Accelerating Real-Time Shading with Reverse Reprojection Caching

Diego Nehab[1] Pedro V. Sander[2] Jason Lawrence[3]

Natalya Tatarchuk[4] John R. Isidoro[4]

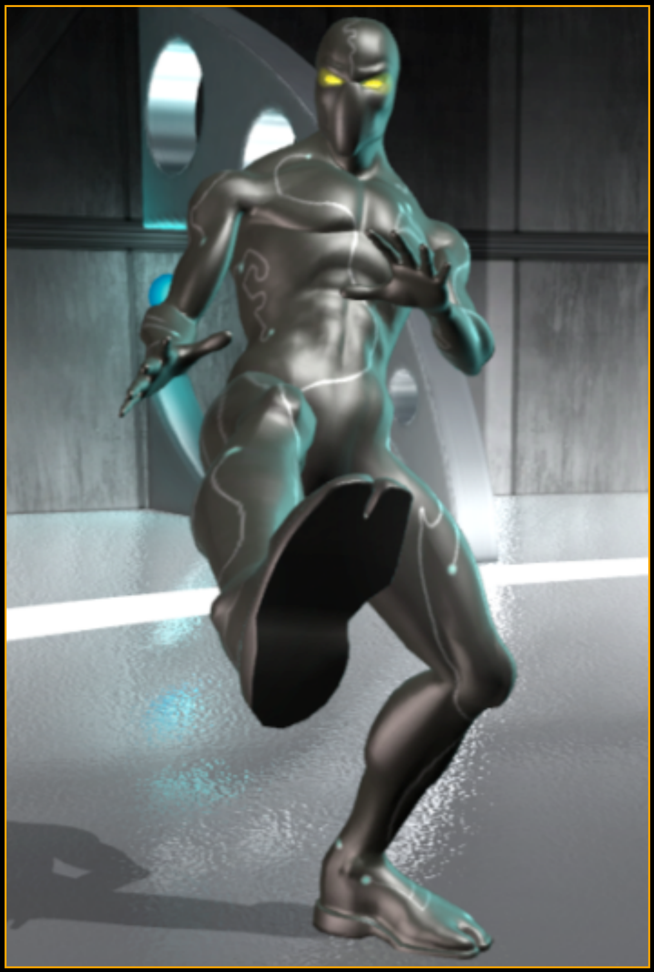
[1]Princeton University

[2]Hong Kong University of Science and Technology

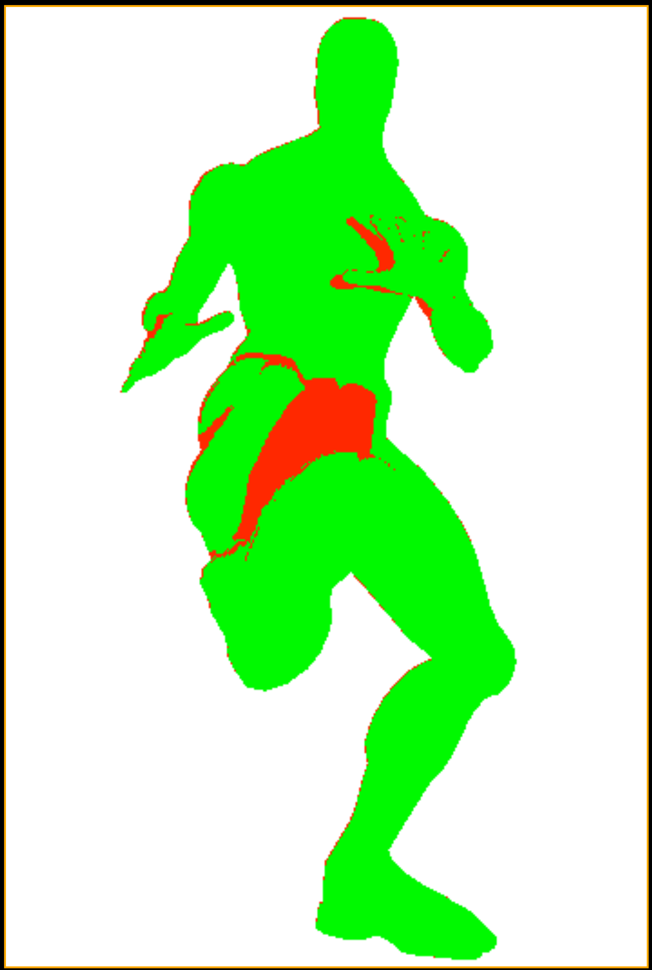
[3]University of Virginia

[4]Advanced Micro Devices, Inc.

Motivation



Motivation



Previous work

- Dedicated hardware
  - Address Recalculation Pipeline [Regan and Pose 1994]
  - Talisman [Torborg and Kajiya 1996]
- Image based rendering
  - Image Warping [McMillan and Bishop 1995]
  - Post Rendering 3D Warp [Mark et al. 1997]

Previous work

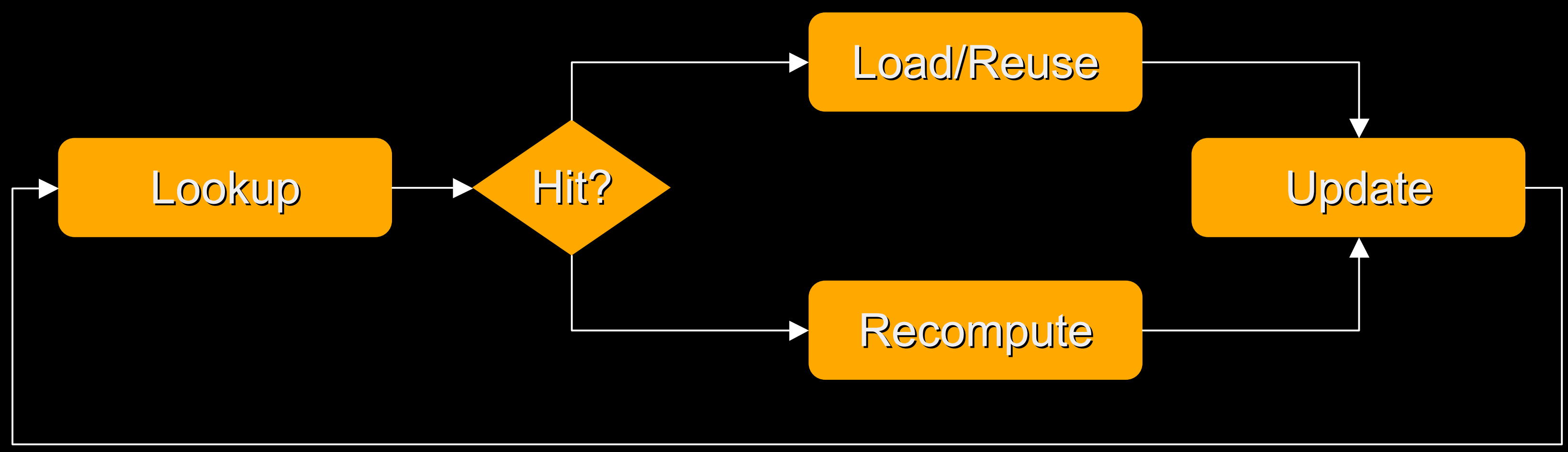
- Interactivity for expensive renderers
  - Frameless rendering [Bishop et al. 1994]
  - Render Cache [Walter et al. 1999]
  - Holodeck/Tapestry [Simmons et al. 1999/2000]
  - Corrective Texturing [Stamminger et al 2000]
  - Shading Cache [Tole et al. 2002]

Our approach

- Explore coherence in real-time rendering

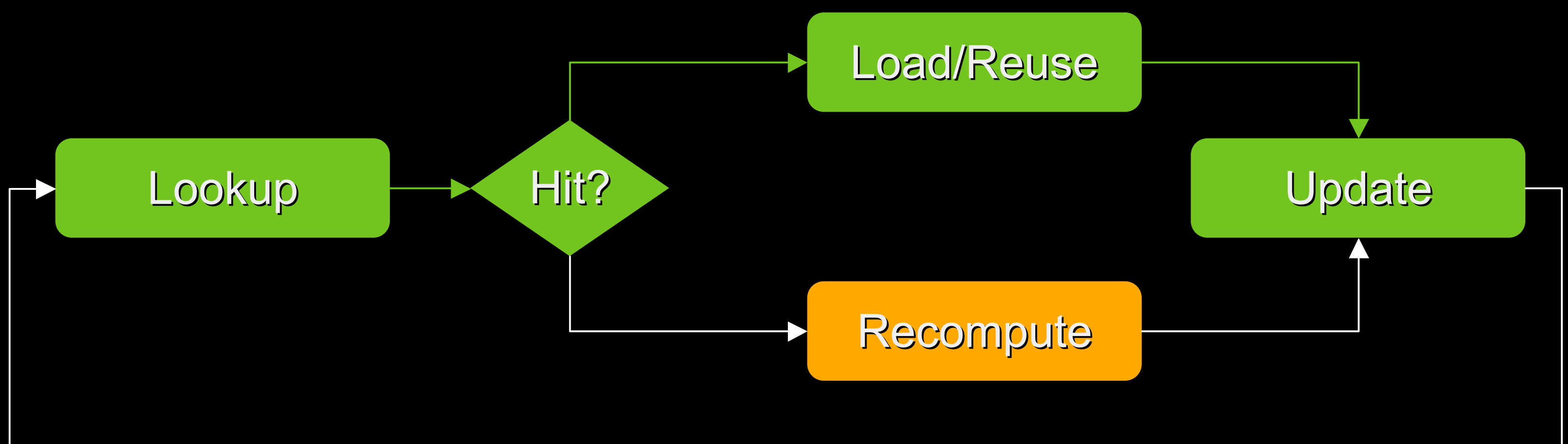
Our approach

- Explore coherence in real-time rendering



Requirements

- Load/reuse path must be cheaper
- Cache hit ratio must be high
- Lookup/update must be efficient

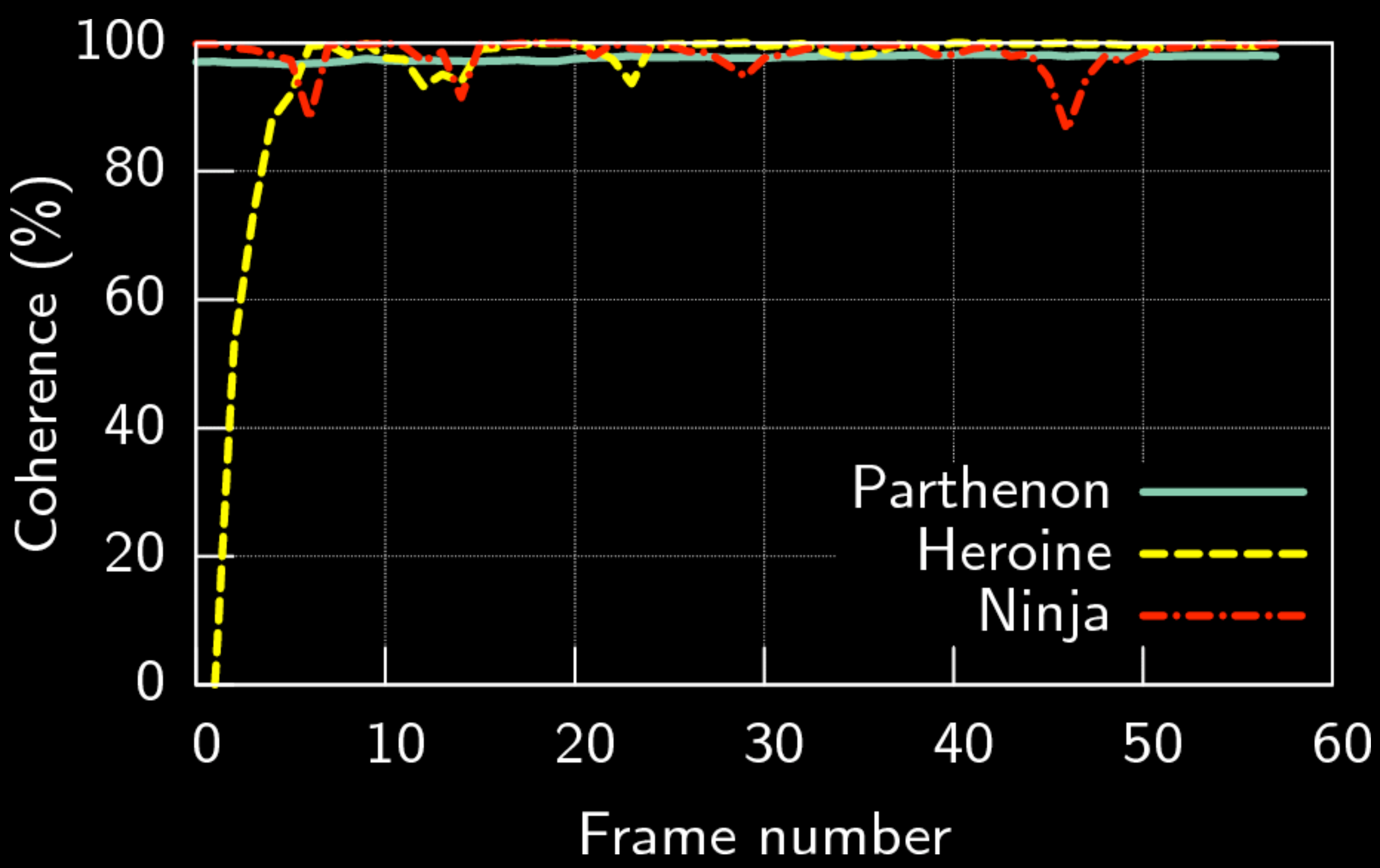


First insight

- Cache *only* visible surface fragments
  - Use screen space buffer to store cache
  - Output sensitive memory
  - Keep everything in GPU memory
  - Leverage hardware Z-buffering for eviction

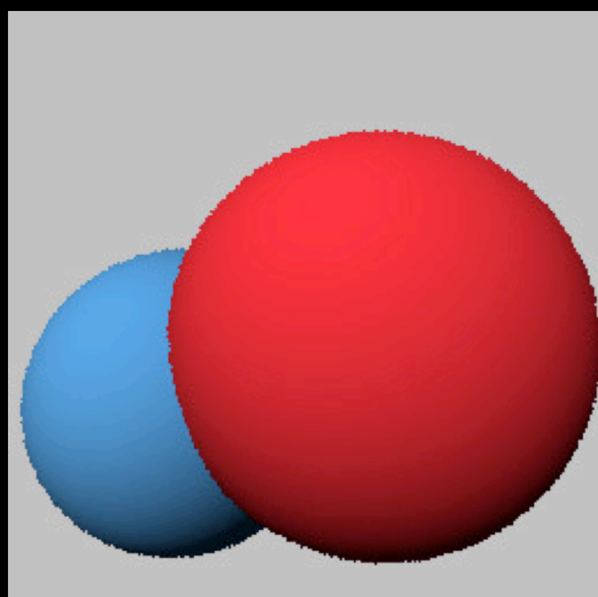
Cache hit ratio

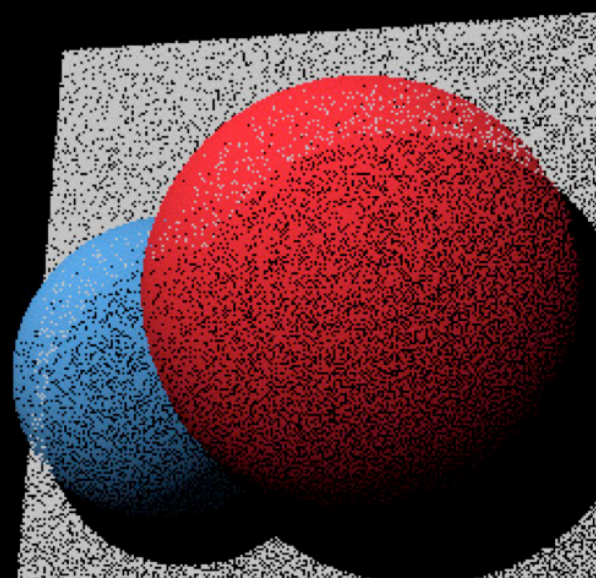
Cache hit ratio results



Second insight

- Use reverse mapping
  - Recompute scene geometry at each frame
  - Leverage hardware filtering for lookup



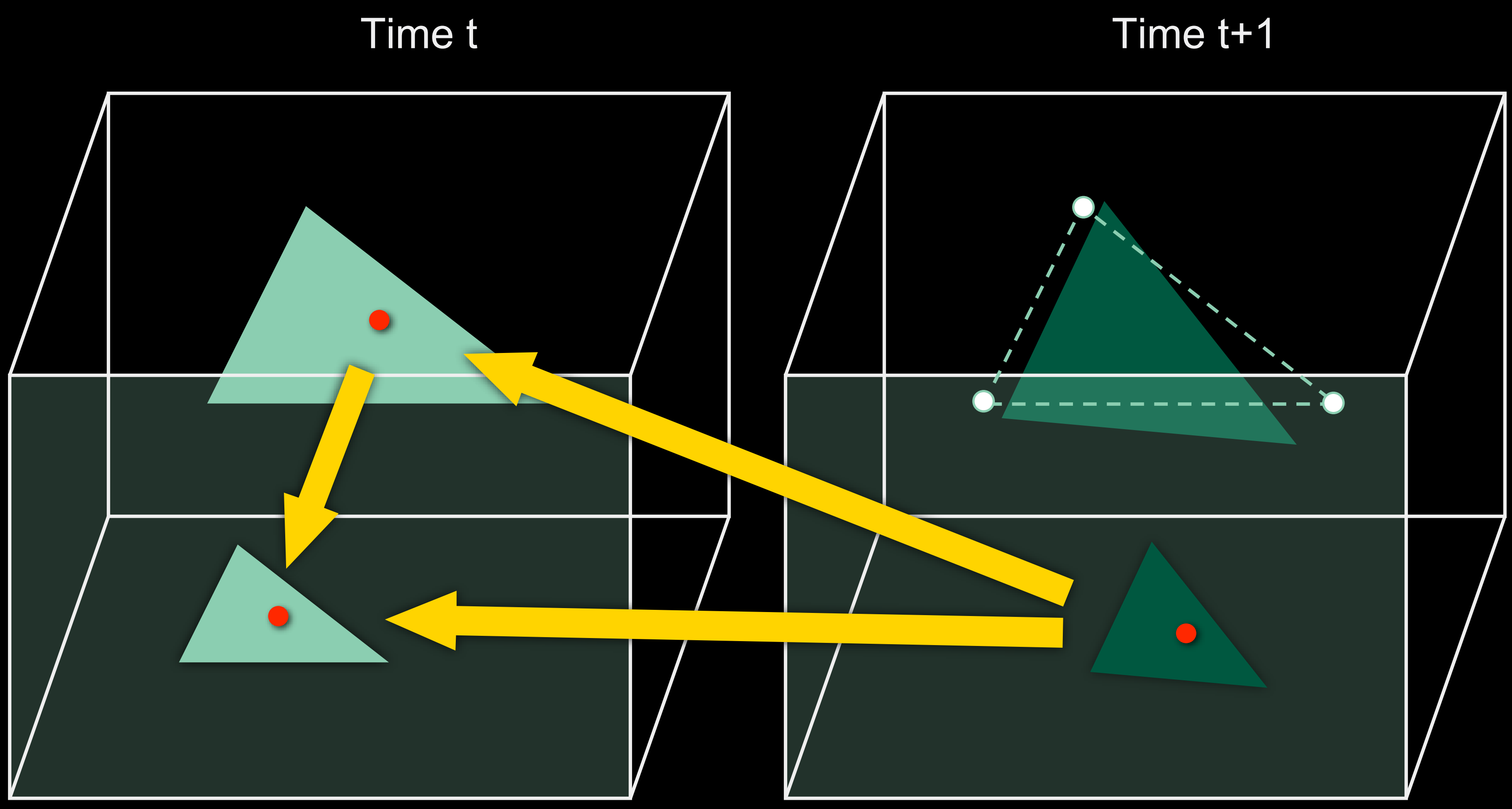


[Walter et al. 1999]

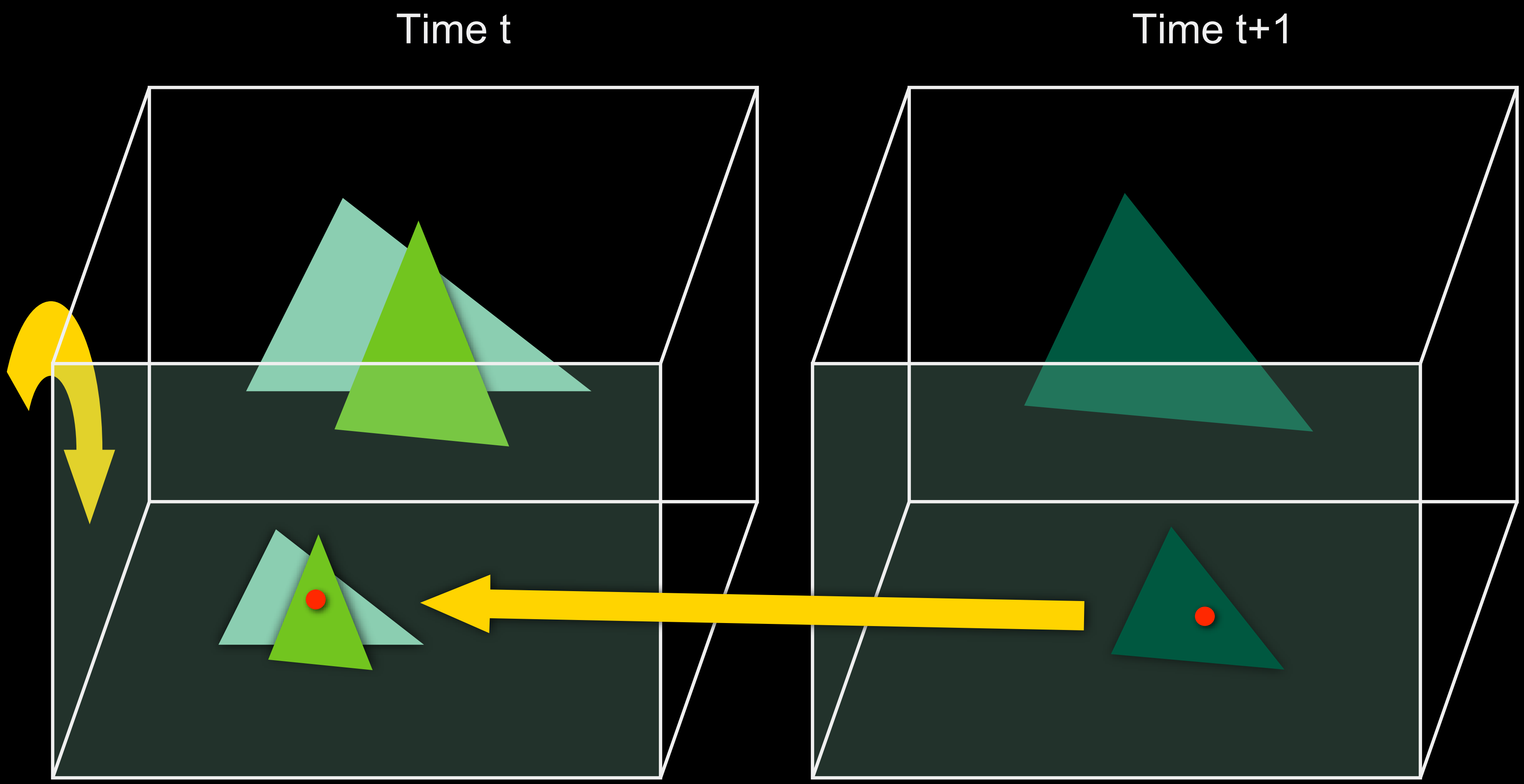
Third insight

- Do not need to reproject at the pixel level
- Hard work is performed at the vertex level

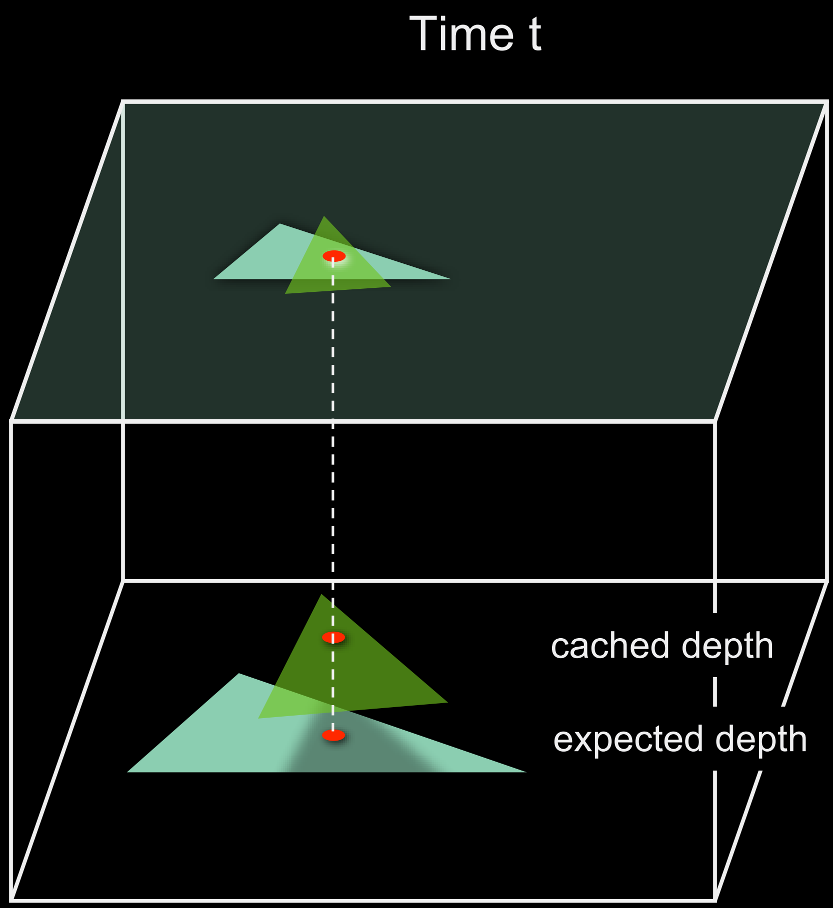
Address calculation



Hit or miss?



Hit or miss?



- Load cached depth
- Compare with expected depth

Third insight

- Do not need to reproject at the pixel level
- Hard work is performed at the vertex level
  - Pass old vertex coords as texture coords
  - Leverage perspective-correct interpolation
  - One single final division within pixel shader

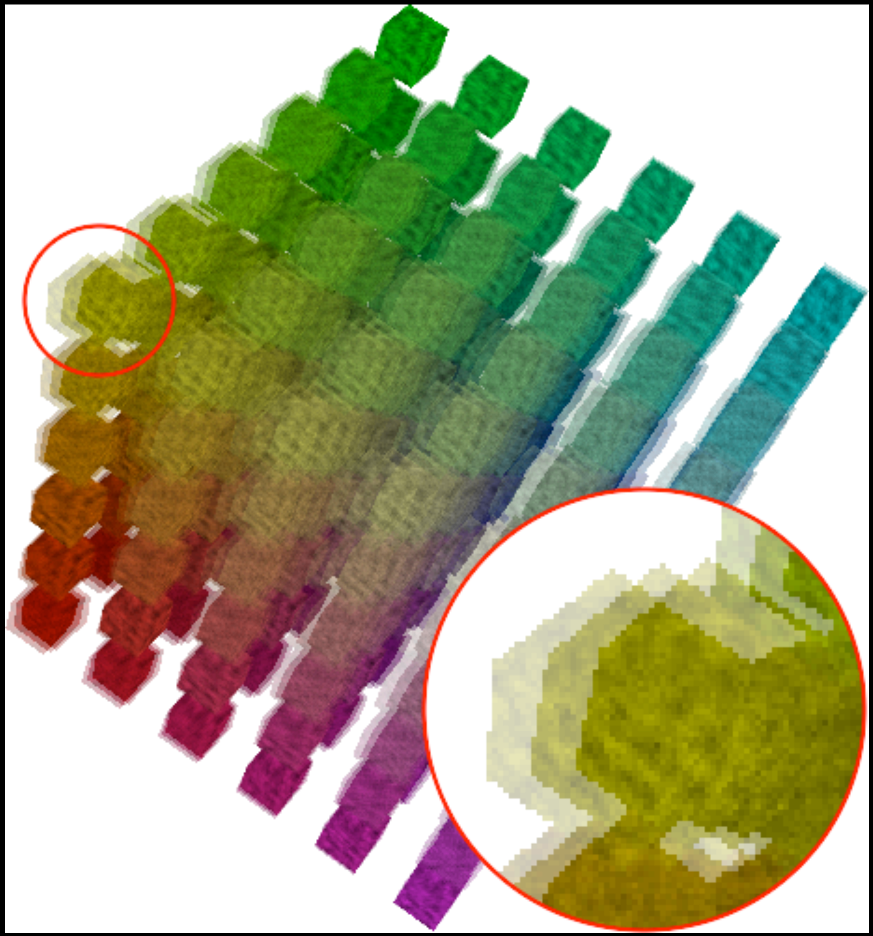
What to cache?

- Slow varying, expensive computations
  - procedural albedo
- Values required in multiple passes
  - color in depth of field or motion blur
- Samples within a sampling process
  - amortized shadow map tests

Refreshing the cache

- Cached entries become stale with time
  - View dependent effects, repeated resampling
- Implicit (multipass algorithms)
  - Flush entire cache each time step
- Random updates
  - Refresh random fraction of pixels
- Amortized update
  - Combine cache with new values at each frame

Motion blur



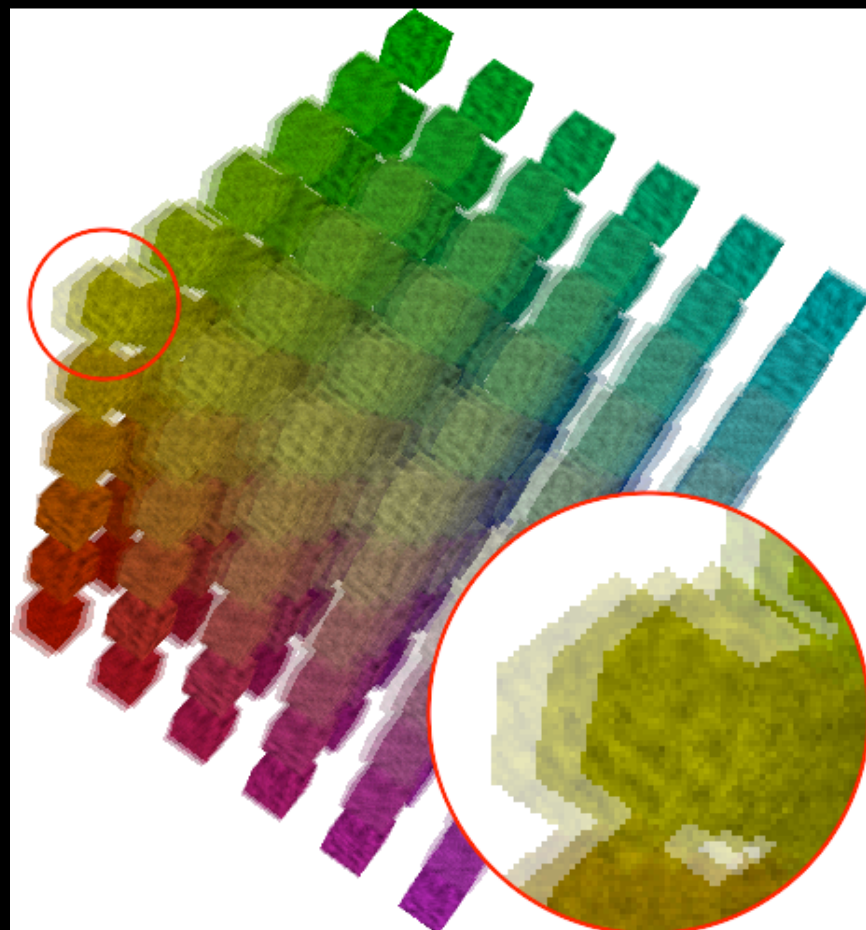
3 passes

60fps brute force

Reuse albedo in multipass

- For each time step
  - Fully compute albedo in first pass
  - For each remaining pass
    - Lookup into first pass and try to reuse

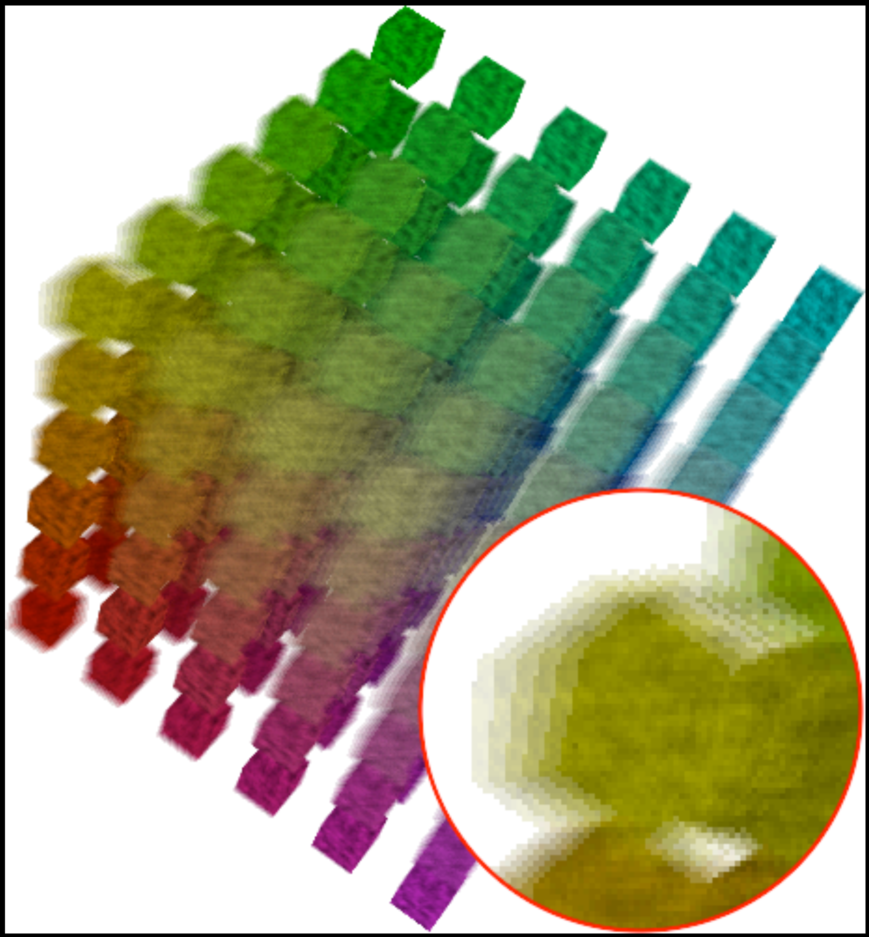
Motion blur



3 passes

60fps brute force

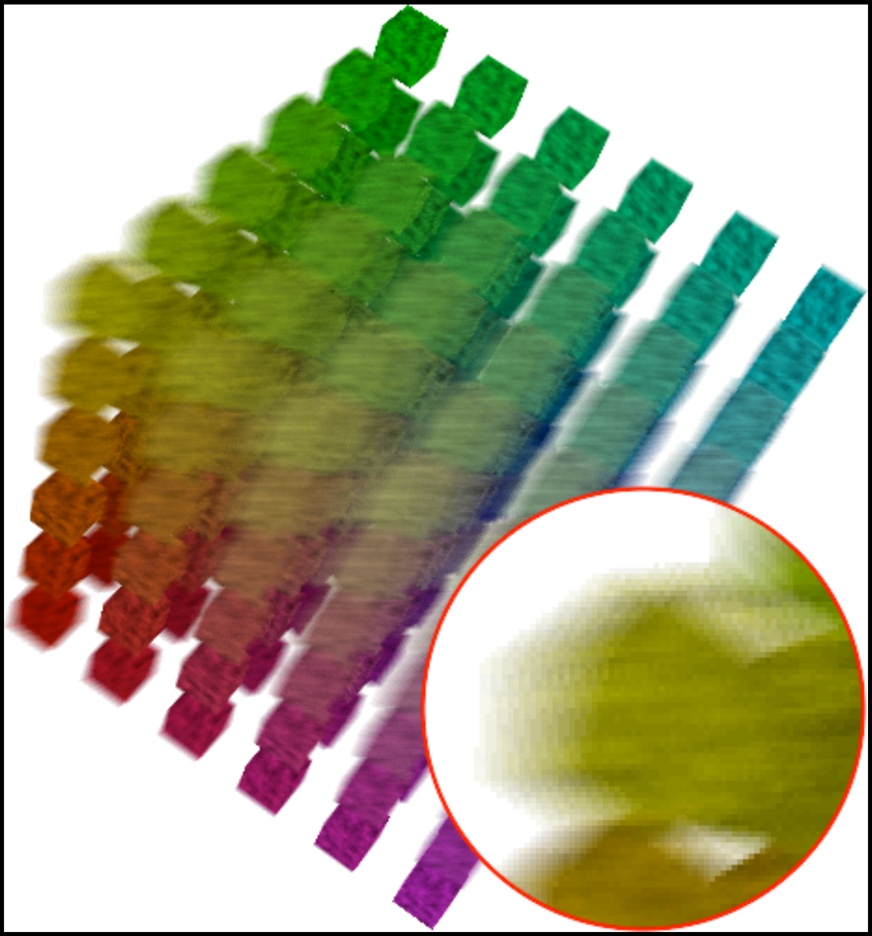
Motion blur



6 passes

60fps cached
30fps brute force

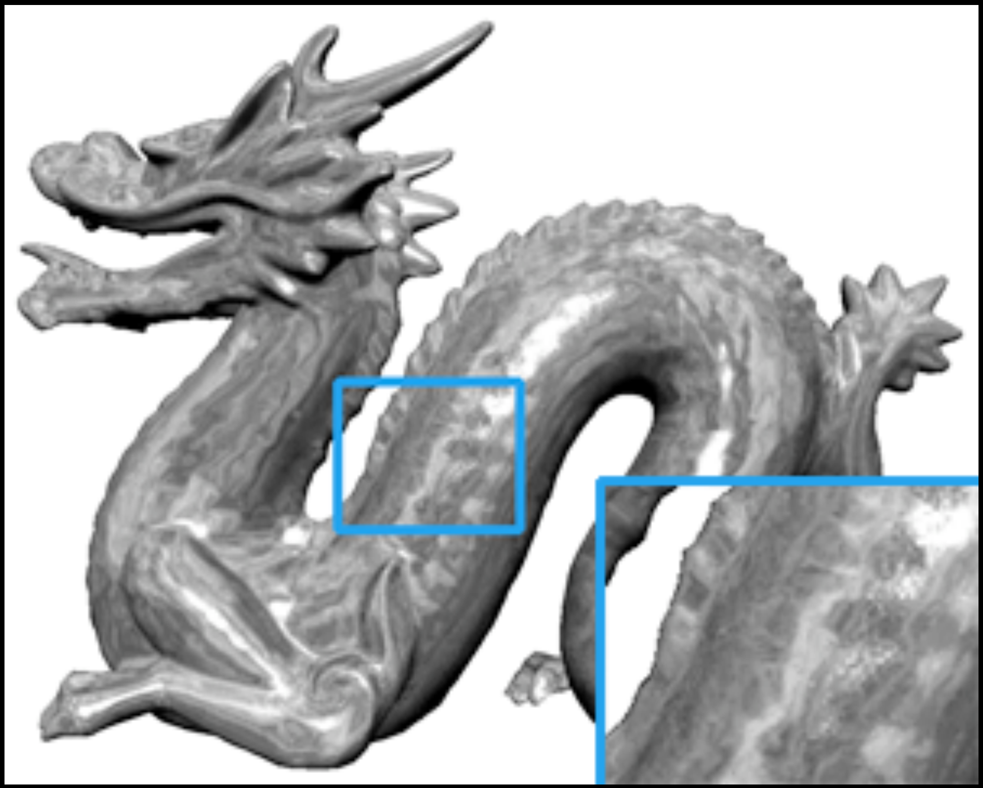
Motion blur

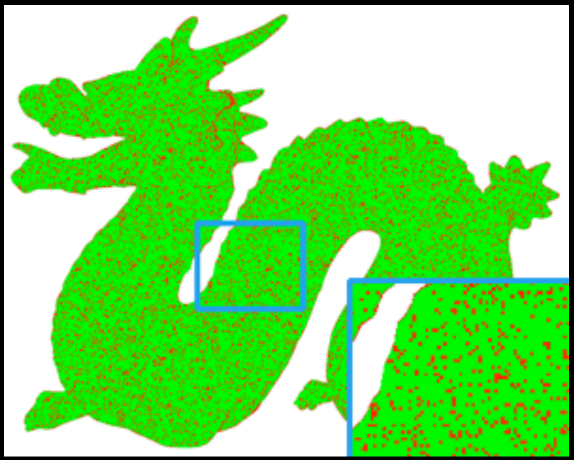


14 passes

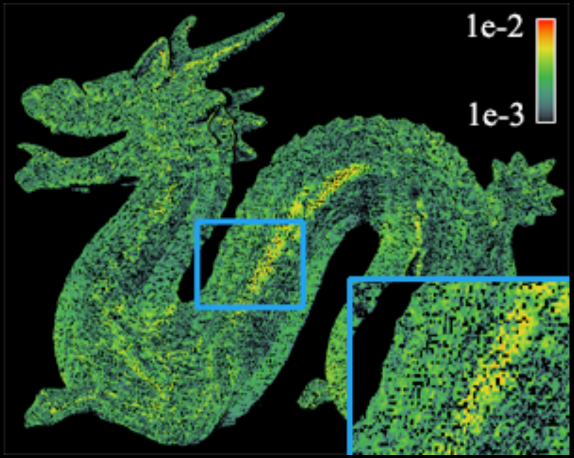
30fps cached

Randomly distributed refresh





1/4th updated



Error plot

Amortized super-sampling

- Cache updated by recursive filter rule

$$C_{t+1} = \lambda C_t + (1-\lambda)s_{t+1}$$

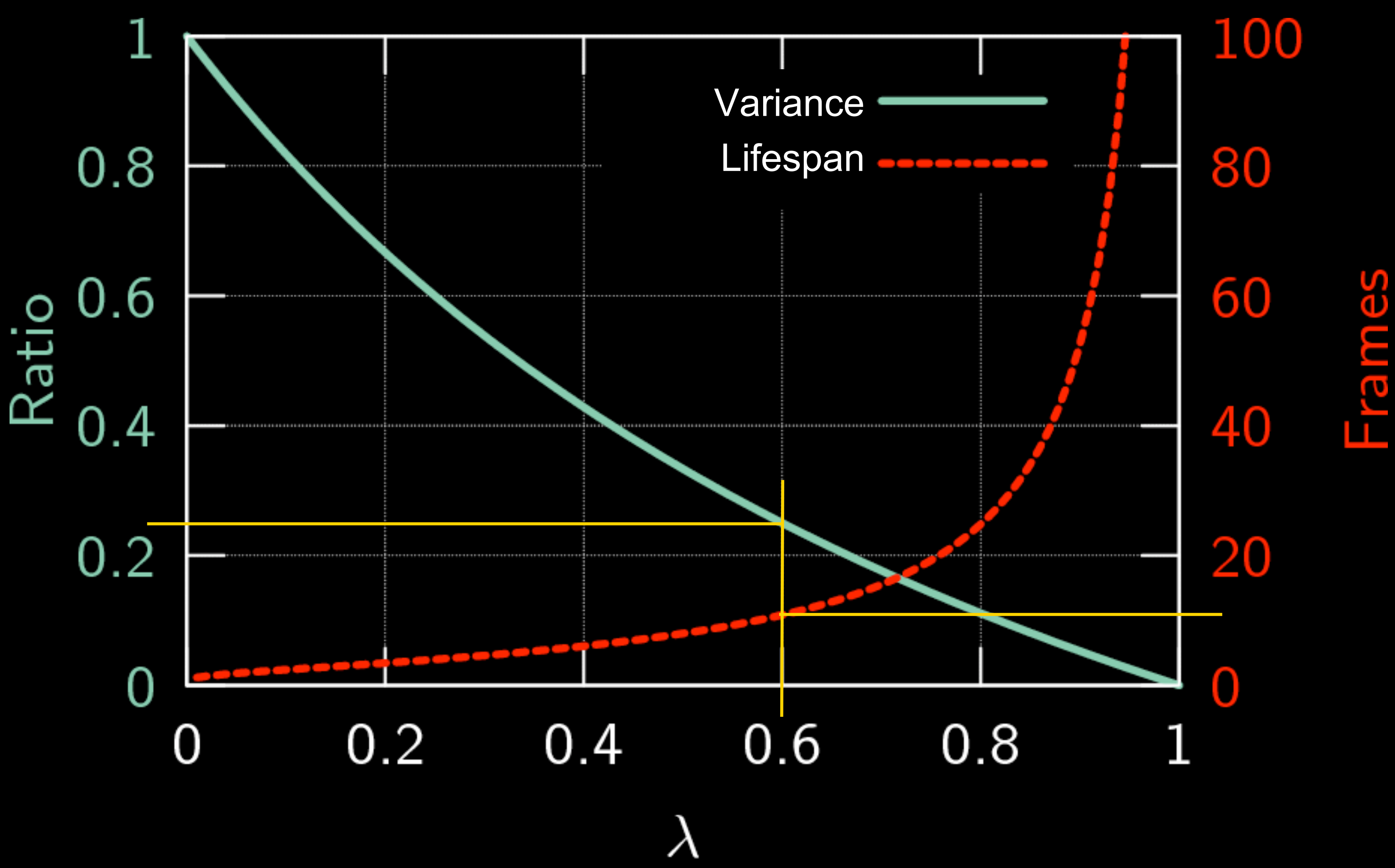
- Lambda controls variance reduction...

$$\frac{\mathrm{var}(C)}{\mathrm{var}(s)} = \frac{1-\lambda}{1+\lambda} < 1$$

- ...but also the lifespan

$$\tau(\lambda) = -\frac{1}{\ln \lambda}$$

Trade-offs


Variance
Lifespan
Ratio
Frames
1
0.8
0.6
0.4
0.2
0
100
80
60
40
20
0
0
0.2
0.4
0.6
0.8
1
λ

Variance reduction at work

16 tap PCF

4 tap PCF

Reusing shadow map tests

- At each frame, perform new shadow tests
- Read running sum from cache
- Blend the two values
- Update cache and display results

Variance reduction at work

16 tap PCF

4 tap PCF

Variance reduction at work

16 tap PCF

4 tap amortized

Conclusions

- Shading every frame anew is wasteful
- We can reuse some of the shading computations from previous frames
- Use reverse reprojection caching to do that in real-time rendering applications
- Less work per frame = faster rendering

Future work

- Track surface points and select shader level of detail based on screenspace speed
- Change refresh rate per pixel based on rate of cached value change
- Use code analysis to automatically select appropriate values to cache