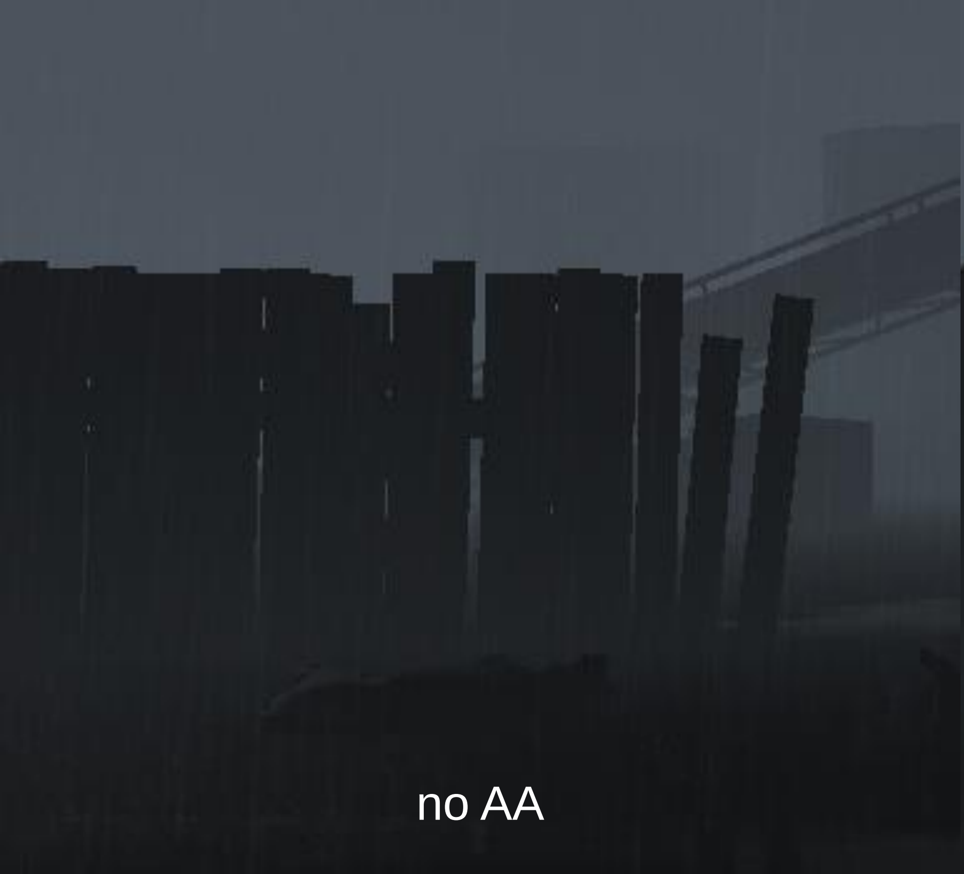
- **INSIDE** has lots of geometric detail, interleaved layers of transparency
- camera always slightly moving ⇒ lots of crawling
- … wanted clean, stable images
- began looking into temporal AA early 2014
- quickly became primary AA solution

Temporal Anti-Aliasing?

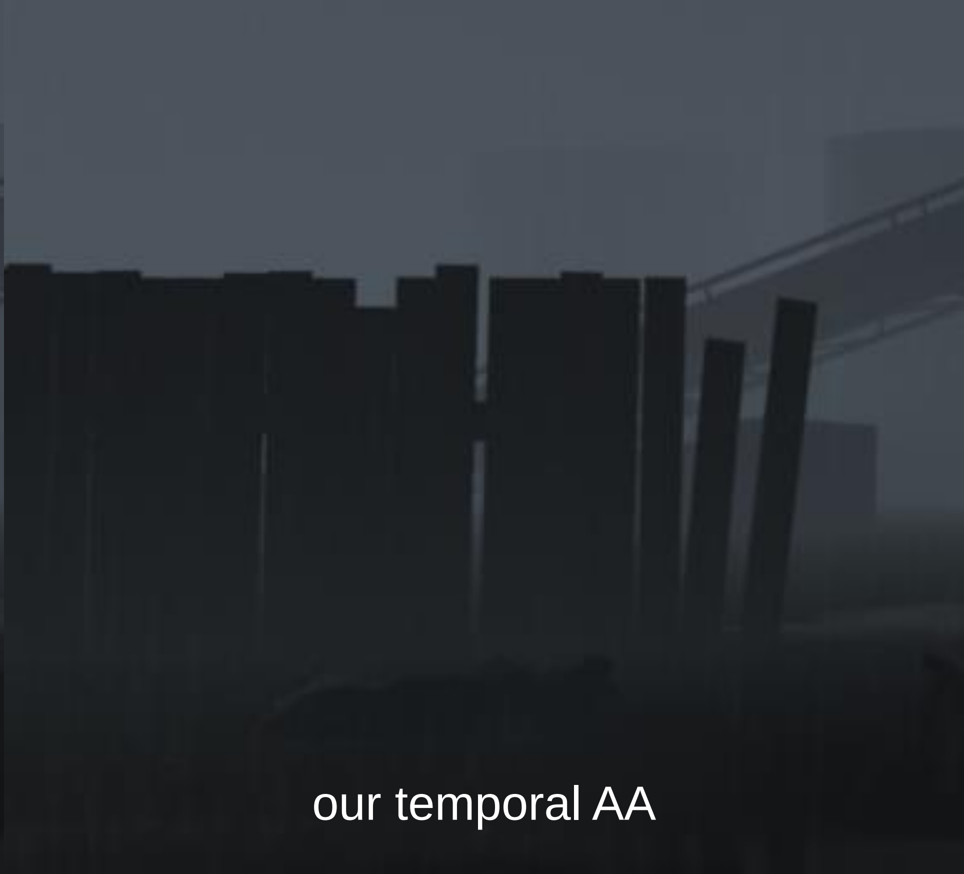
- spatio-temporal post-process technique (… *what?*)
- correlates new fragments with fragments from history buffer
- output becomes next frame in history (feedback loop)
- sub-pixel information recovered over time



What it looks like



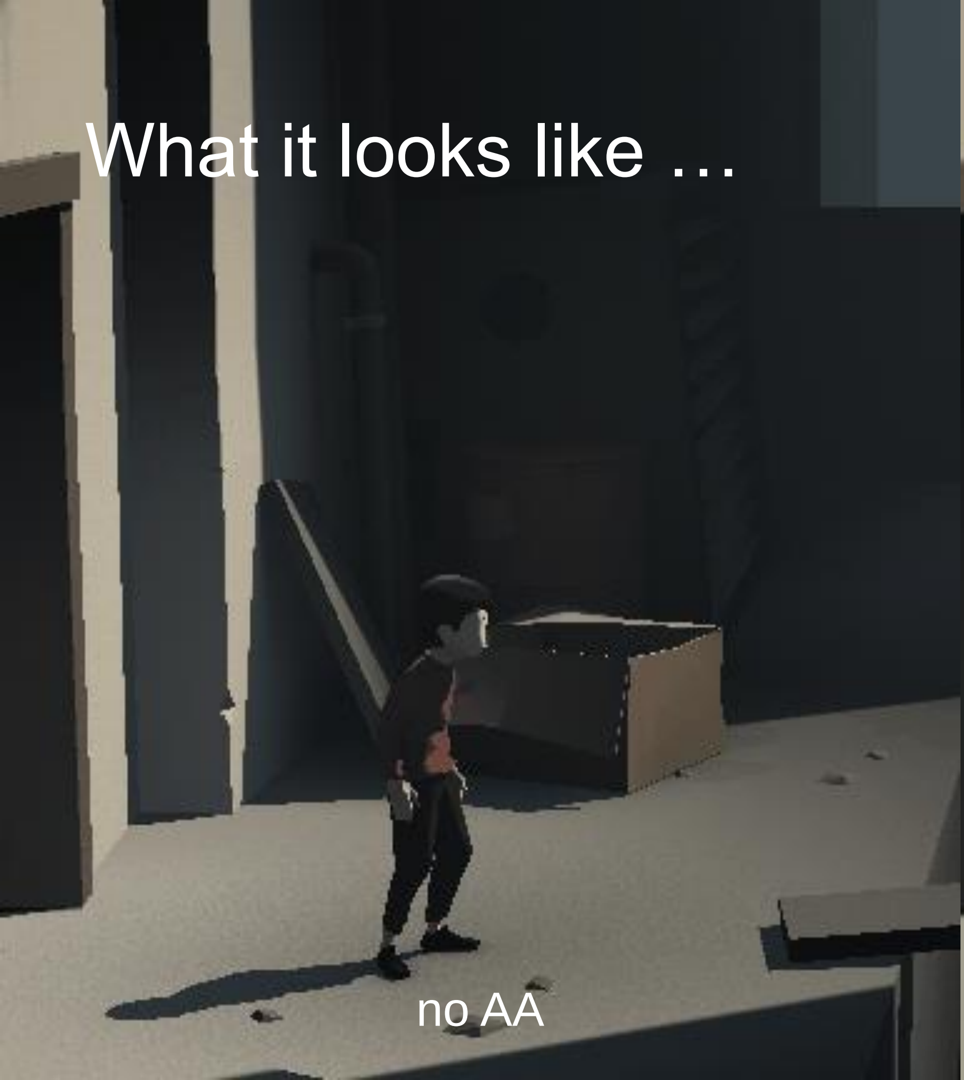
no AA



our temporal AA

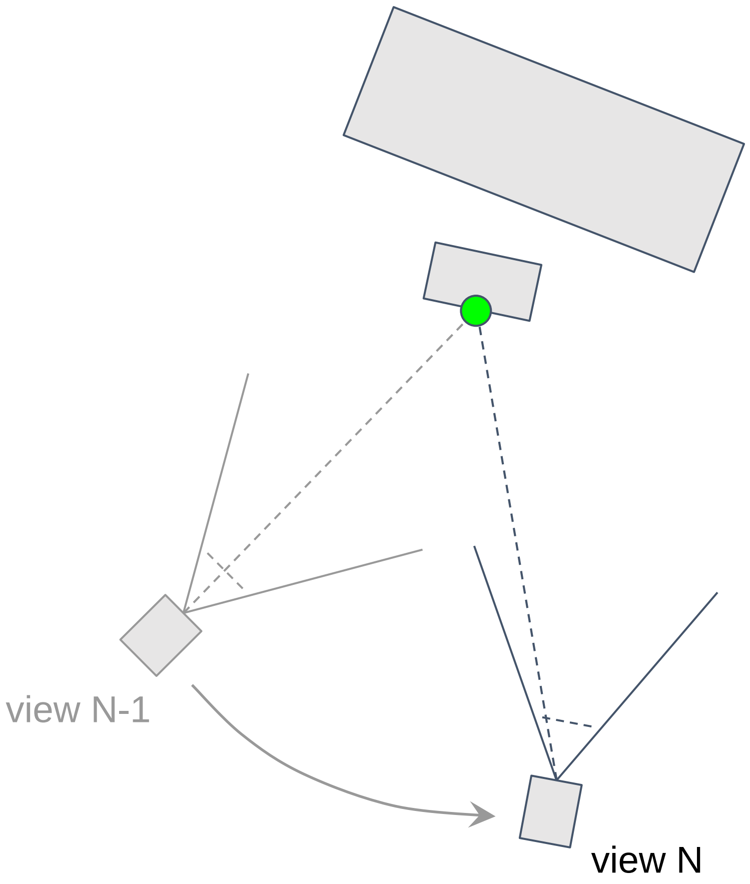
What it looks like …

What it looks like …



First some basic intuition

- local region of a surface fragment may remain in view across multiple frames
- if relationship between viewer and subject changes every frame, then rasterization ⇒ variation
- if we step back in time, then we can use this variation to refine the current frame

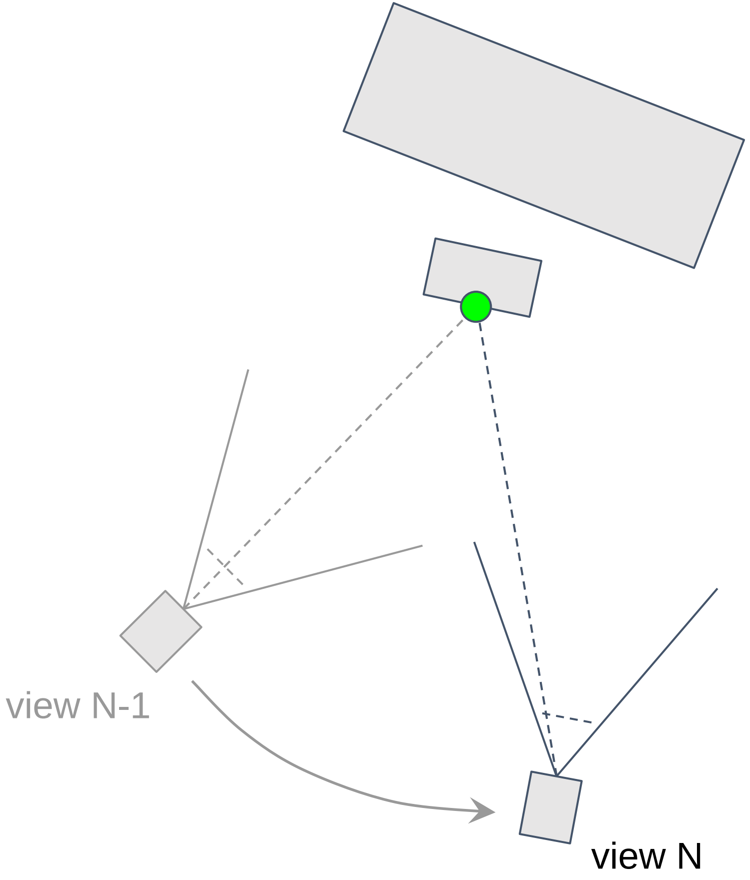


Stepping back in time

- want to correlate current frame fragments with fragments from previous frame(s)

- can do spatially, with reprojection
  - relies on depth buffer information
  - limited to closest written fragment

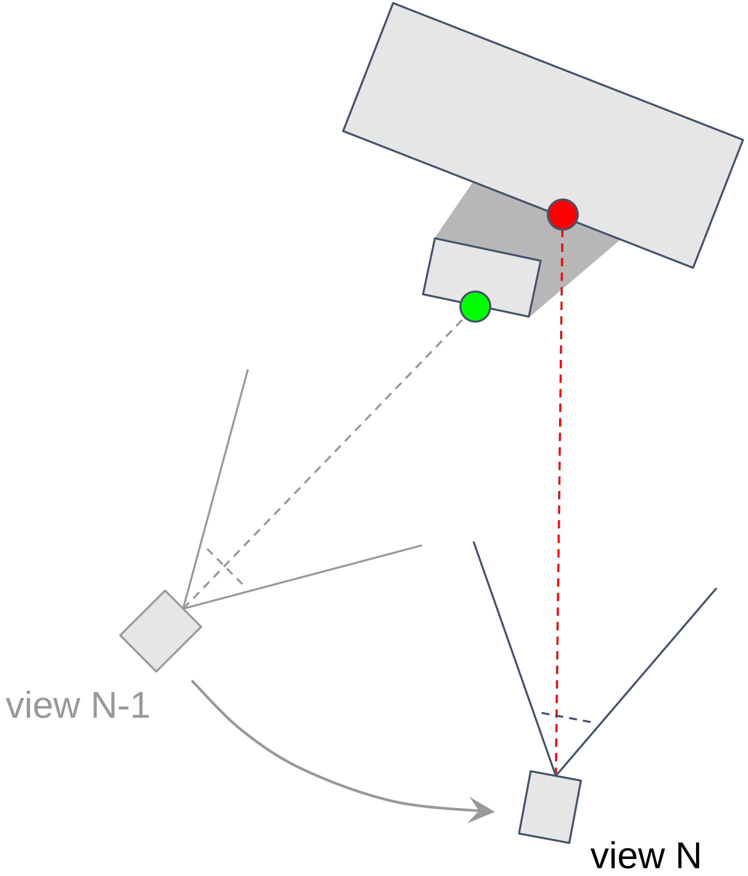
- not always possible
  - sometimes the data just isn’t there



Stepping into void

- fragments can become occluded or disoccluded at any time, making it difficult to accurately step back
    - bummer.. but let's get back to that later

- if relationship between viewer and subject never changes, there *is* no additional information to be gained from stepping back…

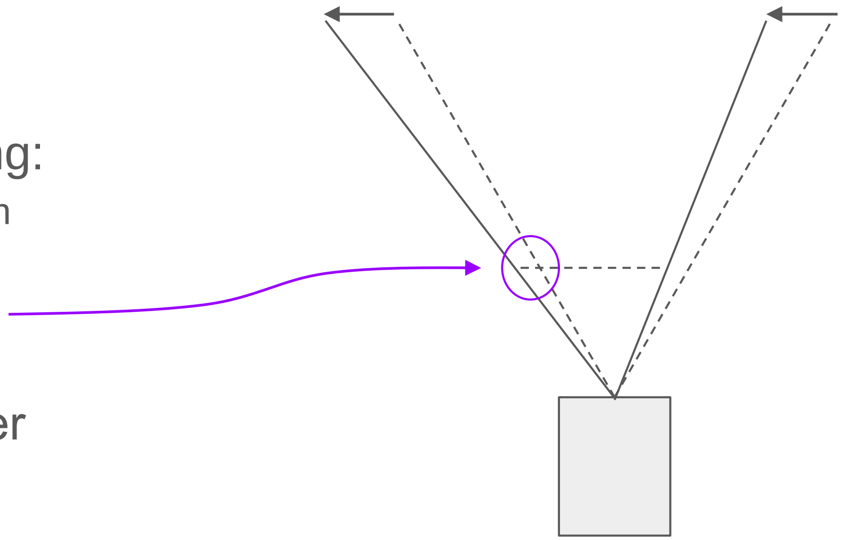


Step 1: Jitter your view frustum

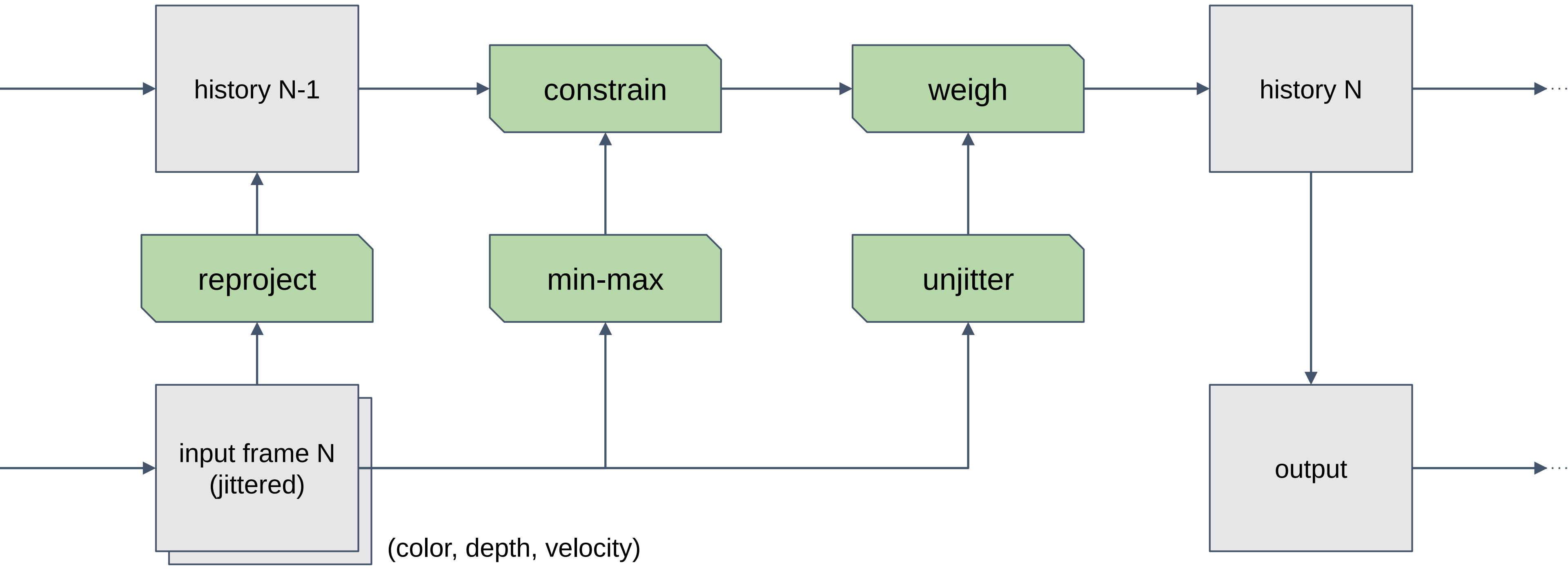
- have established that if camera is static, then we are losing information

- thus, every frame, prior to rendering:
    - get texel offset from sample distribution
    - use offset to calculate projection offset
    - use projection offset to shear frustum

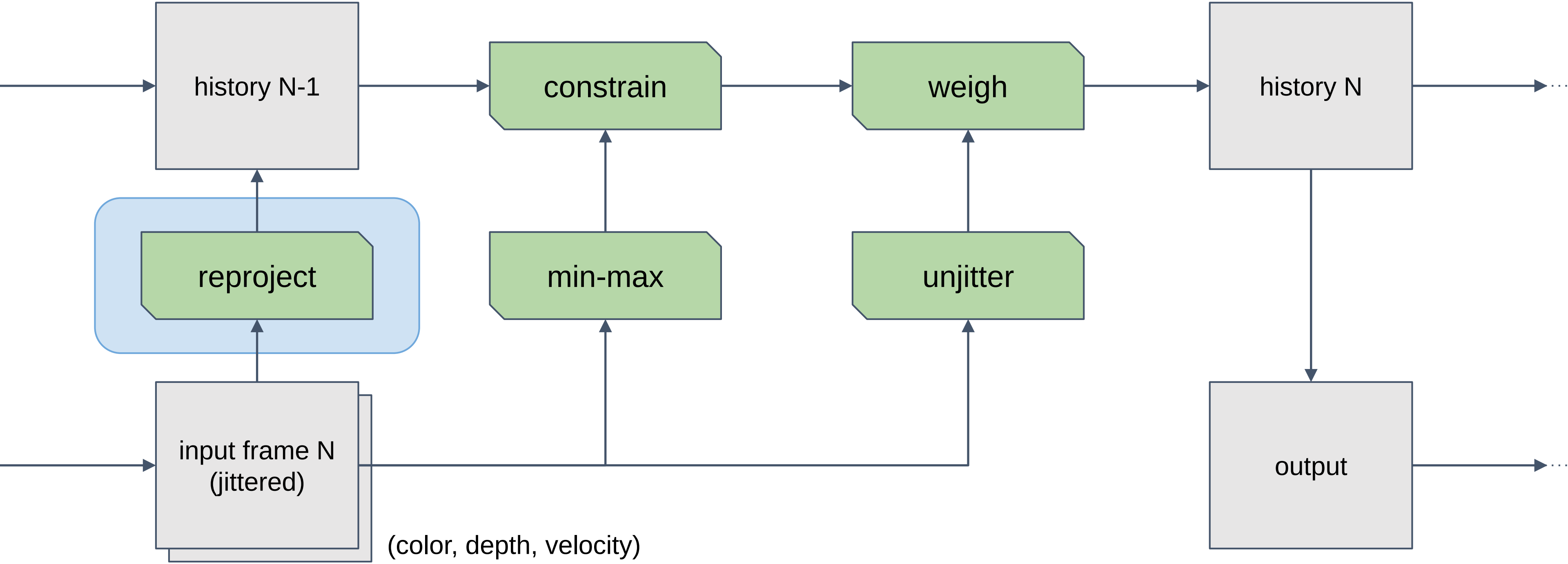
- … more on sample distribution later



Step 2: For every fragment …

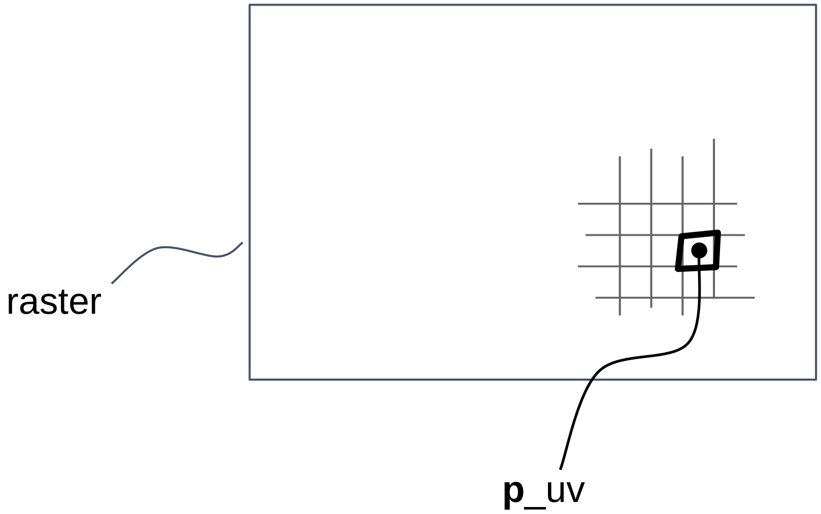


Step 2: For every fragment …



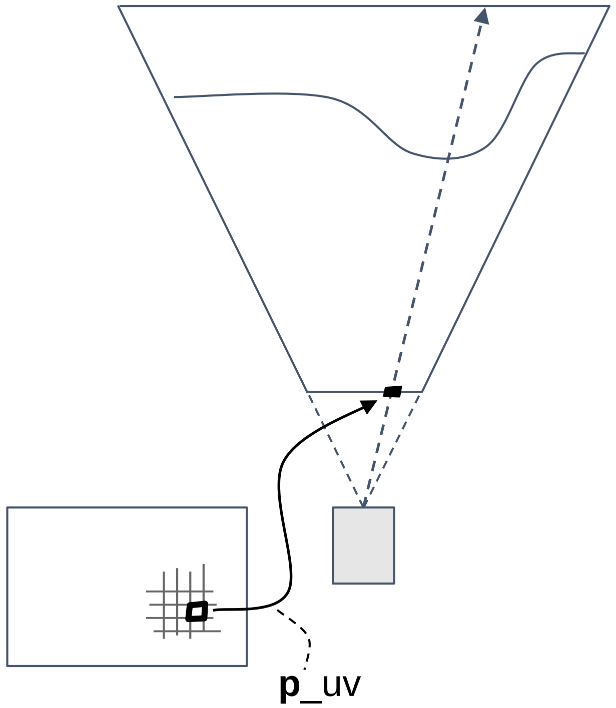
Reprojection of static scenes

- start in current fragment **p**_uv



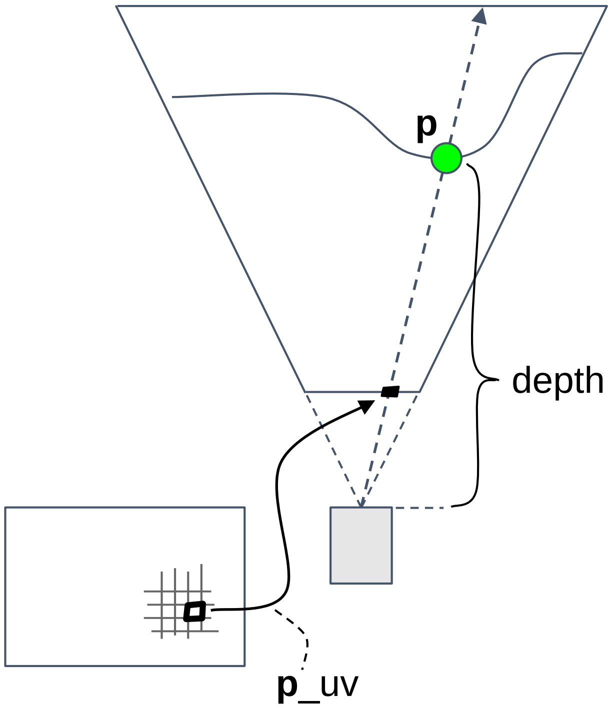
Reprojection of static scenes

- start in current fragment **p**_uv



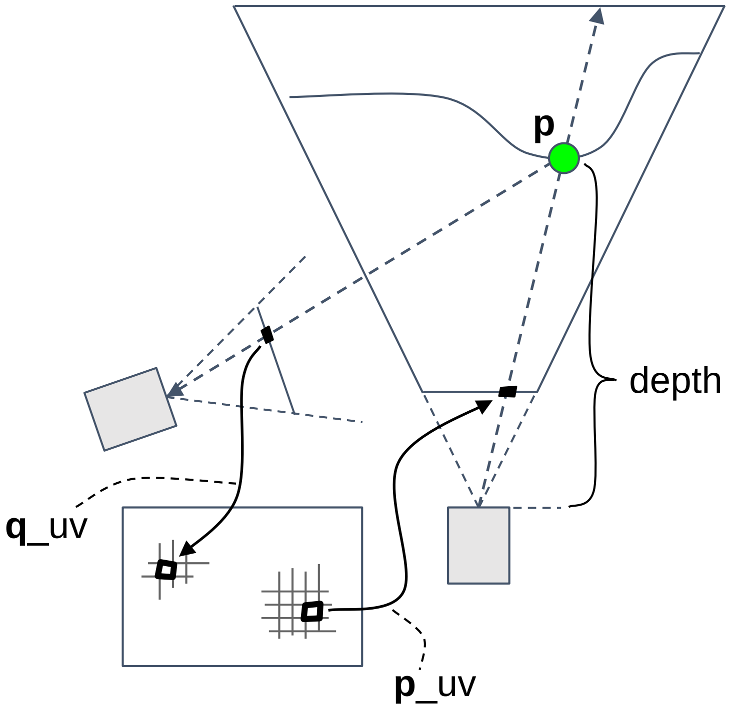
Reprojection of static scenes

- start in current fragment **p_uv**
- reconstruct world space **p** using depth and frustum params for current frame
  - lerp corner ray, scale by linear depth



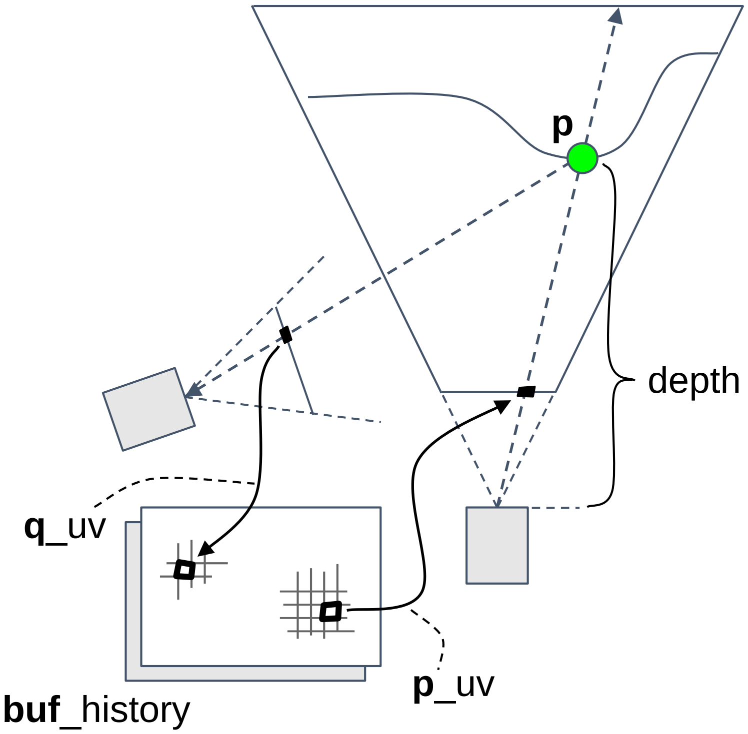
Reprojection of static scenes

- start in current fragment **p**_uv
- reconstruct world space **p** using depth and frustum params for current frame
  - lerp corner ray, scale by linear depth
- then, reproject **p** into previous frame
  - **q**_cs = mul(**VP**_prev’, **p**)
  - **q**_uv = 0.5 * (**q**_cs.xy / **q**_cs.w) + 0.5



Reprojection of static scenes

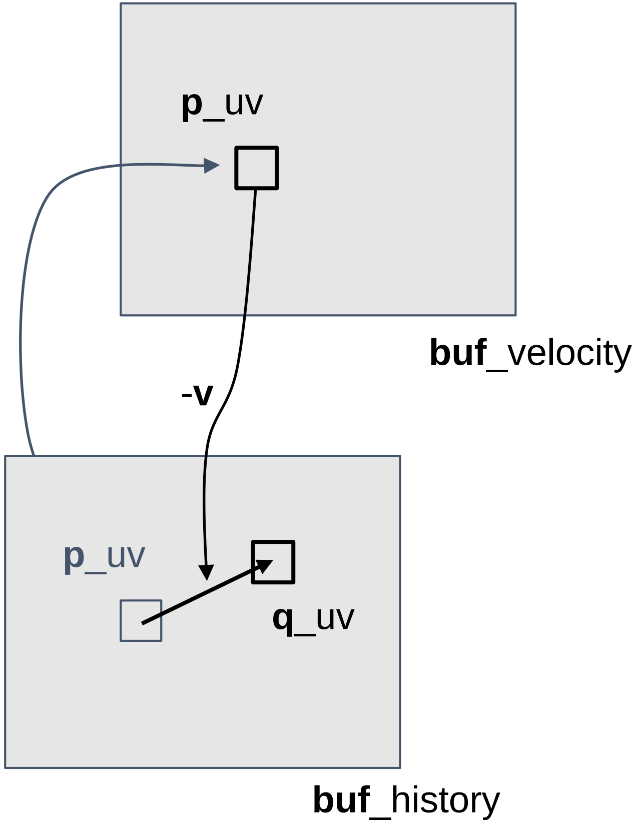
- start in current fragment **p**_uv
- reconstruct world space **p** using depth and frustum params for current frame
  - lerp corner ray, scale by linear depth
- then, reproject **p** into previous frame
  - **q**_cs = mul(**VP**_prev', **p**)
  - **q**_uv = 0.5 * (**q**_cs.xy / **q**_cs.w) + 0.5
- history sample is then
  - **c**_hist = sample(**buf**_history, **q**_uv)



Reprojection of dynamic scenes

- for dynamic scenes we need a velocity buffer
  - separate pass before temporal
  - initialize to camera motion using static reprojection
    - **v** = **p**_uv - **q**_uv
  - then render dynamic objects on top
    - **v** = compute_ssvel(**p**, **q**, **VP**, **VP**_prev')

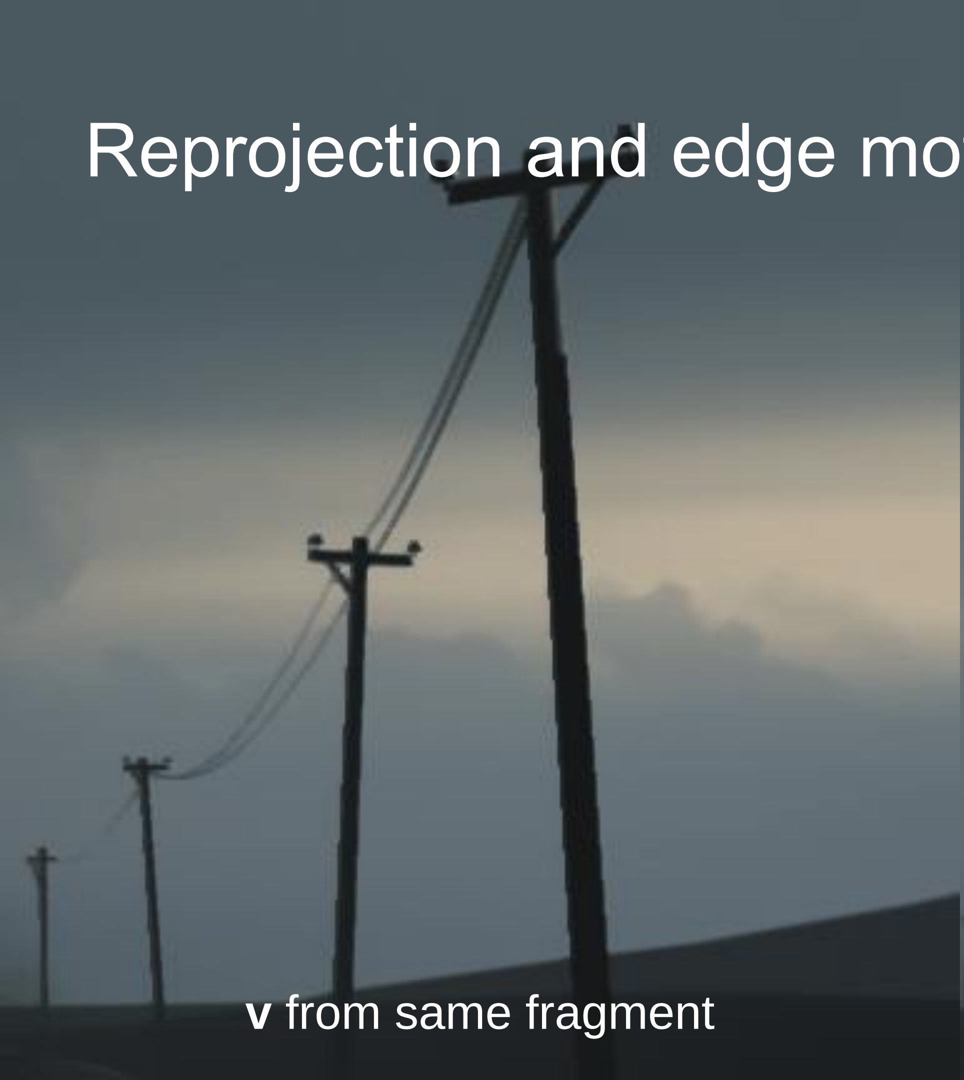
- reprojection step becomes read and subtract
  - **v** = sample(**buf**_velocity, **p**_uv)
  - **q**_uv = **p**_uv - **v**



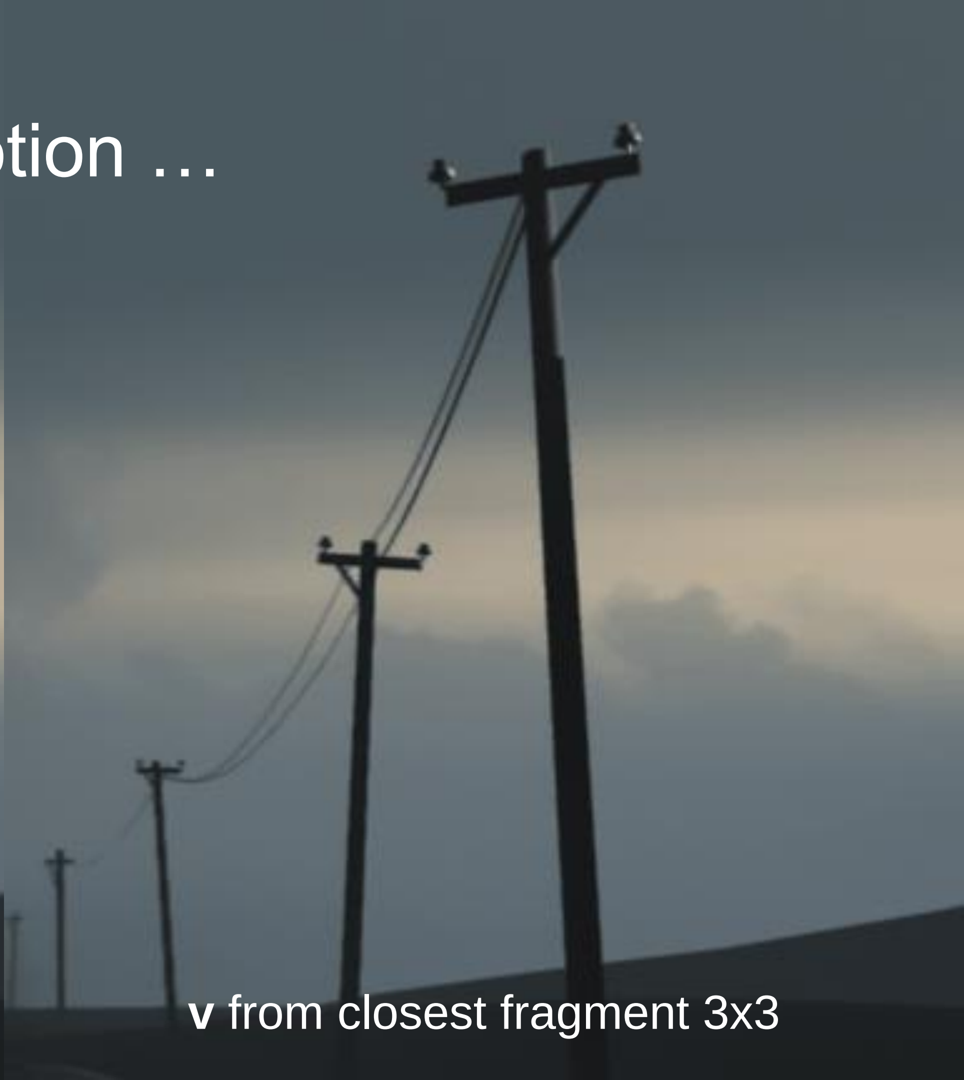
Reprojection and edge motion

- should add: we don’t actually sample **v** directly in **p**_uv
    - else out-of-edge fragments will not travel with occluder
- using velocity of closest (depth) fragment within 3x3 region
    - **v** = sample(**buf**_velocity, closest_fragment(**p**_uv).xy)
- similar to suggestion by [Karis14]
- result: nicer edges in motion

Reprojection and edge motion …

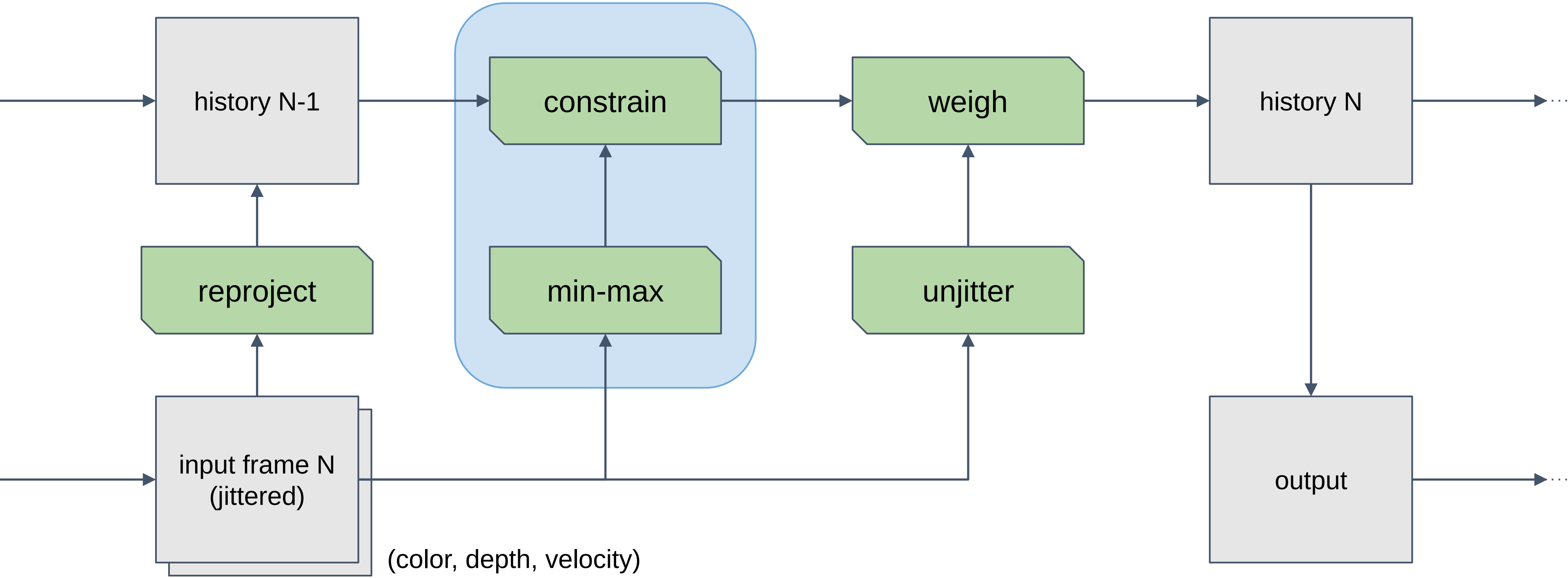


v from same fragment



v from closest fragment 3x3

Revisiting overview …



Constraining history sample

- history sample sometimes invalid
  - because of occlusion / disocclusion
  - because reprojection tracks only opaque
  - (… and we have lots of transparency)
- what if we trivially accept?
  - ghosting / smearing
  - example on the right
- have to constrain

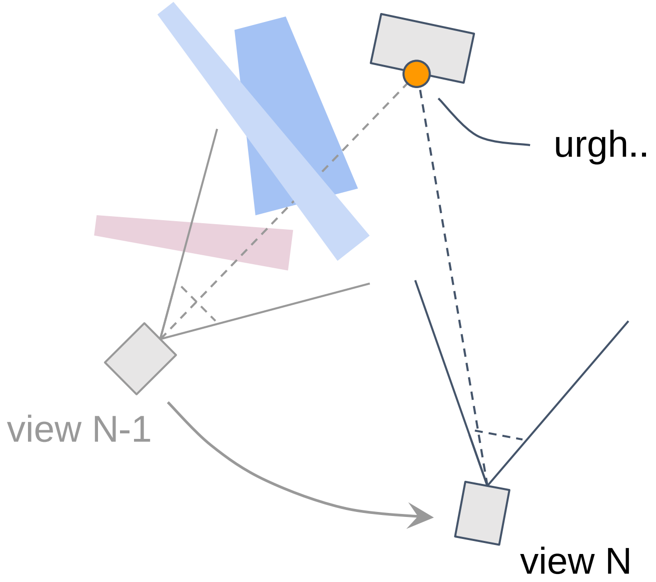
Constraining history sample …

- depth based rejection, velocity weighing [Sousa11] [Jimenez11]

- attempted this, found too fragile for our case
  - hard to eliminate ghosting with sliding threshold
  - (… in history, threshold itself is ghosting)

- also: transparency layers still smearing
  - didn't want to run temporal after opaque!
  - needed something else, so back to the brick wall

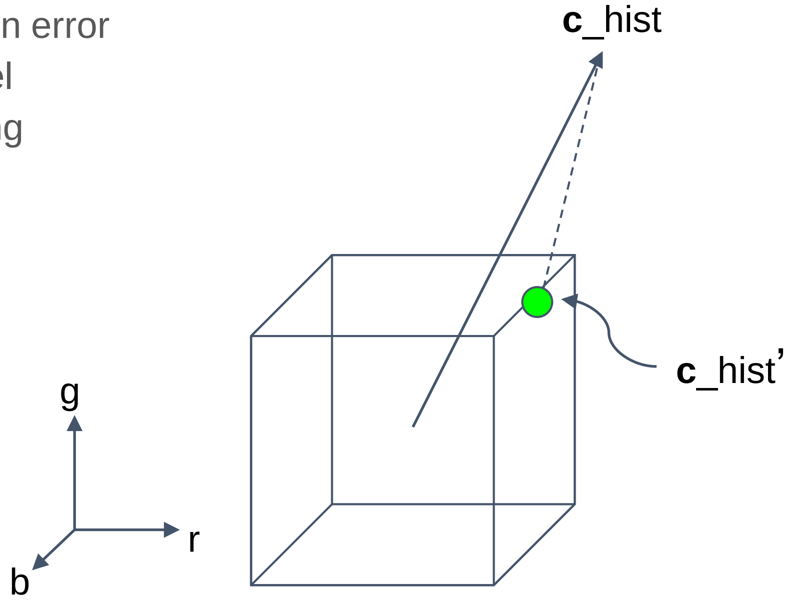
- neighbourhood clamping to the rescue.



Neighbourhood clamping 101

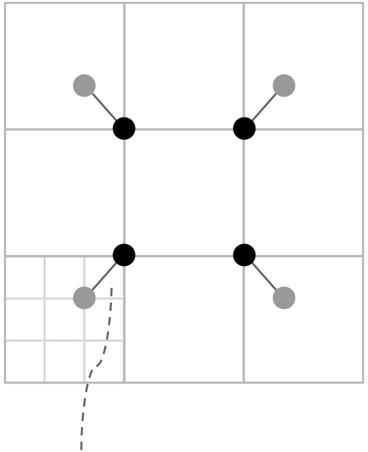
- [Sousa13] clamp history to neighbourhood of current sample
  - essentially per-frame upper bound on reprojection error
  - clamp color to min-max of 4 taps and center texel
  - big improvement in stability over velocity weighing

- pure color space operation
  - **cn**_min = sample_local_min(**buf**_color, **p**_uv)
  - **cn**_max = …// similar
  - **c**_hist’ = clamp(**c**_hist, **cn**_min, **cn**_max)



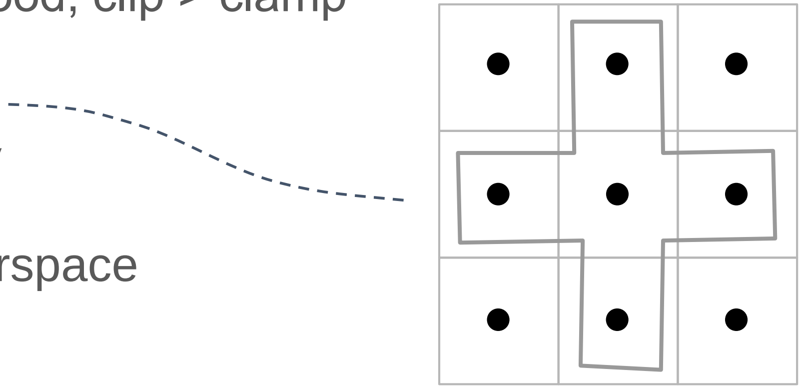
Neighbourhood clamping, first pass

- during production, the first implementation was a dynamic variation of the 4-tap approach
  - variable distance to 4 sample points, decided per-pixel
  - higher velocity ⇒ closer to center texel (strict on motion)
  - decent results without requiring per-object velocities
- we used this for about a year(!)
  - “early” first pass enabled artists to tailor effects and content
- later… decided to add per-object velocities
  - axed dynamic 4-tap approach in favor of image quality
  - switched to rounded 3x3 neighbourhood and clipping

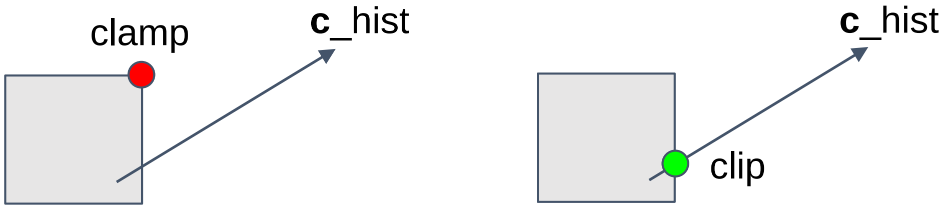


Neighbourhood clamping, now clipping

- [Karis14] larger “rounded” neighbourhood, clip > clamp
    - min-max of 3x3 neighbourhood
    - blend with min-max of 5 taps in ‘+’ pattern
    - bit more expensive, but better image quality

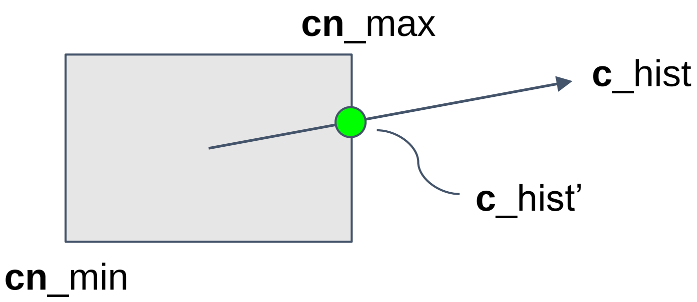


- clipping prevents clustering when colorspace is distant from history sample



A little note on line-box clipping

- proper line clip is “slow”
- we just clip towards aabb center
  - transform color vector into unit space
  - calc divisor and apply in clip space

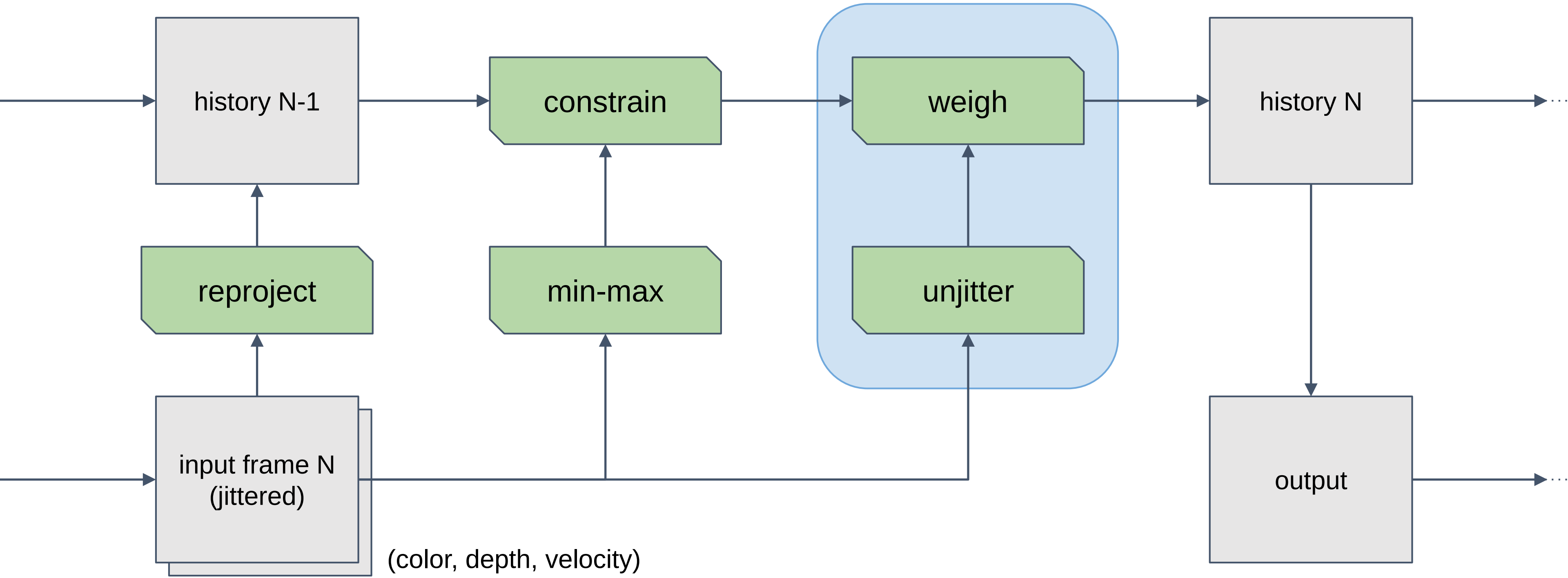



```
// note: clips towards aabb center + p.w
float4 clip_aabb(
    float3 aabb_min, // cn_min
    float3 aabb_max, // cn_max
    float4 p,        // c_in’
    float4 q)        // c_hist
{
  float3 p_clip = 0.5 * (aabb_max + aabb_min);
  float3 e_clip = 0.5 * (aabb_max - aabb_min);

  float4 v_clip = q - float4(p_clip, p.w);
  float3 v_unit = v_clip.xyz / e_clip;
  float3 a_unit = abs(v_unit);
  float ma_unit = max(a_unit.x, ax(a_unit.y,
                      a_unit.z));

  if (ma_unit > 1.0)
    return float4(p_clip, p.w) + v_clip / ma_unit;
  else
    return q;// point inside aabb
}
```

Revisiting overview …



Final blend, weighing constrained history

- weigh constrained history and unjittered input
  - **c**_hist' = …// constrained history sample
  - **c**_in' = sample(**buf**_color, unjitter(**p**_uv).xy)
  - **c**_feedback = lerp(**c**_in', **c**_hist', k_feedback)

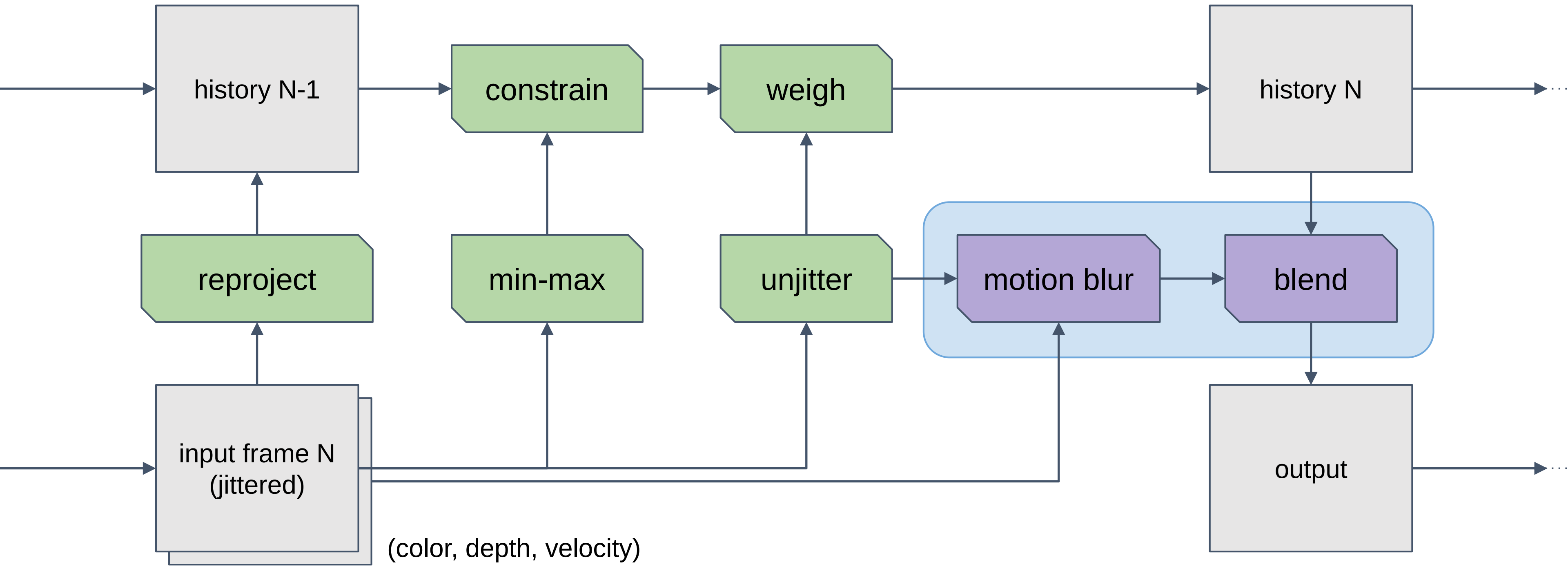
- update history buffer and copy to output
  - **rt**_history = **c**_feedback
  - **rt**_output = blit(**rt**_history)

- want to use high feedback factor to increase retention
  - beware of artefacts

Trailing artefacts

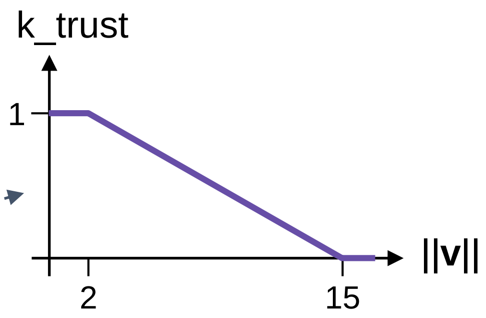
- history fragments can linger if none of their neighbours force them out
- observation: boy silhouette fragments
  - fast motion during turns, landings, etc.
- only distinct at artificially low resolution and framerate, wanted to remedy anyway
- *idea*: conceal with output-only motion blur
  - target history and output in same pass with MRT

Big picture 2.0: Adding motion blur to the mix …



Final blend with motion blur fallback

- update history buffer just like before
  - **rt**_history = **c**_feedback
- for output target, blend with motion blurred input
  - **c**_motion = sample_motion(**buf**_color, unjitter(**p**_uv), **v**)
  - **rt**_output = lerp(**c**_motion, **c**_feedback, k_trust)
  - k_trust = invlerp(15, 2, ||**v**||)// works well for us.



- forces transition to motion blur (no history!) for fast moving fragments
  - includes immediate neighbours, due to **v** relying on closest_fragment(…)

Final blend with motion blur fallback …

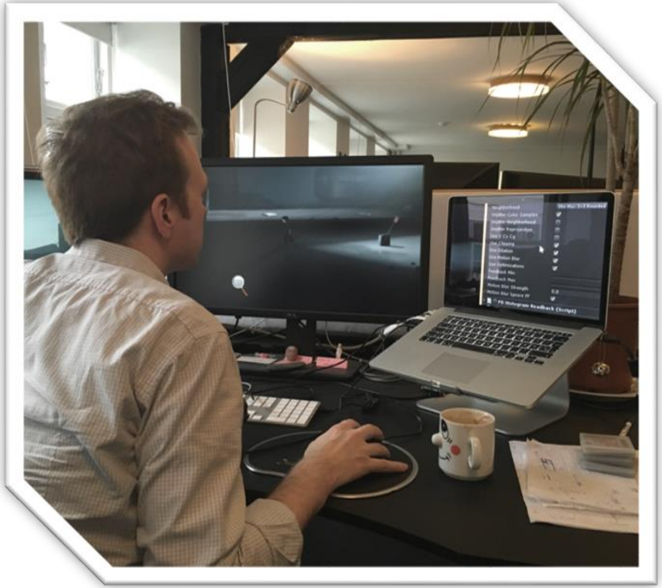
no motion blur fallback

with motion blur fallback

On picking a good sample distribution

- lots of trial and error, took practical approach
- … head close to screen, magnifying glass, obsessing over high contrast regions
- wanted to find good balance between quality and speed of convergence
- heuristics: side-scrolling game

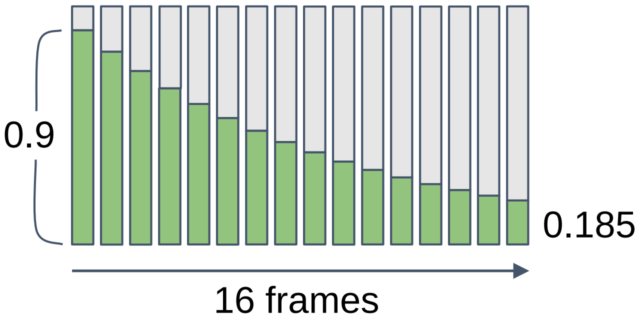
On picking a good sample distribution …



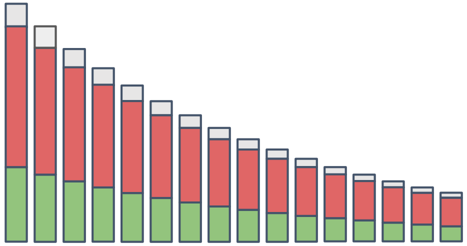
… inspecting many pixels

On picking a good sample distribution …

- using exponential history
  - samples weigh less over time
  - need high feedback factor
    - avoid visible cycle

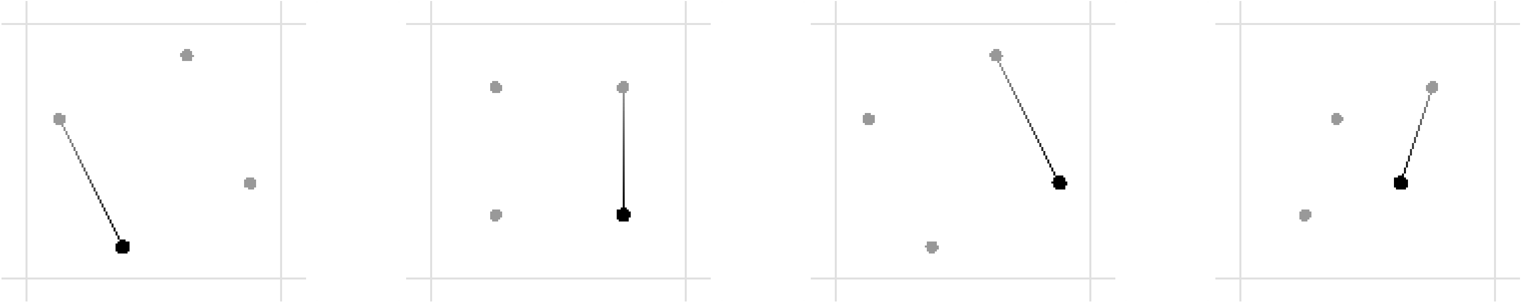


- nice to revisit same sub-pixel regions often
  - clamp/clip will compress tail
  - quickly return to that data

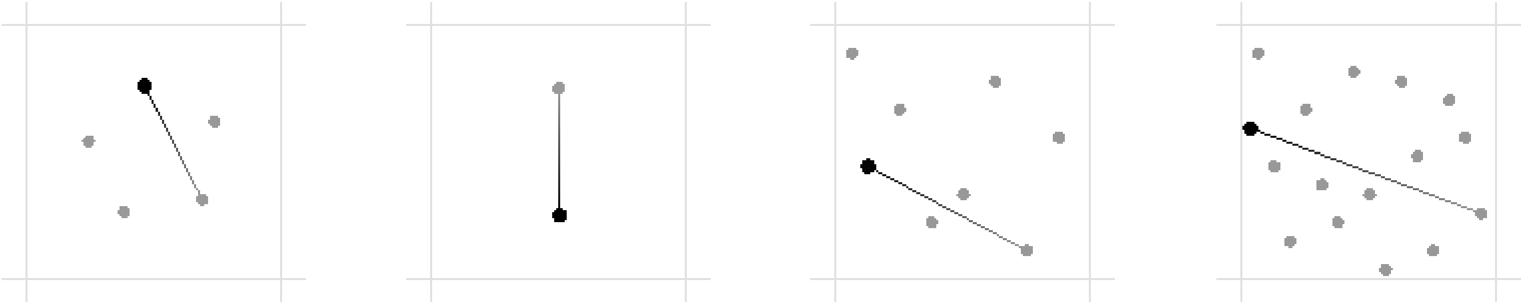


- initially used very few sample points …

Some of the sequences tested



uniform4 helix



halton(2,3) x8

halton(2,3) x16

Closing remarks on sample distributions

- while using 4-tap neighbourhood, “uniform 4 helix” was my favourite
  - short cycle ⇒ when sample is rejected, comes back to it quickly
  - not regular uniform 4
    - every step crosses horizontal center line
    - good at closing horizontal seams

- after moving to 3x3 and clipping, switched to 16 indices of halton(2,3)
  - much better coverage ⇒ much nicer edges
  - revisits sub-pixel regions quickly despite cycle length

- thought about motion-perpendicular pattern; needs more cooking time
  - perhaps squeeze along line of camera motion?

Summary of implementation

- jittering view frustum
  - 16 first samples of halton(2,3)
- generating velocity buffer
  - camera motion + dynamics (manual tagging, eurgh)
- reprojection using velocity
  - based on closest (depth) fragment
- neighbourhood clipping
  - center-clip to RGB min-max of “rounded” 3x3 region
- motion blur fallback
  - kicks in when $||\mathbf{v}|| > 2$, and full effect at 15
  - does not apply to history

temporal pass
~1.7ms on xb1
@ 1920x1080

Was greatly inspired by

- [Yang09] individual sub-pixel buffers, reprojection
  (Amortized Supersampling)
- [Sousa11] [Jimenez11] exponential history, velocity weighing
  (Anti-Aliasing Methods in CryENGINE 3)
- [Sousa13] neighbourhood clamping; “SMAA-1tx”
  (CryENGINE 3 Graphics Gems)
- [Karis14] clipping over clamping, YCoCg constraints
  (High Quality Temporal Supersampling)
- [McGuire12] motion blur reconstruction filter
  (A Reconstruction Filter for Plausible Motion Blur)

Temporal also has some really nice side-effects™

- stochastic everything
  - shadows
  - reflections
  - volumetrics

- discussed as part of talk about **INSIDE** rendering :) definitely go see it.

job@playdead.com

That's it! Thank you for coming.

Questions?

full source code: https://github.com/playdeadgames/temporal/

email me at lasse@playdead.com

@codeverses

Bonus slides

Clipping in YCoCg

- [Karis14] suggests clipping in YCoCg instead of RGB
- Intel has a nice page with illustrations and the transformations
- … ultimately not used for **INSIDE**
- our implementation still supports it