
SIGGRAPH2015
Xroads of Discovery

SIGGRAPH2015
Xroads of Discovery

The 42nd International Conference and Exhibition
on Computer Graphics and Interactive Techniques

ARM®

Bandwidth-Efficient Rendering

Marius Bjørge
ARM

Agenda

- Efficient on-chip rendering
- Post-processing
  - Bloom
  - Blur filters

Efficient on-chip rendering

- Extensions
  - Framebuffer fetch
  - Pixel Local Storage

- Why extensions?
  - Surely mobile GPUs are already bandwidth-efficient?

Framebuffer fetch

- Read the current fragment’s previous color value
- ARM also supports reading the previous depth and stencil values of the current fragment

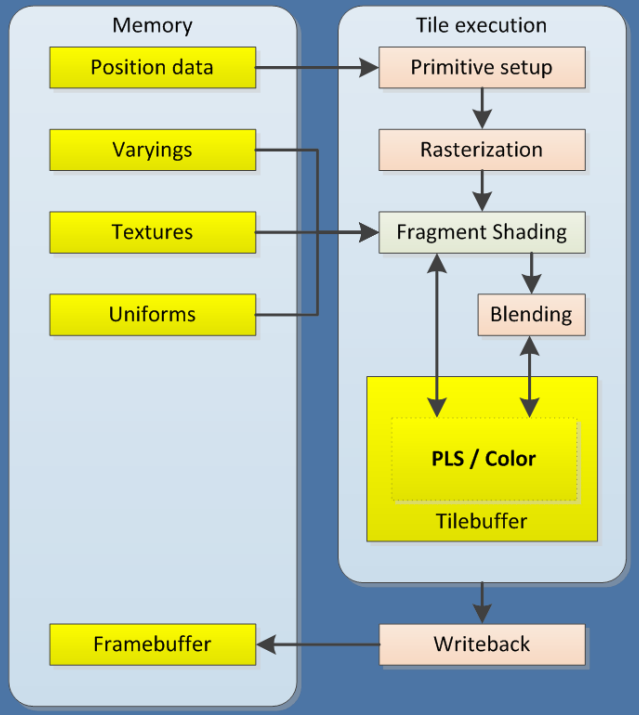
- Useful for
  - Programmable blending
  - Programmable depth/stencil testing

Pixel Local Storage (PLS)

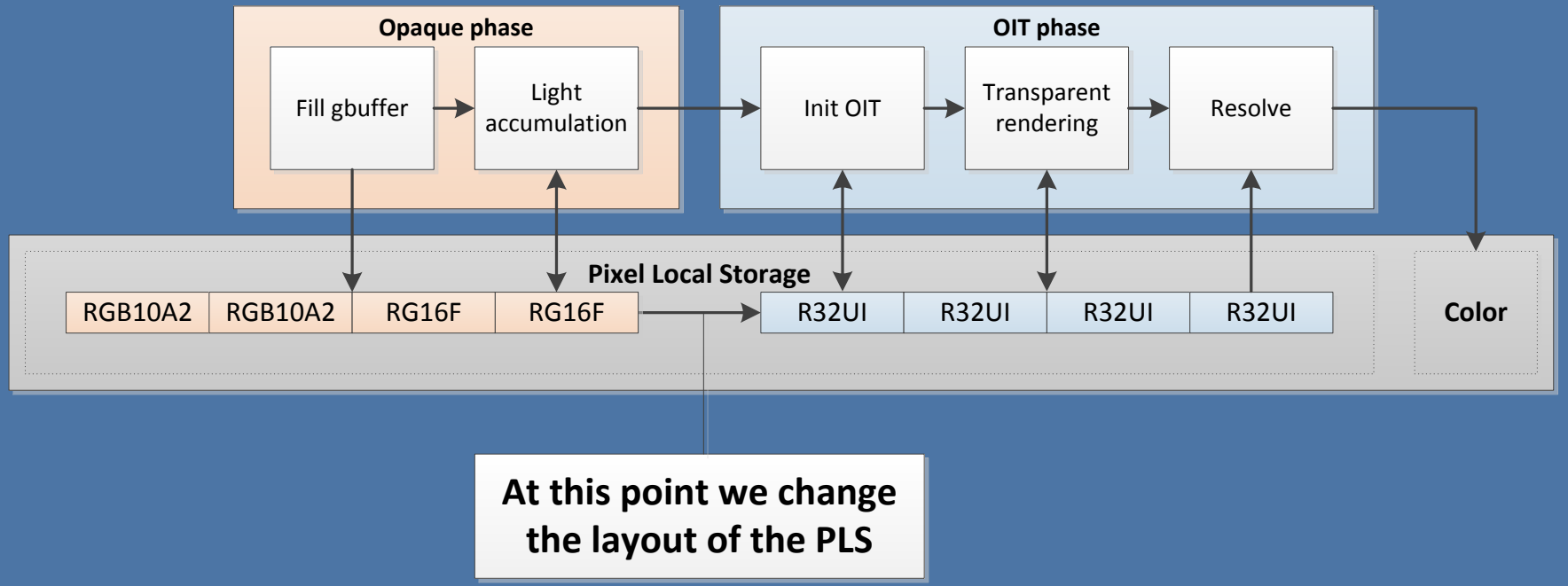
- Per-pixel storage that is persistent throughout the lifetime of the frame
  - Read/write access
  - Storage stays on-chip
  - Storage layout declared per fragment shader invocation – does not depend on framebuffer format
- Useful for
  - Deferred shading
  - Order Independent Transparency [1]
  - Volume rendering

Pixel Local Storage (PLS)

- Rendering pipeline changes slightly when PLS is enabled
  - Writing to PLS bypasses blending
- Note
  - Fragment order
  - PLS and color share the same memory location



Pixel Local Storage (PLS)




Post-processing

Post-processing

- High-end mobile devices typically have small displays with massive resolutions
- Rendering at native resolution is often out of the question, especially if you add post-processing to the mix

- Solution: mixed resolution rendering
  - Go as low as you can without sacrificing quality, and then upscale

Mobile post-processing

On-chip

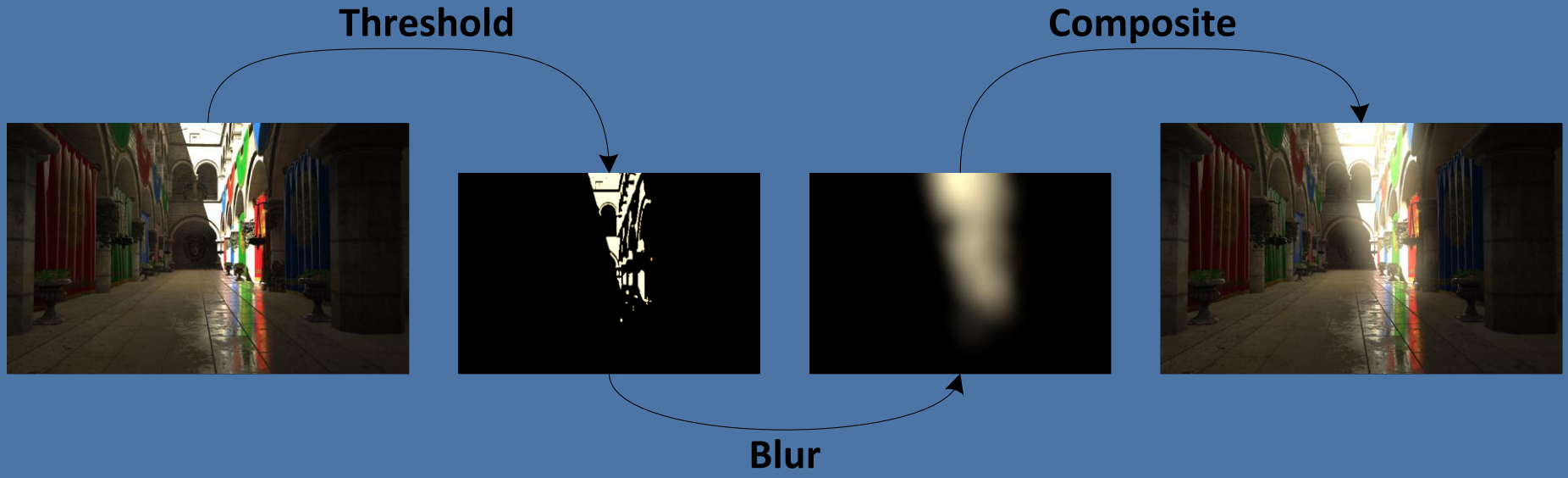
- Color Grading
- Tonemapping

Off-chip

- Anti-aliasing
- Bloom
- Depth of Field
- Screen Space Ambient Occlusion
- Screen Space Reflections

Bloom

- Doesn’t have to be physically correct
- Wide + thin

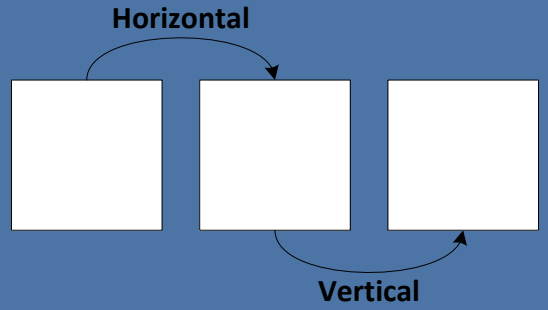


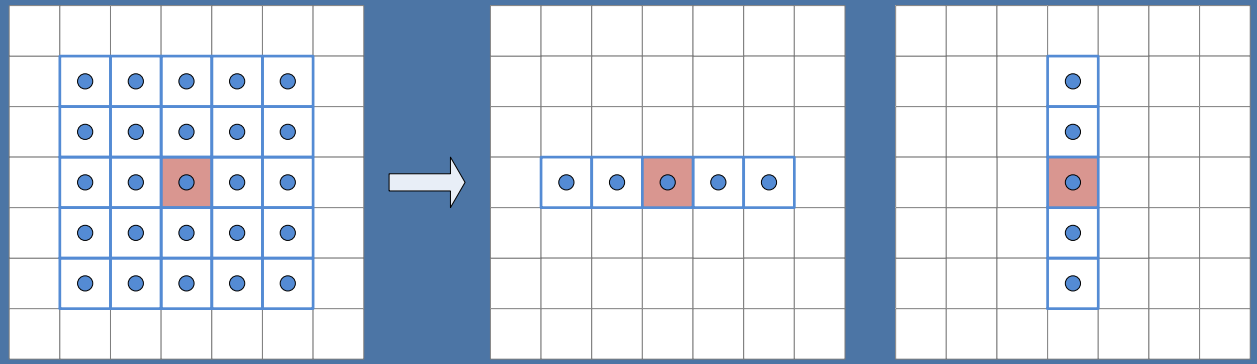
Blur

- What makes a good blur filter?
- Goal:
  - High quality
  - Stable
  - High performance

Box blur

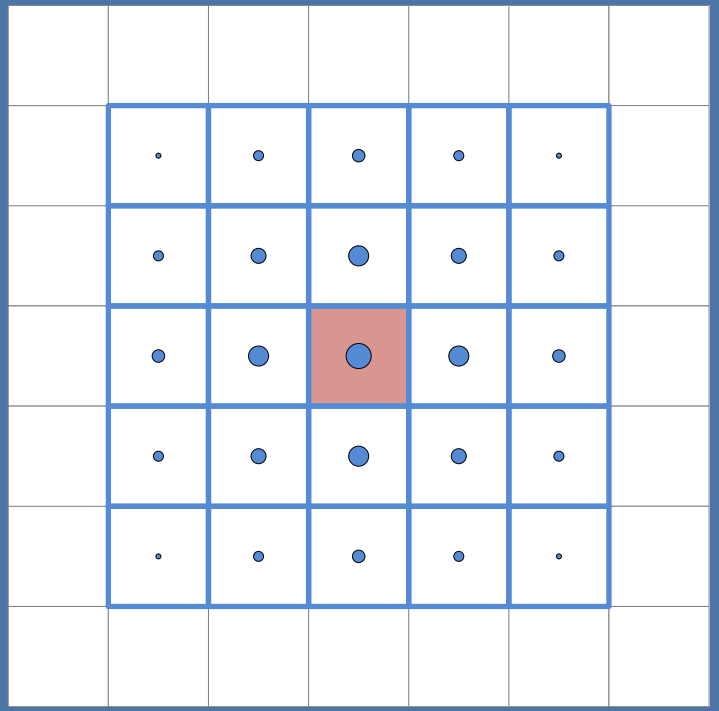
- 5x5 box blur = 25 samples
- Separate the blurs
  - 5 + 5 = 10 samples





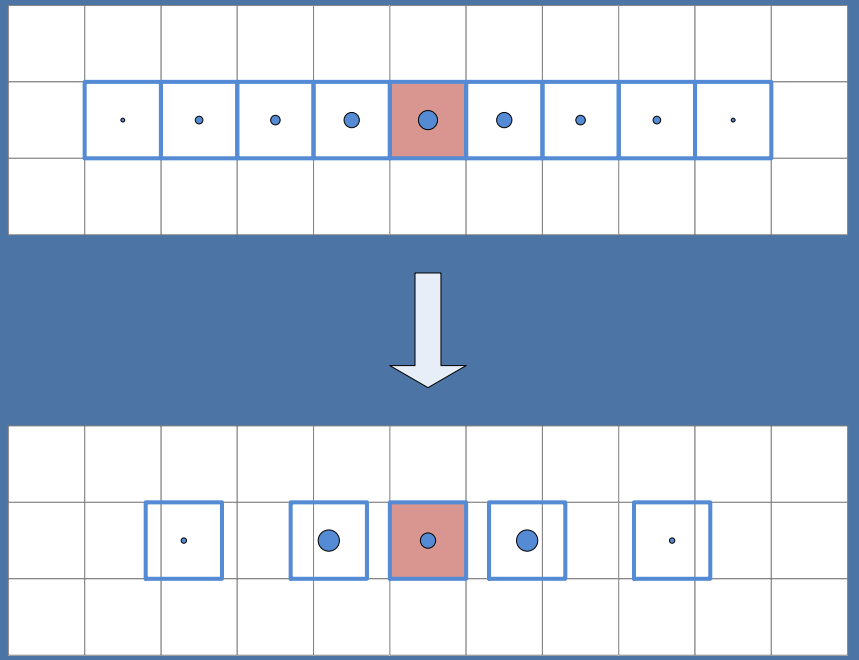
Gaussian blur

- Convolve a gaussian function over the image
- Separable just like the box filter

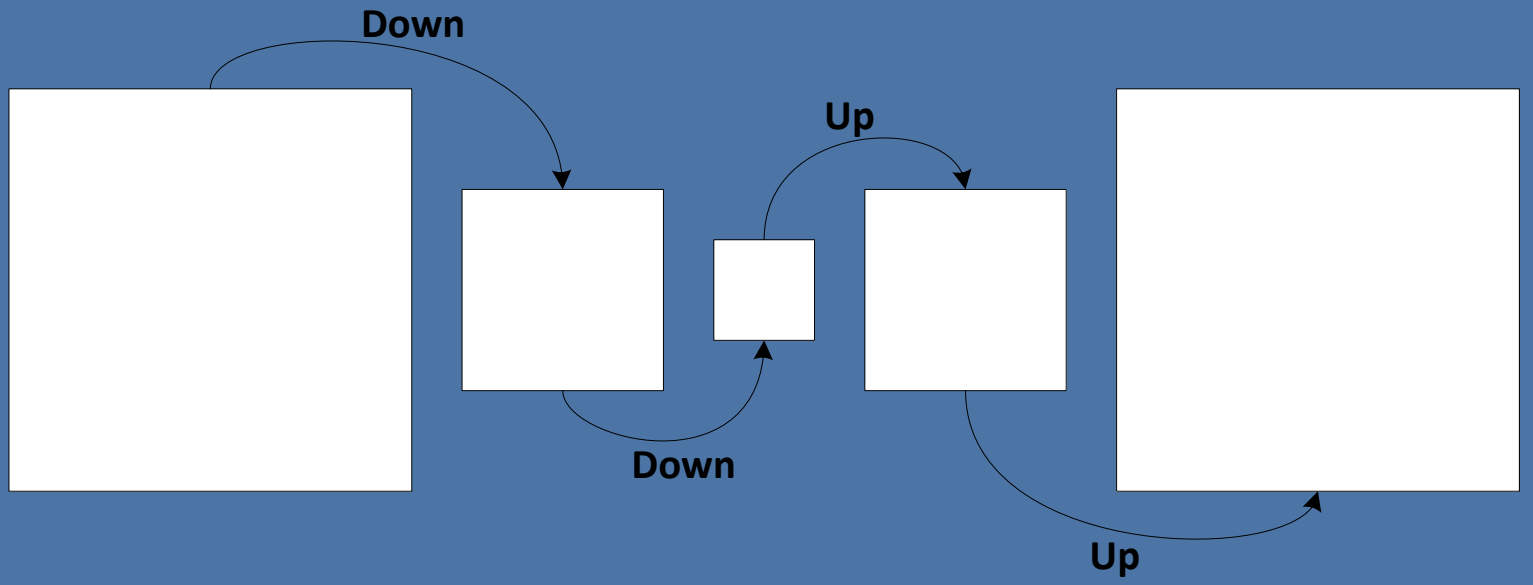


Linear sampling optimization [2]

- Reduce number of texture lookups by exploiting the HW texture unit
  - Modify sample offsets and gaussian weights
- Get 9x9 at similar cost as 5x5

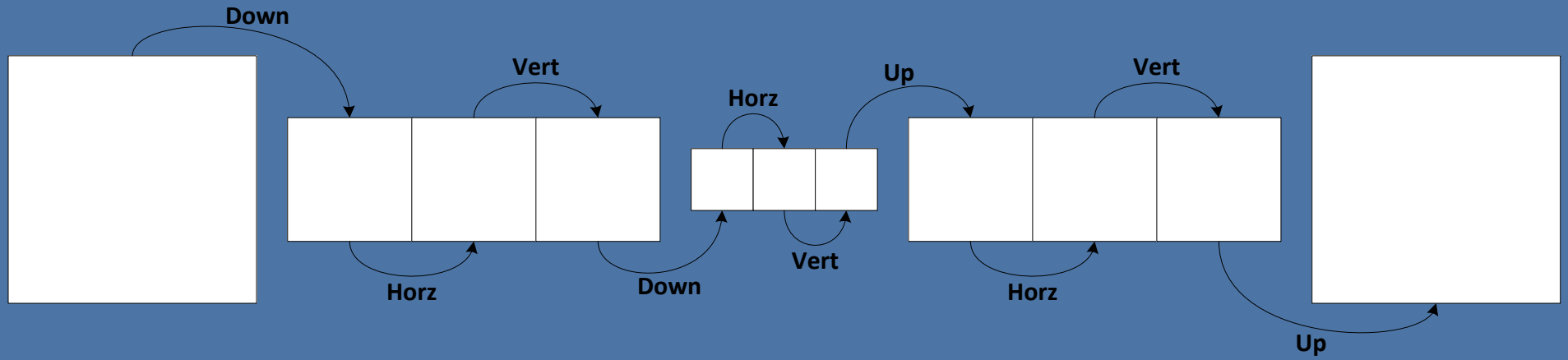


Mixing resolutions

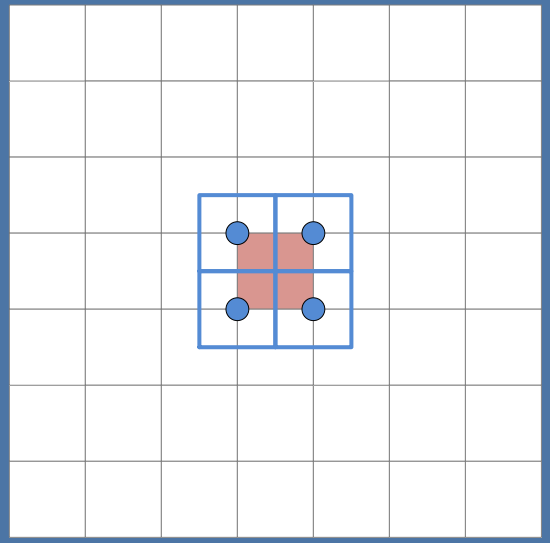


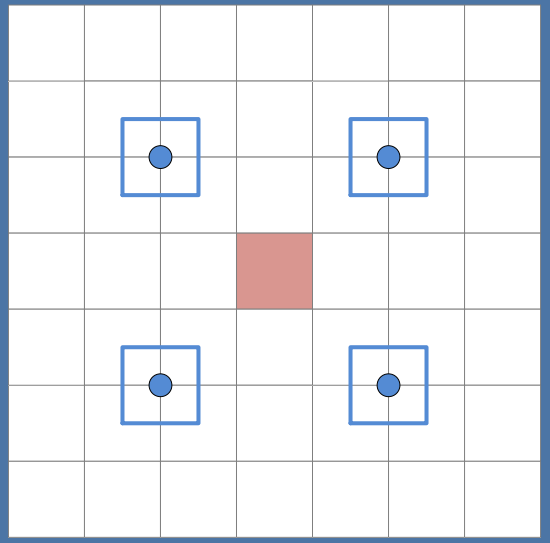
Mixing resolutions

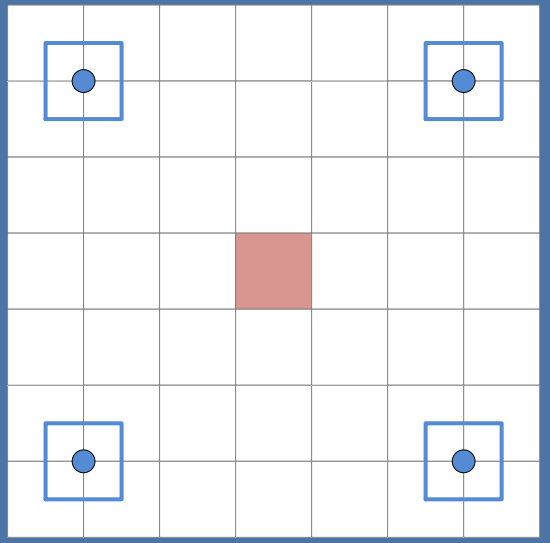
- Gets increasingly complicated when using separable kernels



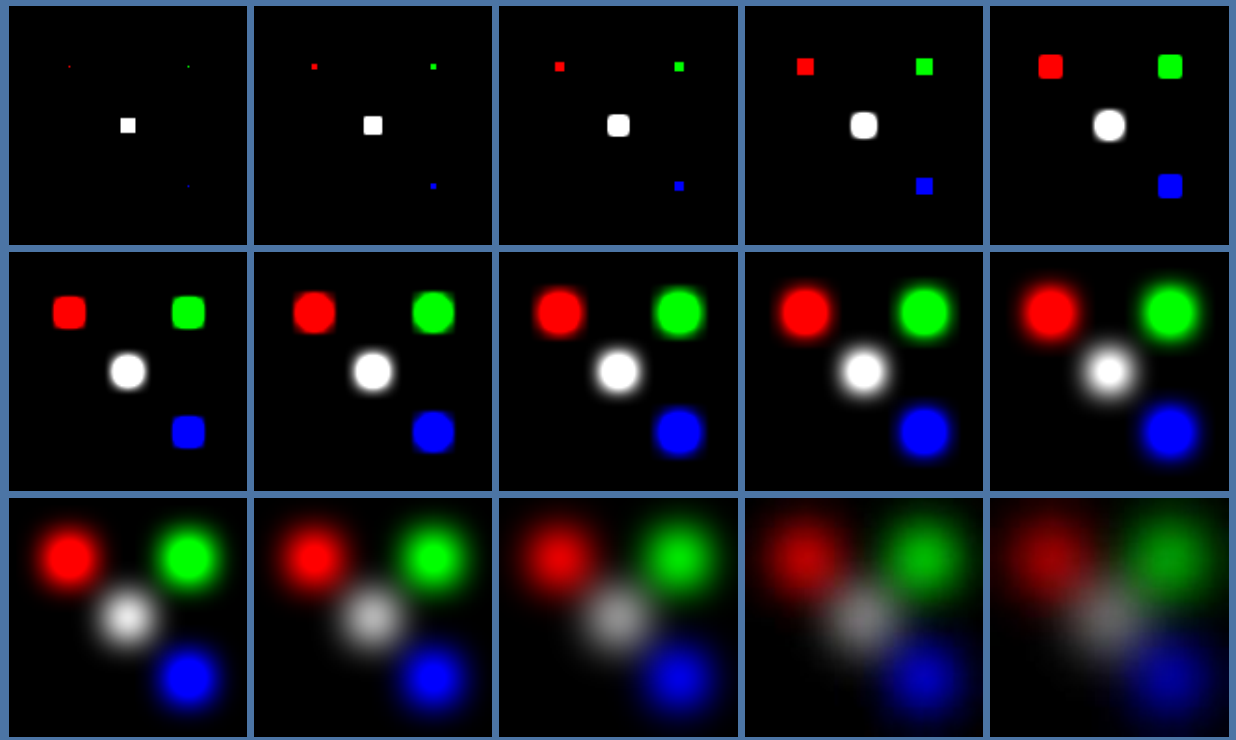
Kawase blur [3]





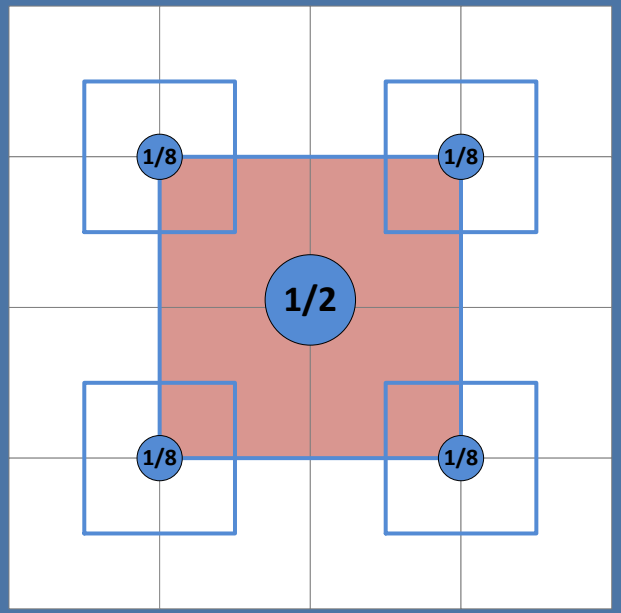


Kawase blur

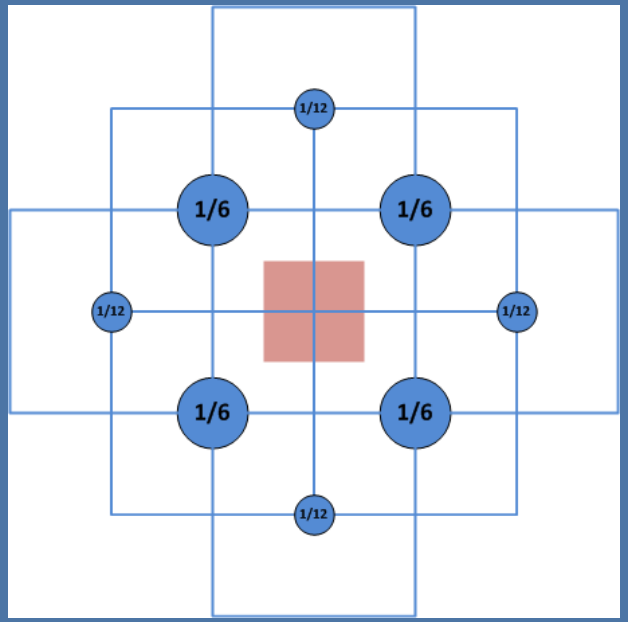


“Dual filtering”

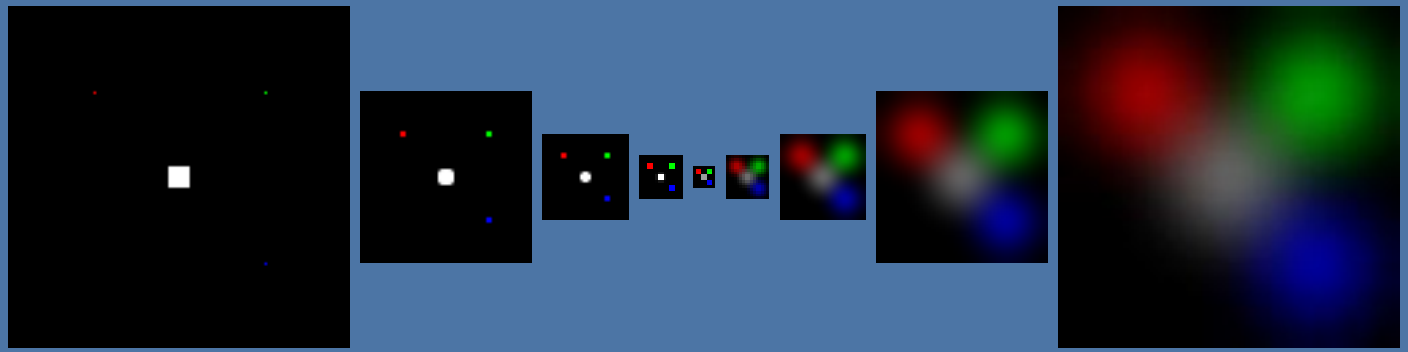
Downsample filter



Upsample filter



“Dual filtering”



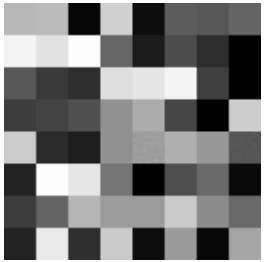
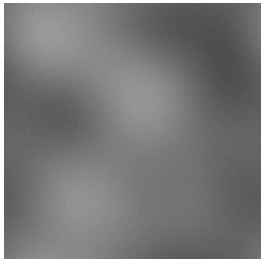
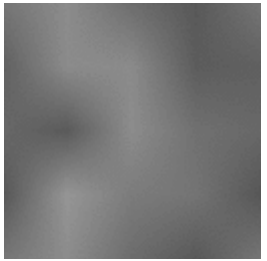
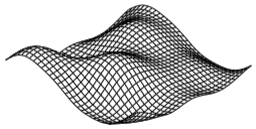
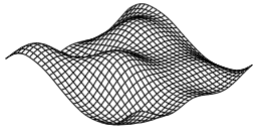
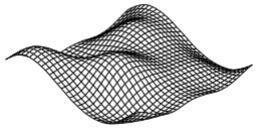
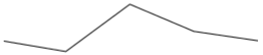
Comparing filters

Comparison setup

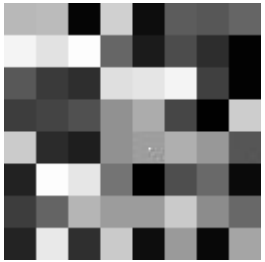
- 97x97 blur
- Gaussian used as reference
- Kawase
  - First downsample to 1/16th resolution
  - Setup with 0, 1, 2, 3, 4, 4, 5, 6, 7 distances passes
- “Dual filtering” setup with 8 passes
- Naïve method which relies on glGenerateMipmap

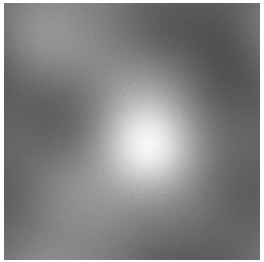
Input	Reference	Dual	Kawase
			
			
	PSNR:	**49.78 dB**	**50.02 dB**

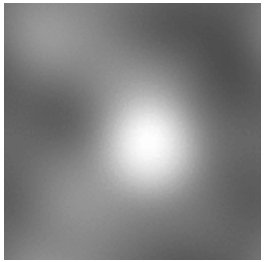
Stability comparison

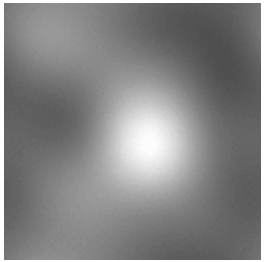
Input	Reference	Dual	Kawase	Naive
				
				
				

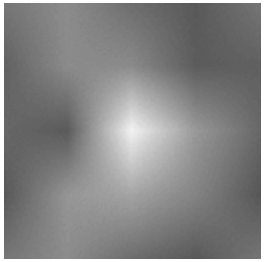
Input	Reference	Dual	Kawase	Naive

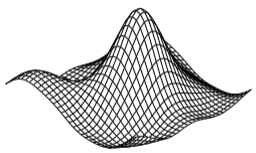


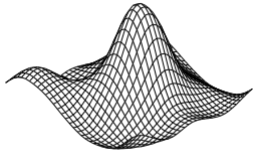


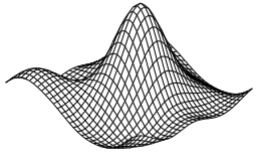


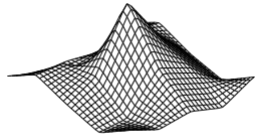


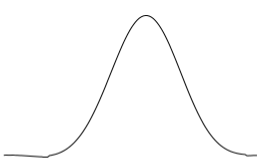


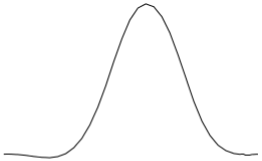


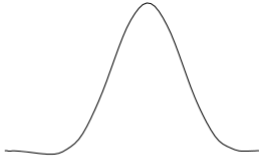


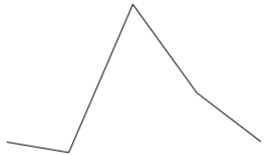


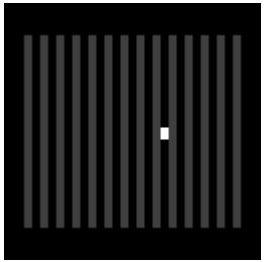
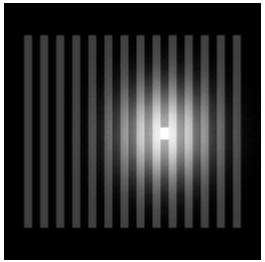
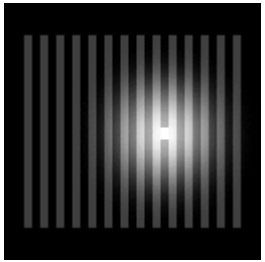
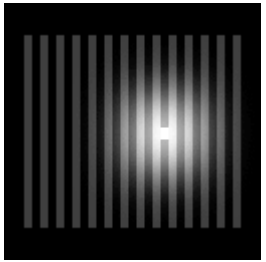
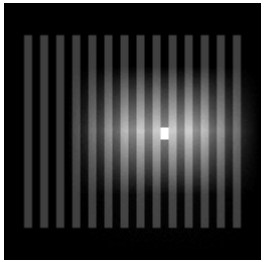






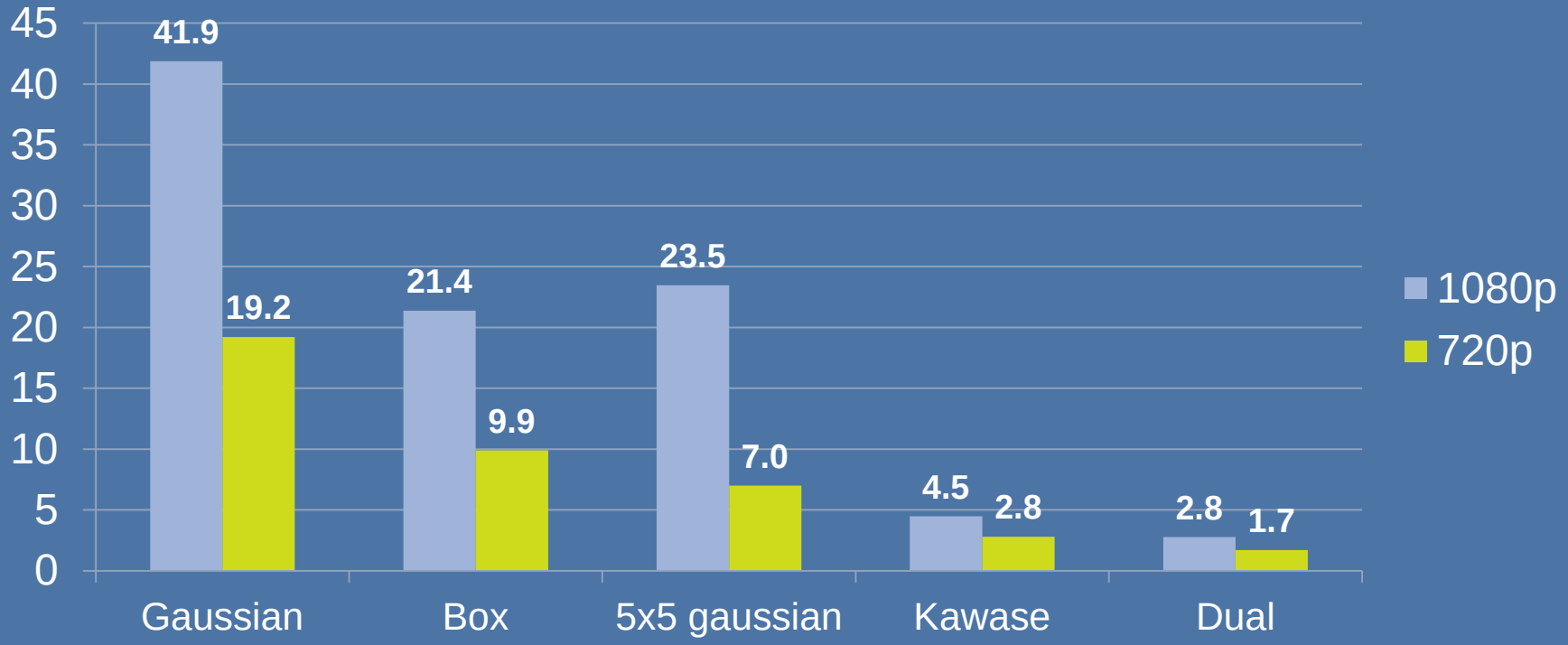




Input	Reference	Dual	Kawase	Naive
				

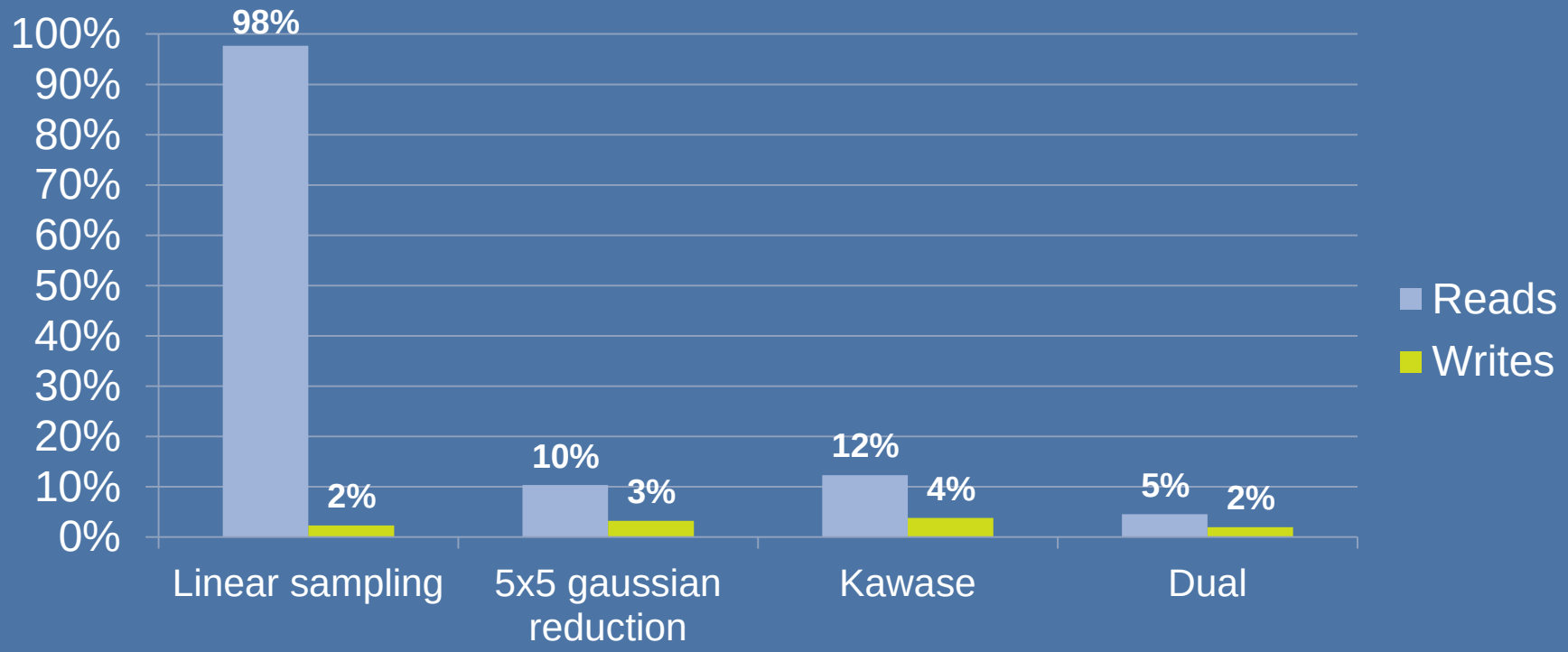
Performance comparison

Performance (ms)

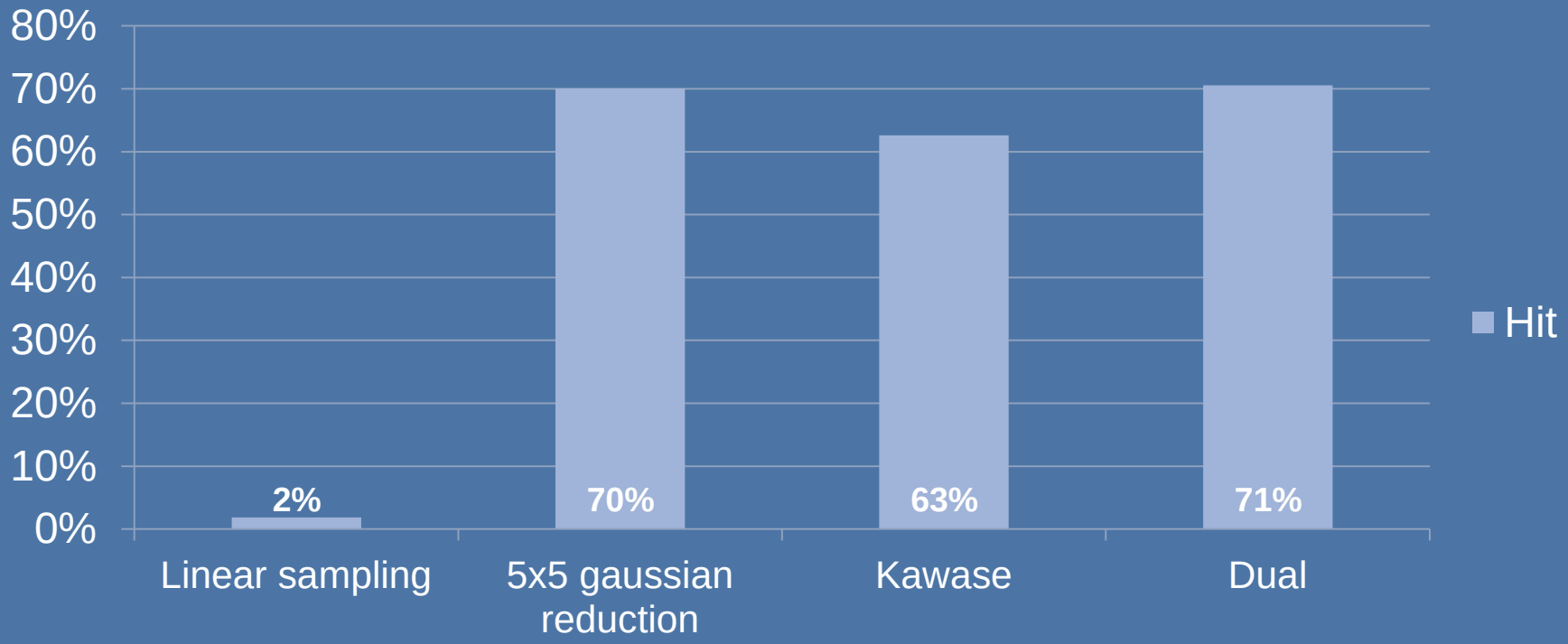


Tested on a Mali-T760 MP8

Bandwidth


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
98%
2%
10%
3%
12%
4%
5%
2%
Linear sampling
5x5 gaussian reduction
Kawase
Dual
Reads
Writes

Cache utilization



Summary

- On-chip rendering
  - Please use the extensions
- Bloom
  - Multi-pass mixed resolution
  - “Dual filter” blur
- Next steps
  - Work on getting on-chip rendering into future core APIs
  - Look into alternative data flows for doing blurs

Thanks!

- Questions?
  - Marius.Bjorge@arm.com

- References

1. Efficient Rendering with Tile Local Storage [Siggraph 2014]
2. http://rastergrid.com/blog/2010/09/efficient-gaussian-blur-with-linear-sampling/
3. Frame Buffer Postprocessing Effects in DOUBLE-S.T.E.A.L [GDC 2003]