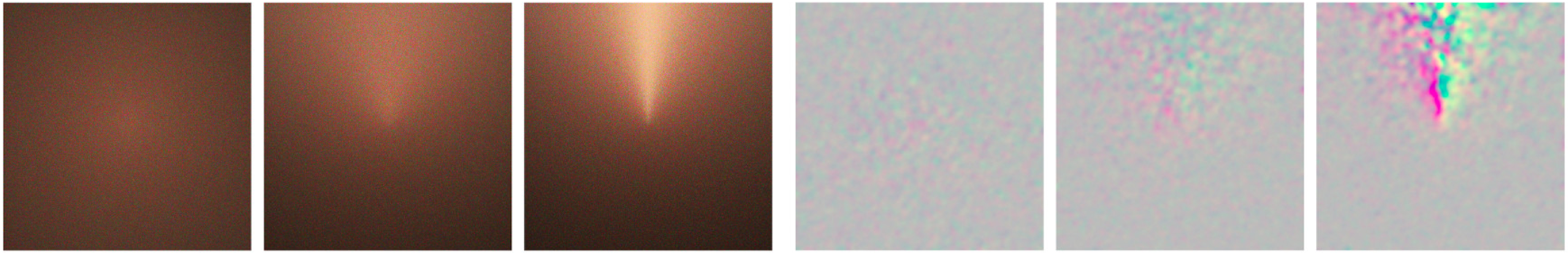


# Fast temporal reprojection without motion vectors

**Johannes Hanika** and Lorenzo Tessari
and Carsten Dachsbacher
Karlsruhe Institute of Technology

# motivation

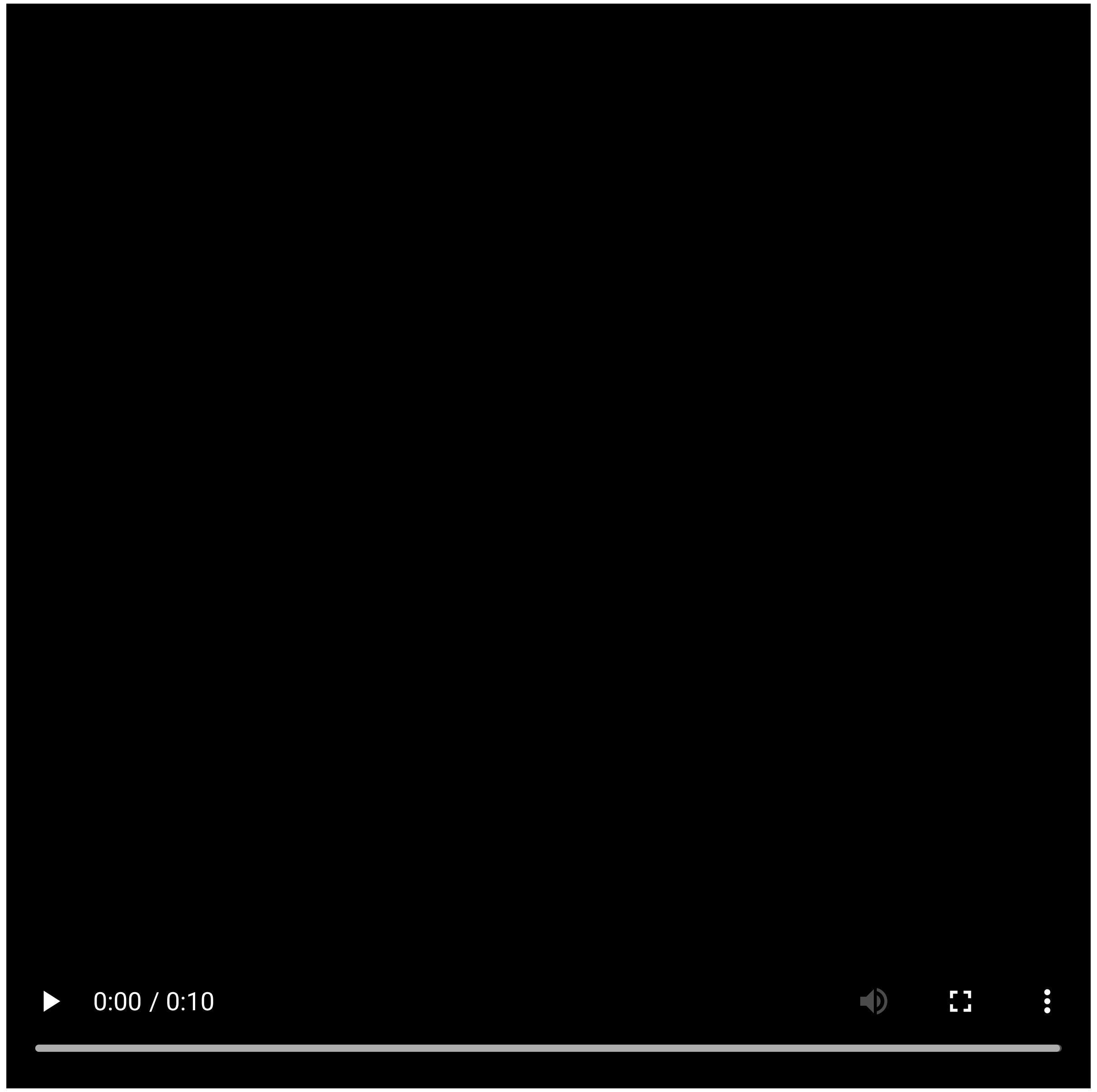
- a lot of coherence on the table, re-rendering similar things every few milliseconds!

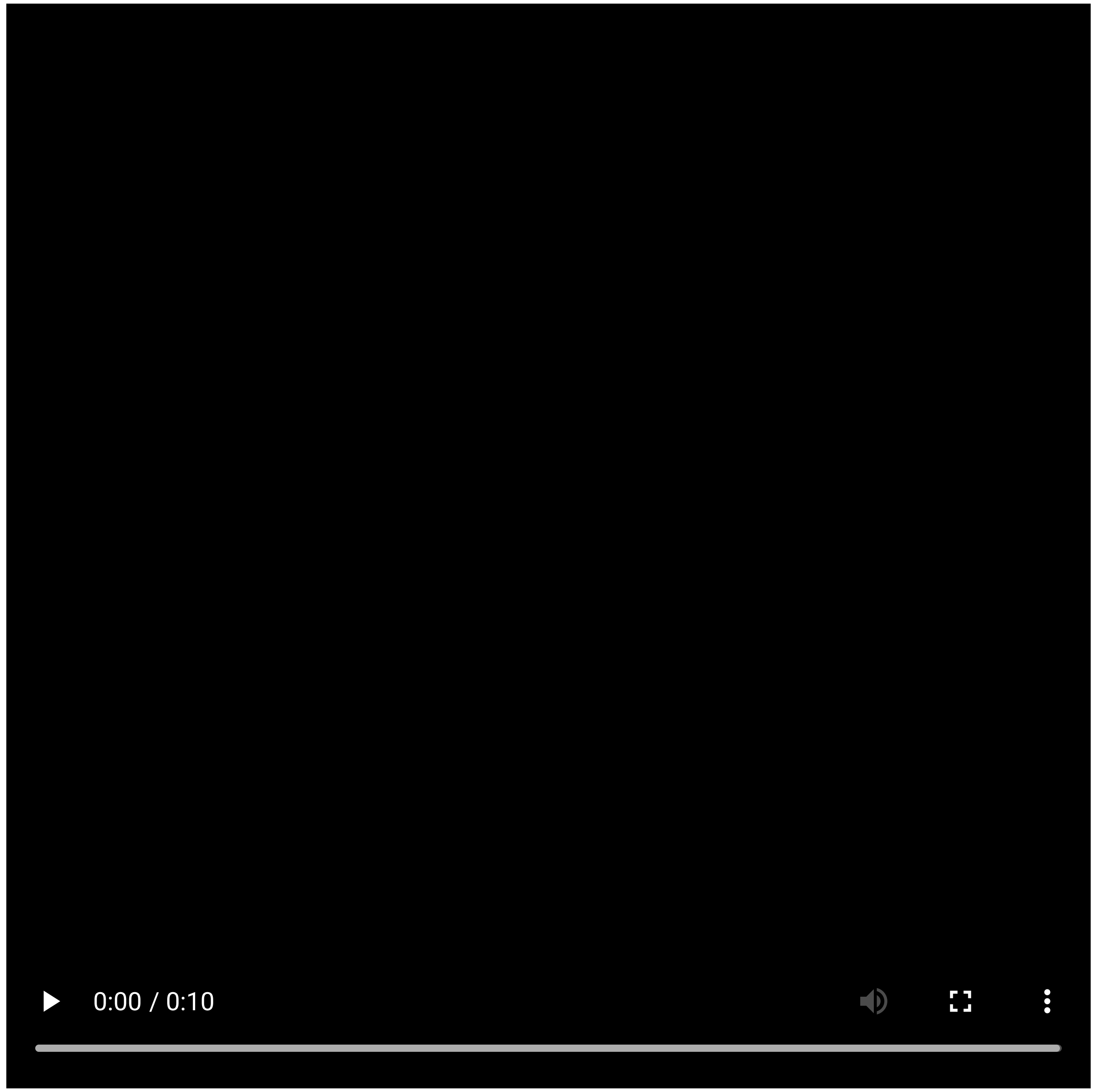
image: Schied et al. *Gradient Estimation for Real-Time Adaptive Temporal Filtering* 2018

- many methods use motion vectors, for instance
  - temporal anti aliasing (TAA)
  - screen space adaptive importance sampling (ReSTIR)
  - input to (recurrent) neural networks

# motion vectors

- motion is incoherent, contradicting, and hard to track analytically!

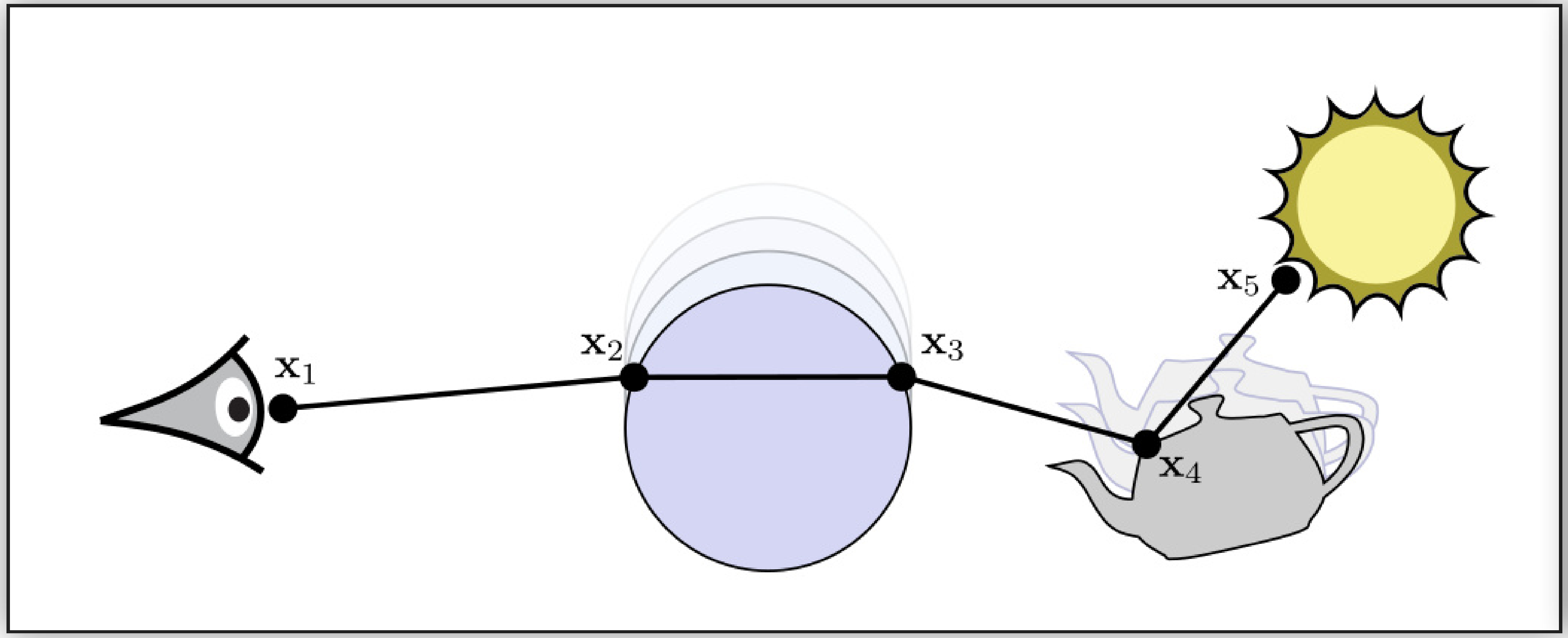


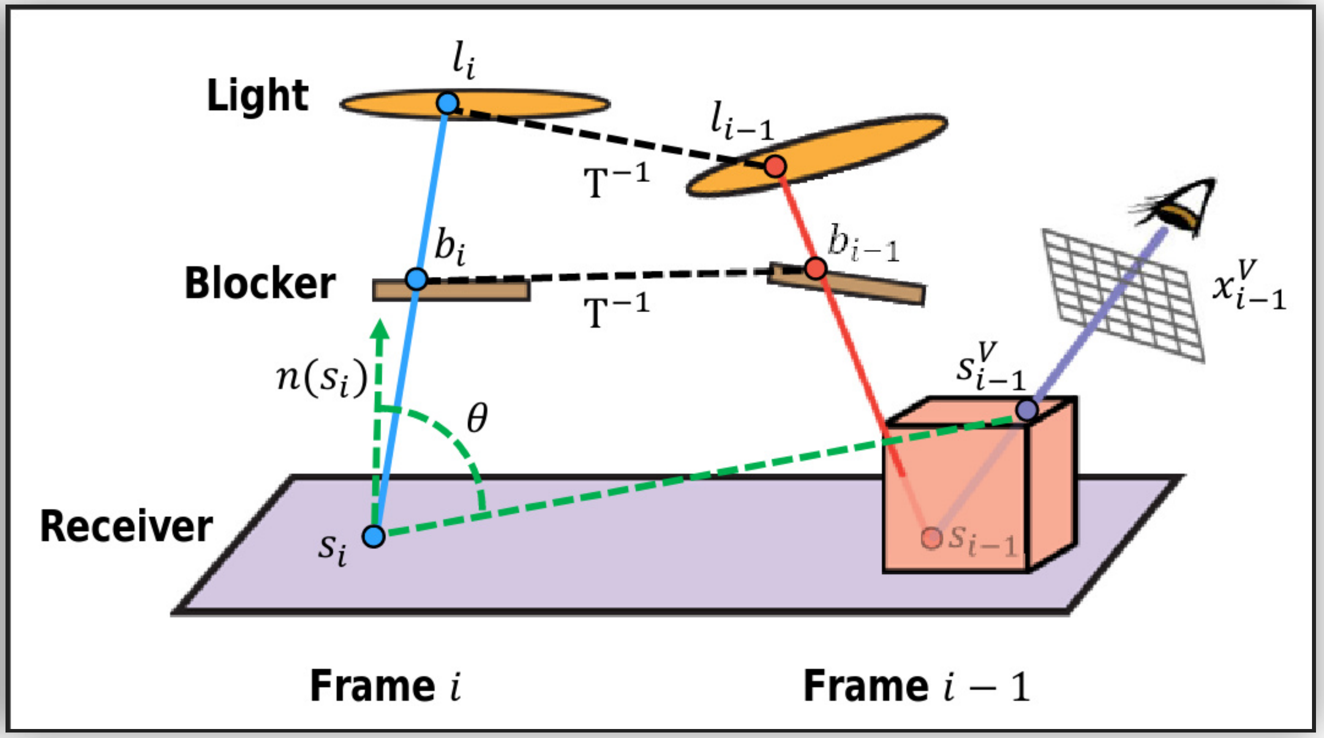



"Seascape" by Alexander Alekseev aka TDM

# related work: path space

- possible to track every path space effect separately (do we want that? does it scale?):
- Zimmer et al.
  Path-space Motion Estimation and Decomposition for Robust Animation Filtering
  (EGSR 2015)
- Zeng et al.
  Temporally Reliable Motion Vectors for Real-time Ray Tracing
  (CGF 2021)



# related work: image space

- Hasinoff et al.
  Burst photography for high dynamic range and low-light imaging on mobile cameras
  (TOG 2016)

- Darbon et al.
  Fast nonlocal filtering applied to electron cryomicroscopy
  (Biomedical imaging 2008)

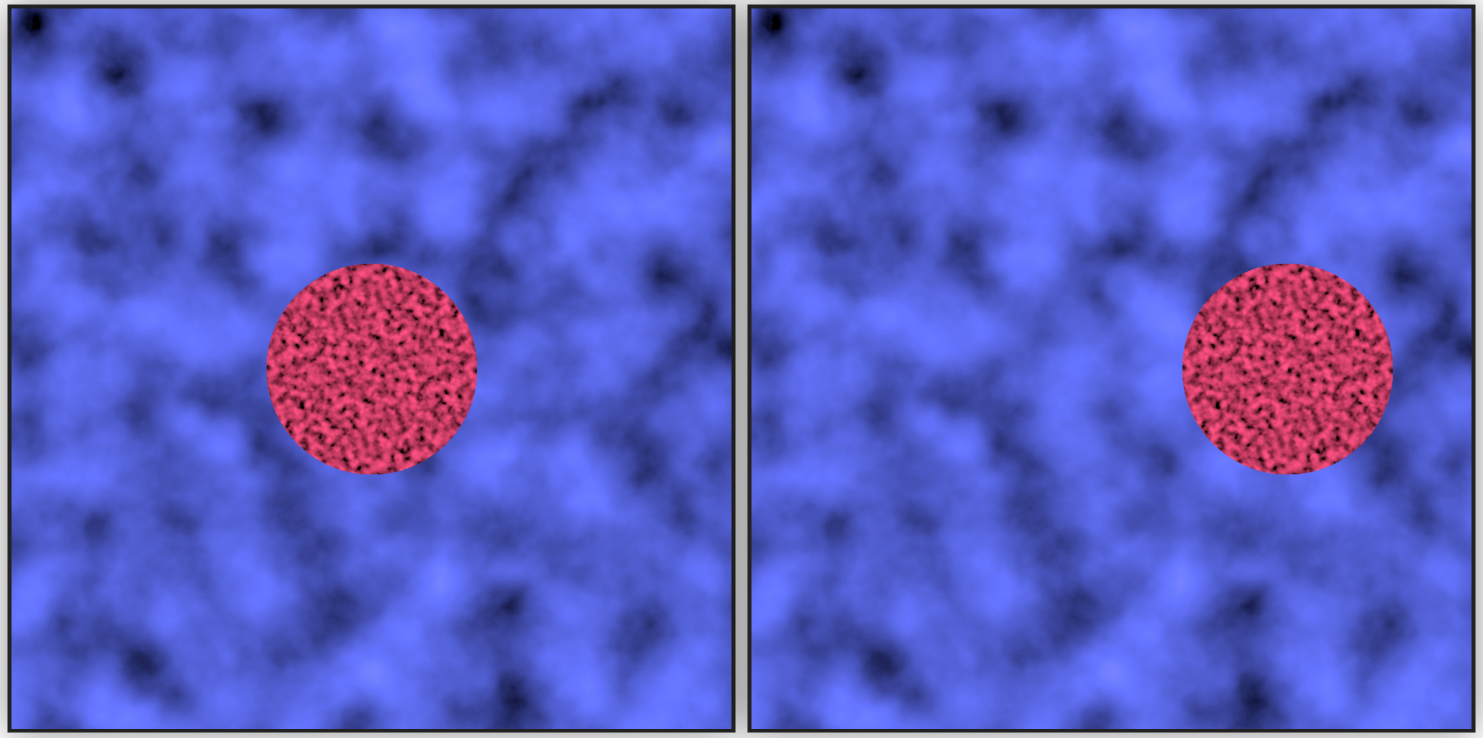

Burst photography for high dynamic range and low-light imaging on mobile cameras
Samuel W. Hasinoff, Dillon Sharlet, Ryan Geiss, Andrew Adams, Jonathan T. Barron, Florian Kainz, Jiawen Chen, Marc Levoy
Google Research


FAST NONLOCAL FILTERING APPLIED TO ELECTRON CRYOMICROSCOPY
Jérôme Darbon, Alexandre Cunha, Tony F. Chan, Stanley Osher, Grant J. Jensen

- we'll combine ideas from these last two to arrive at a fast technique

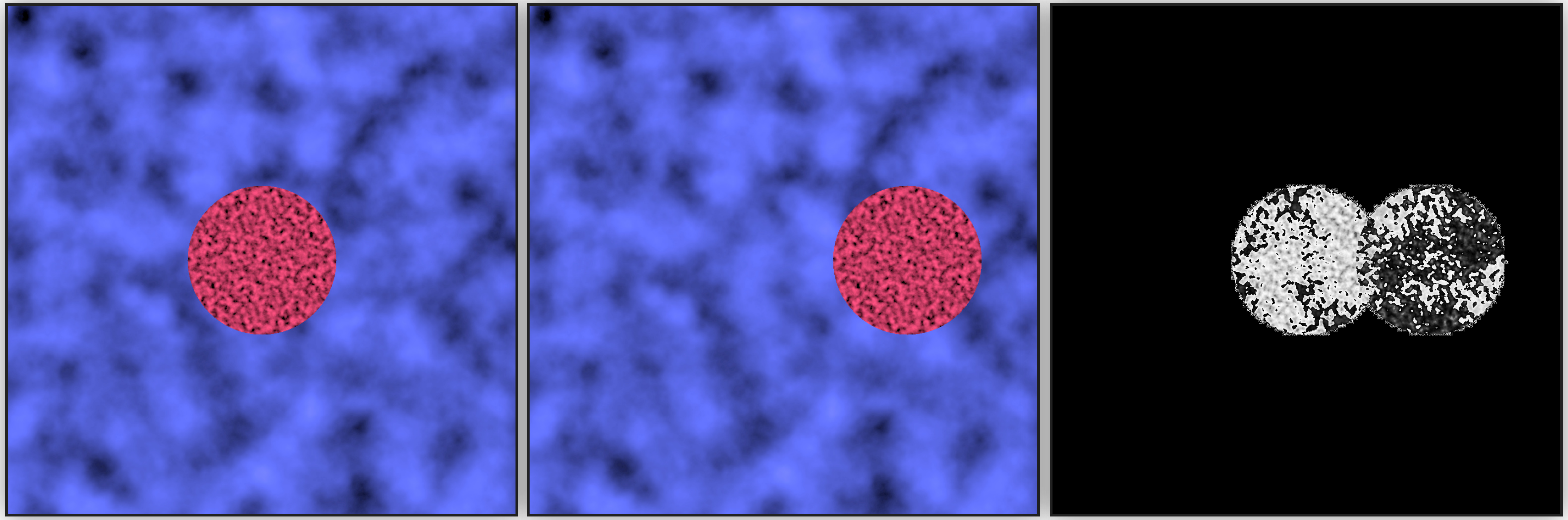
# find best displacement vector for each pixel

- shift images against each other and take difference [Darbon et al. 2008]



# find best displacement vector for each pixel

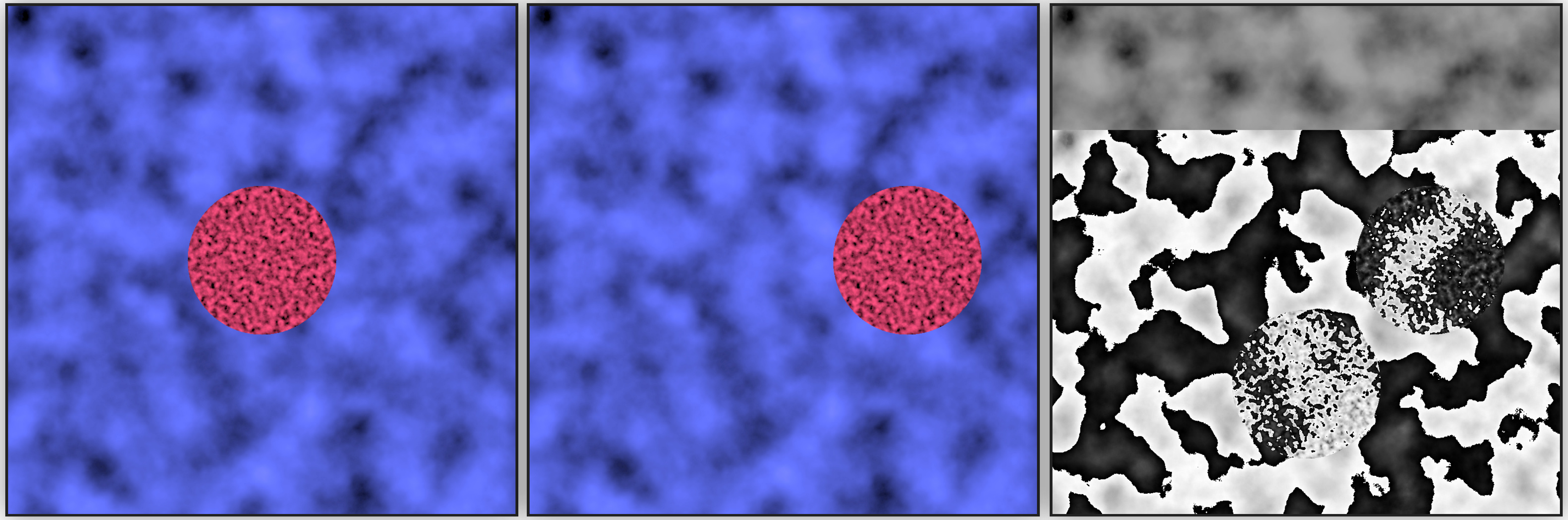
- shift images against each other and take difference $s = (0, 0)$



- black is better: zero shift looking pretty good for the static parts!

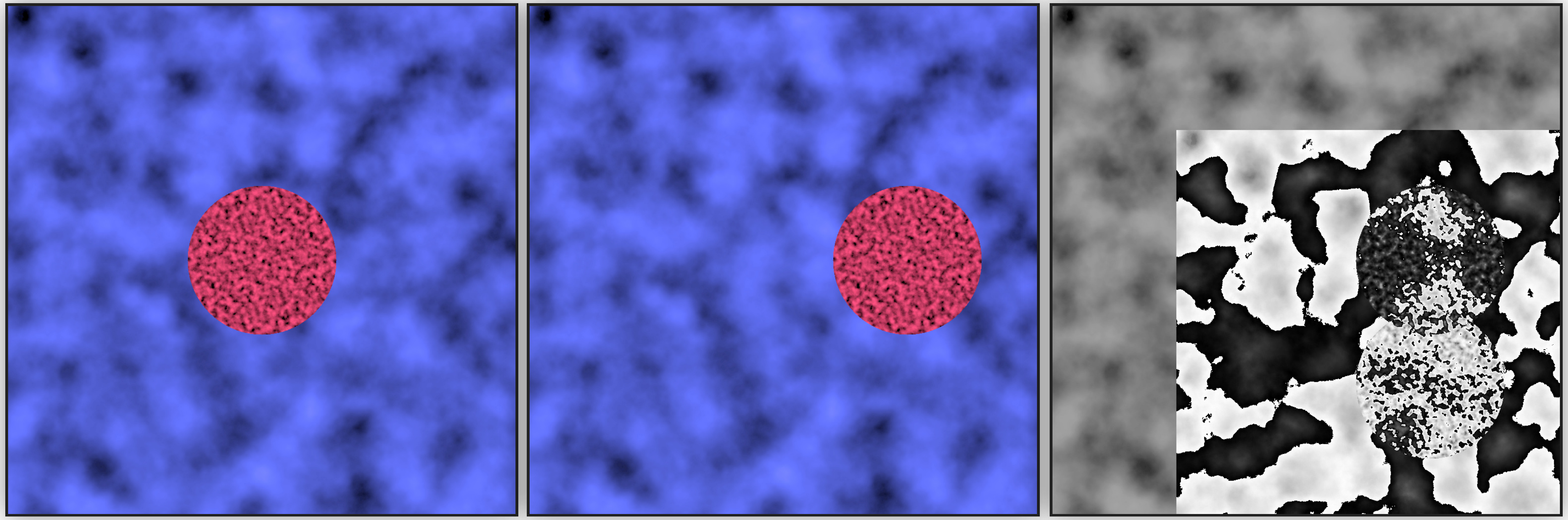
# find best displacement vector for each pixel

- shift images against each other and take difference $s = (0, 250)$



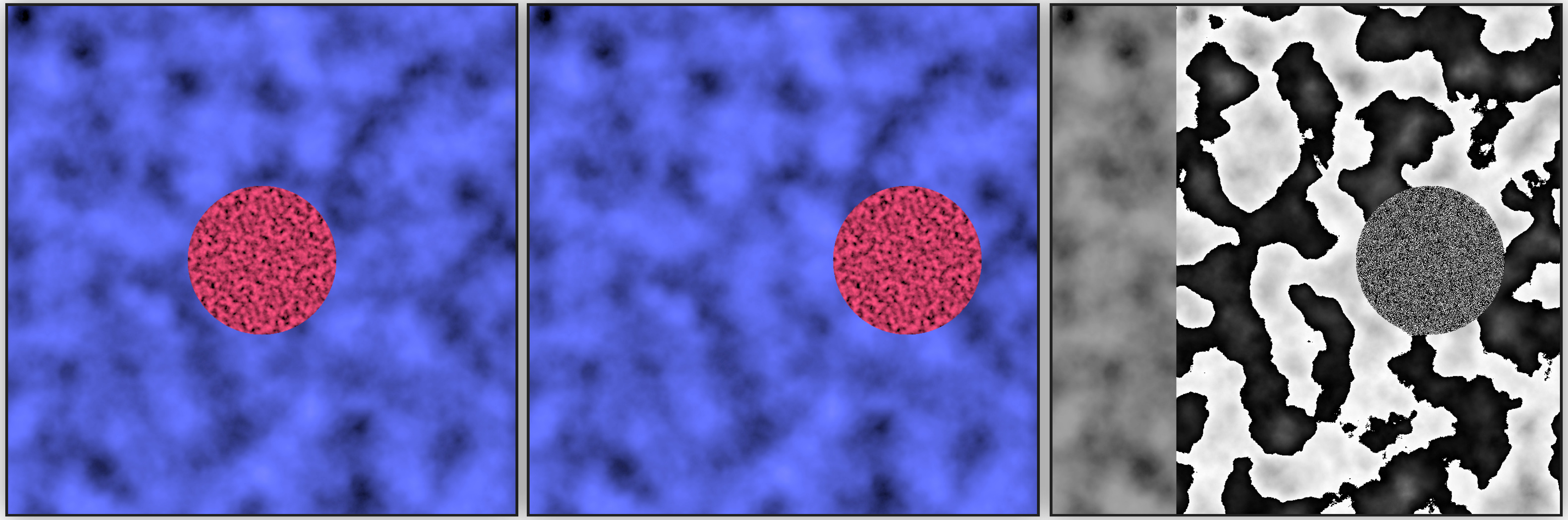
# find best displacement vector for each pixel

- shift images against each other and take difference $s = (250, 250)$



# find best displacement vector for each pixel

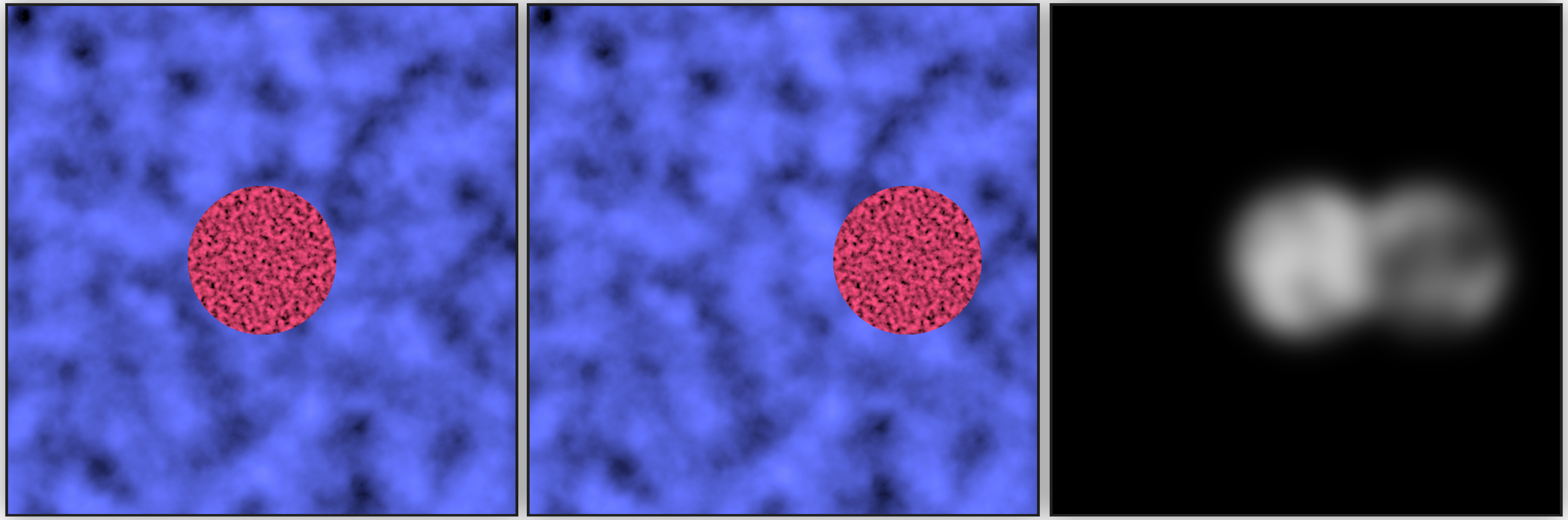
- shift images against each other and take difference $s = (250, 0)$



- this could be good for the moving circle
- input is noisy, pixel difference doesn't tell us much
- use patch-based difference, equivalent to blurring the difference images [Darbon et al. 2008]

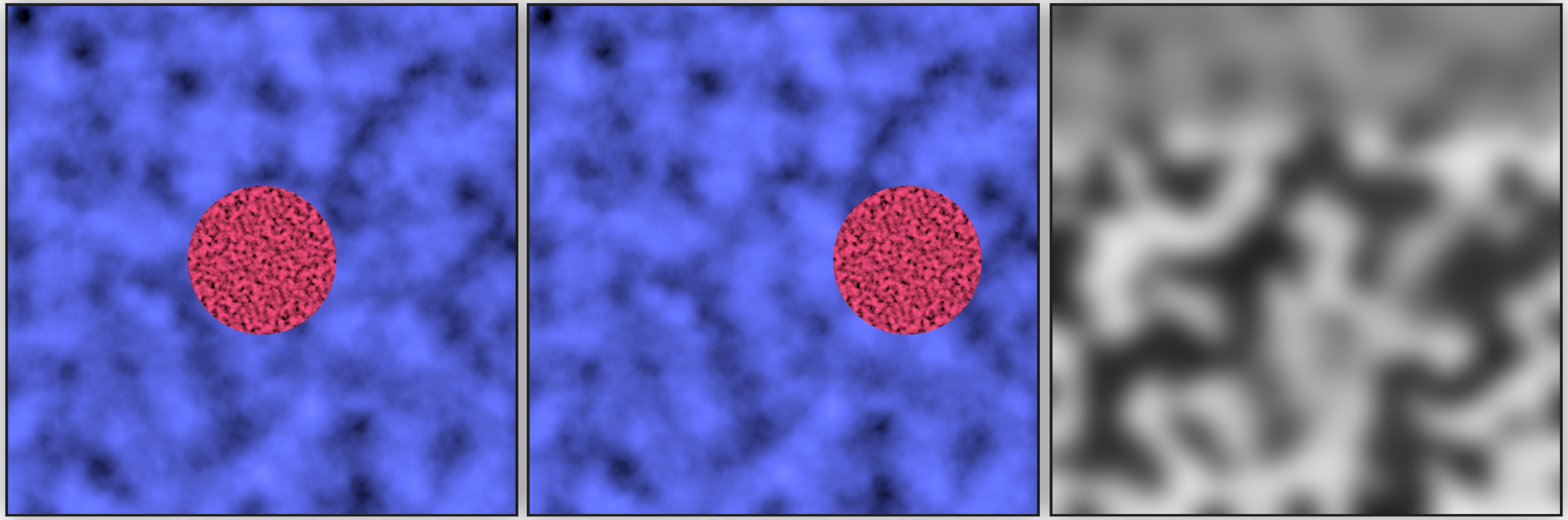
# find best displacement vector for each pixel

- shift images against each other and take difference $s = (0, 0)$



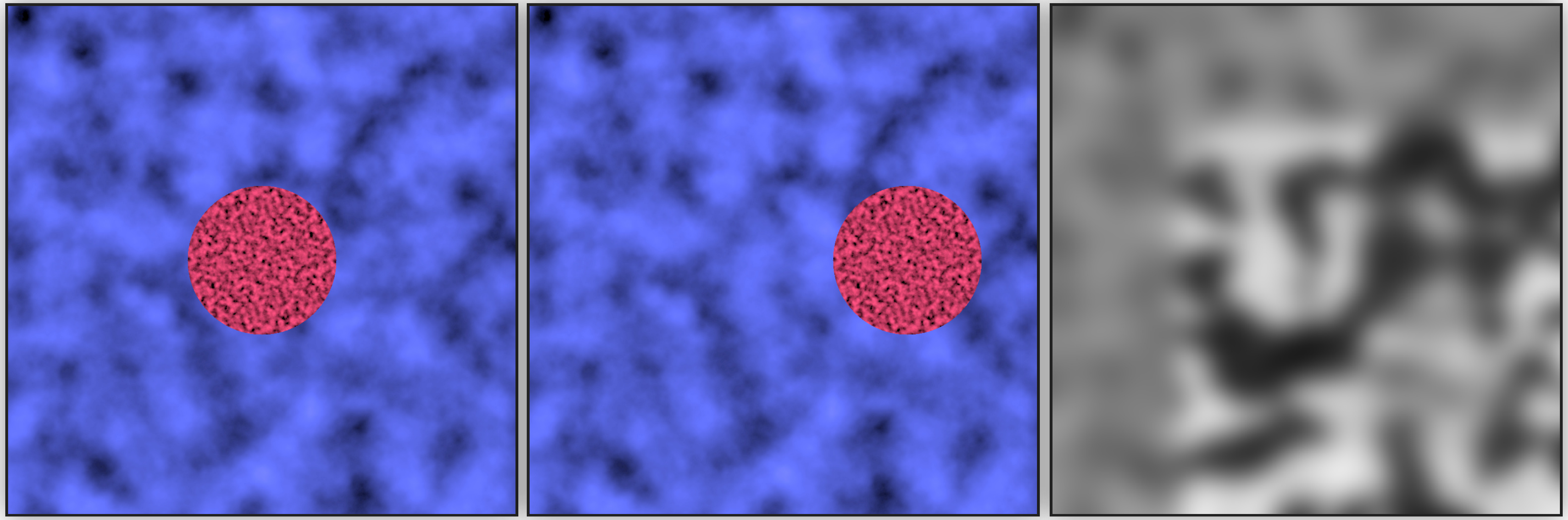
# find best displacement vector for each pixel

- shift images against each other and take difference $s = (0, 250)$



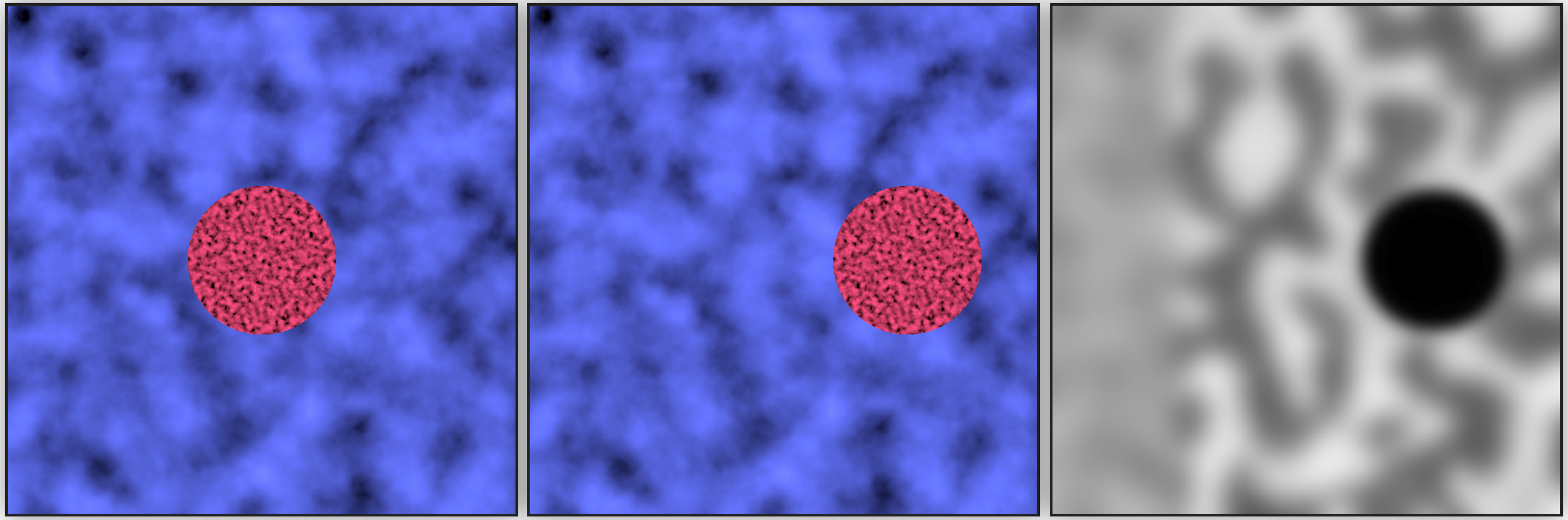
# find best displacement vector for each pixel

- shift images against each other and take difference $s = (250, 250)$



# find best displacement vector for each pixel

- shift images against each other and take difference $s = (250, 0)$

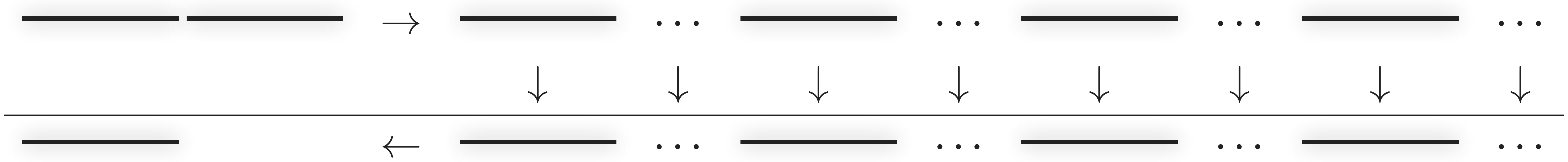


# find best displacement vector for each pixel

- which shift vector $s$ has the smallest error for each pixel?

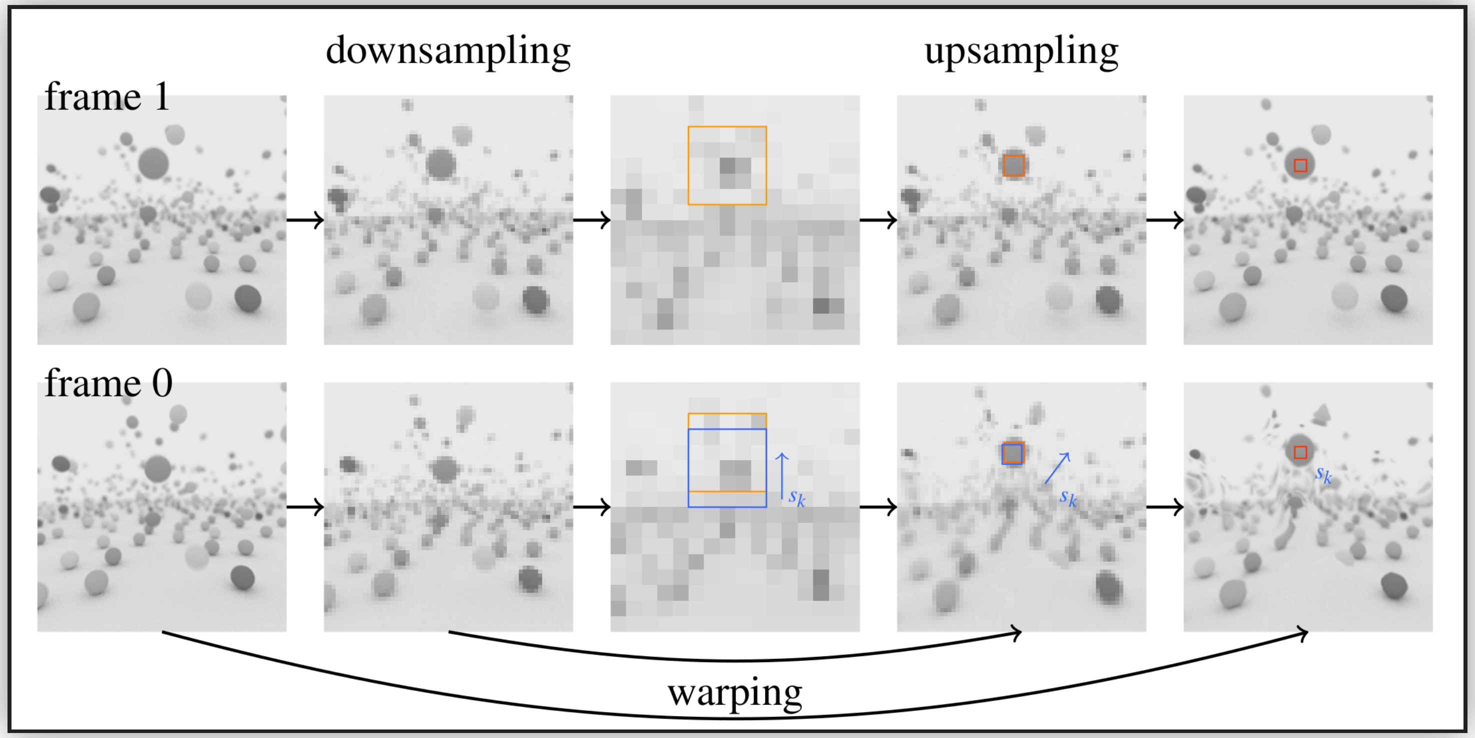
| . | . | . | . |
|---|---|---|---|
| $s = (0, 0)$ | $s = (0, 250)$ | $s = (250, 0)$ | $s = (250, 250)$ |

# implementation



# acceleration: hierarchical matching

- enables smaller windows $|S|$ (we use 5x5)




- enforces coherence/smoothness (a good thing?)
- offset vectors limited to one of the discrete shifts we tested!

# smooth upsampling of offsets [Hasinoff et al. 2016]

- generate "landscape" of distance values in 5x5 grid (for each pixel)
- fit quadric, compute minimum
- subpixel offsets from low res matching buffers!

| | $s_x = -2$ | $s_x = -1$ | $s_x = 0$ | $s_x = 1$ | $s_x = 2$ |
|---|---|---|---|---|---|
| $s_y = -2$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $s_y = -1$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $s_y = 0$ | $\cdots$ | $\cdots$ | | $\cdots$ | |
| $s_y = 1$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $s_y = 2$ | $\cdots$ | $\cdots$ | | $\cdots$ | |

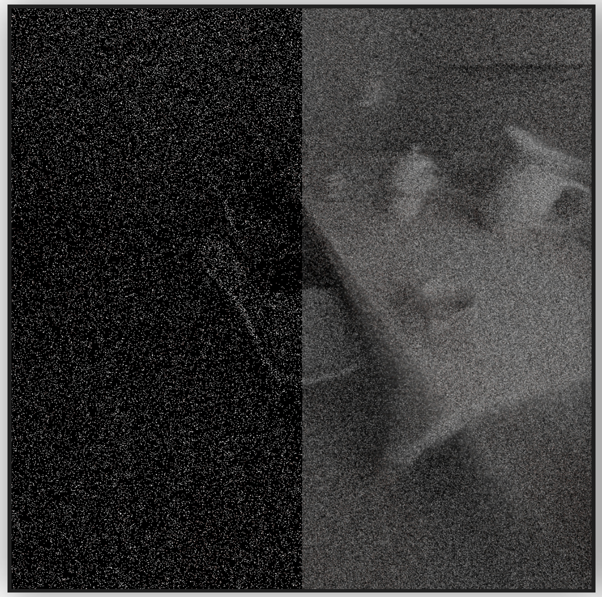
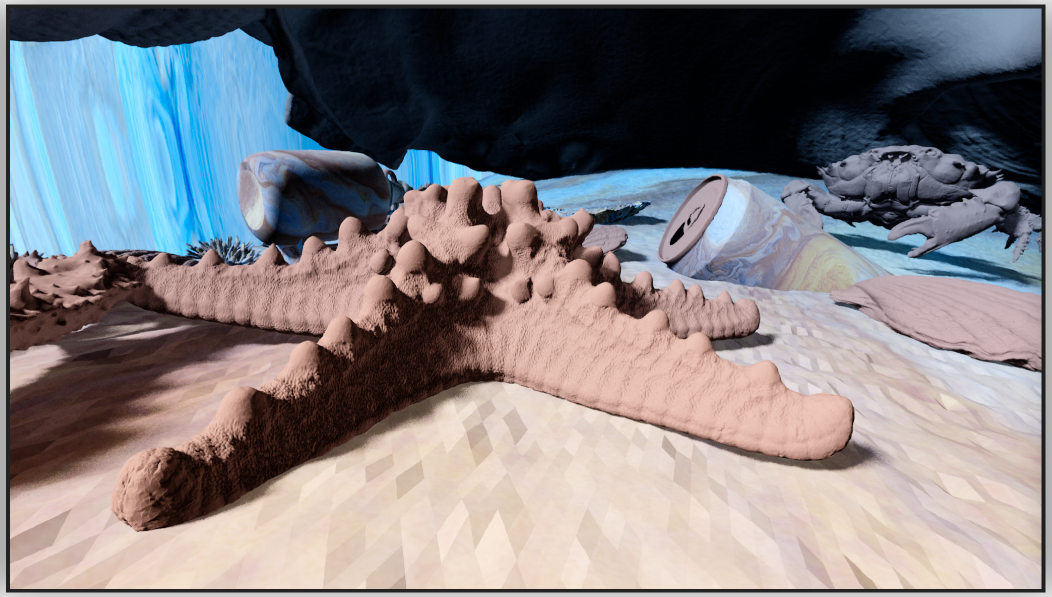
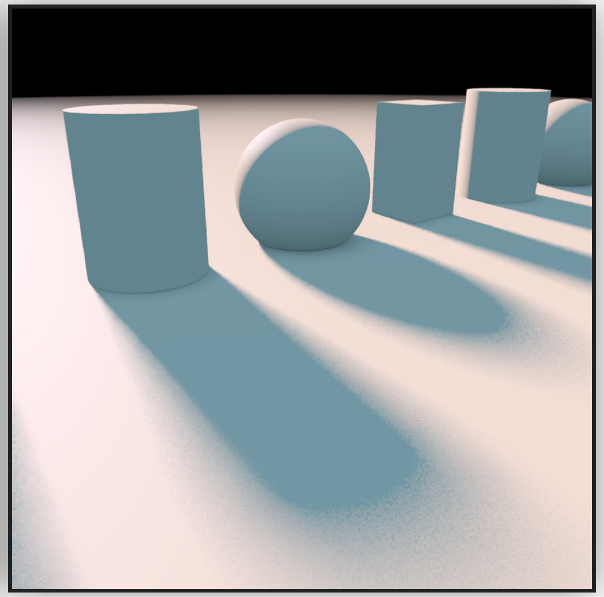
.

# blur

- fast blur on array of textures?
  - we use Kawase style blurs (on low res buffers)
  - requirement: need to preserve extrema of non-downsampled blur!
  - see our source code for current version (I'm sure this can be improved)

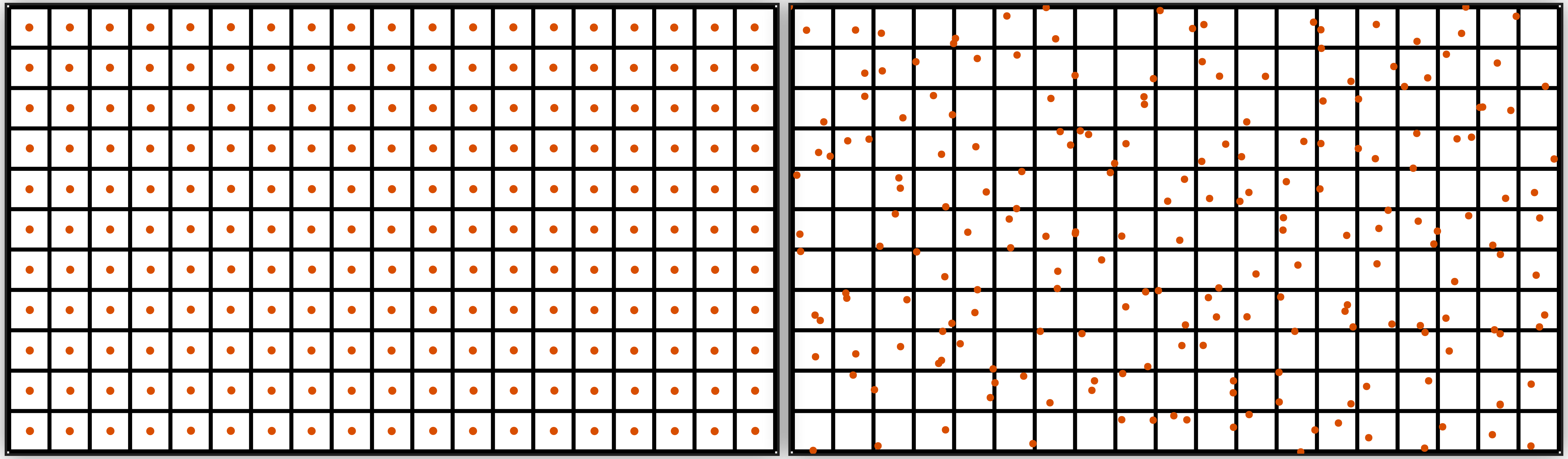
# applications

- this work is only dealing with offset/motion vector detection!
- no image merging/denoising/TAA!
- lots of applications in the paper

# one more result (not in the paper)

- TAA for ray tracing with per-pixel filter importance sampling
- sample 3x3 Blackman/Harris

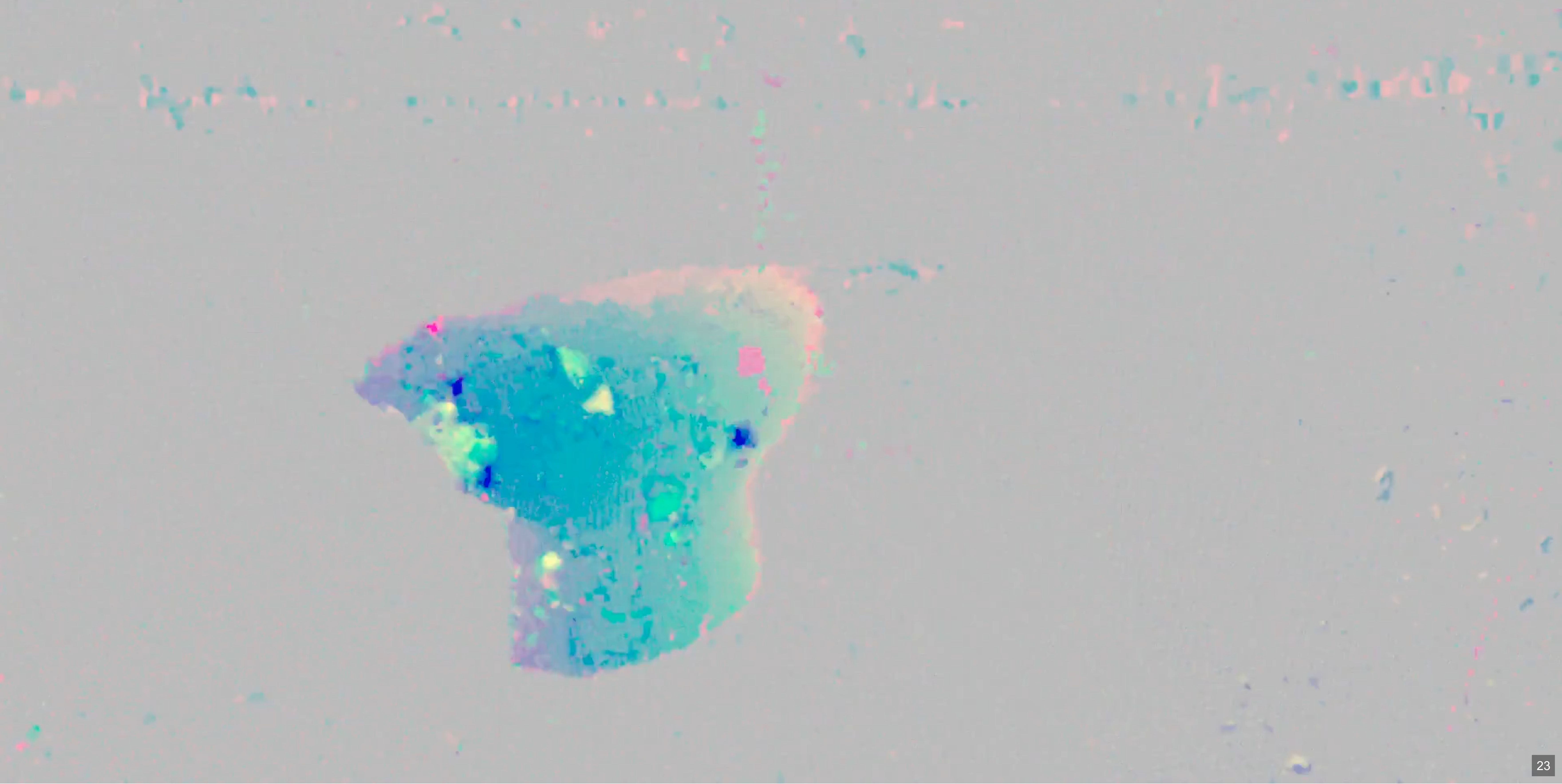


```
vec2 res = vec2(cos(rand.y*M_PI*2.0), sin(rand.y*M_PI*2.0));
float r = 0.943404 * asin(0.636617 * asin(sqrt(rand.x))); // surprisingly good fit to inverse cdf
return res * r;
```

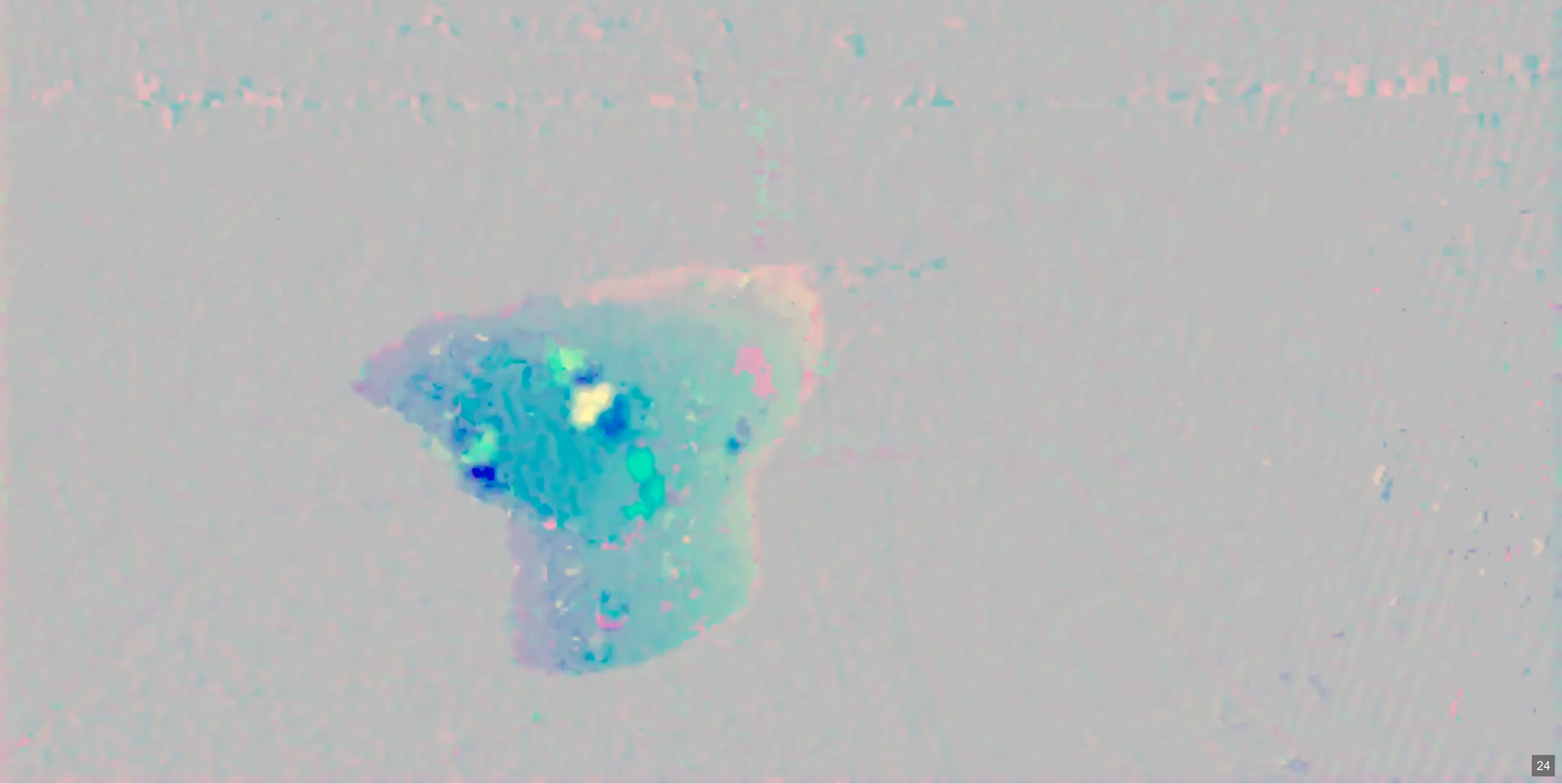
# let's look at some videos

- setting:
  - matching on AOV buffers
  - 5x5 blur
  - 1920x1080
  - RTX 2080Ti
- will show two variants: full res and 4x4 subsampled input

# offsets without subsampling: 5ms



# offsets 4x4 subsampling: 0.5ms



# TAA without subsampling

# TAA with 4x4 subsampling

# accuracy vs. performance

- the 4x4 has *better* edges because it doesn't undo the pixel filter importance sampling!

1x1

# accuracy vs. performance

- the 4x4 has *better* edges because it doesn't undo the pixel filter importance sampling!

4x4

# conclusion

- there may be inconsistent motion vectors even for one pixel
  - objects
  - shadows
  - reflections
  - procedural noise / water / clouds / fire ..

- coherence often works/helps to speed things up
  - work on downsampled buffer!
  - careful: don't enforce coherence where there is none

# conclusion contd.

- presented a relatively fast technique for "motion vectors" in image space
  - 0.5ms / 1080p / 2080Ti (5x5 blur, 4x4 subsampling)
  - can still compute motion vectors in layers as [Zeng et al. 2021]
    at least don't have to manually run after each effect

- future work
  - how to optimally combine aligned frames for denoising? AMD FSR 2.0?
  - optimise more for speed!

# thank you for listening!

- code on github
  https://github.com/hanatos/vkdt/tree/master/src/pipe/modules/align
- updated occasionally