Efficient Compute Shader Programming

Bill Bilodeau

AMD

Topics Covered in this Talk

- Direct Compute Overview
- GPU Architecture
- Compute Shader Optimization
  - GPUPerfStudio 2.5
  - Code Example: Gaussian Blur
- Ambient Occlusion
- Depth of Field

Direct Compute

- DirectX interface for general purpose computing on the GPU
  - General purpose computing can be done in a pixel shader
  - Compute Shader advantages
    - More control over threads
    - Access to shared memory
    - No need to render any polygons

Compute Shader Uses in Games

- High quality filters
  - When 2x2 HW biliniar filtering isn't good enough
- Post Processing Effects
  - Screen space ambient occlusion
  - Depth of Field
- Physics
- AI
- Data Parallel Processing
  - Any algorithm that can be parallelized over a large data set

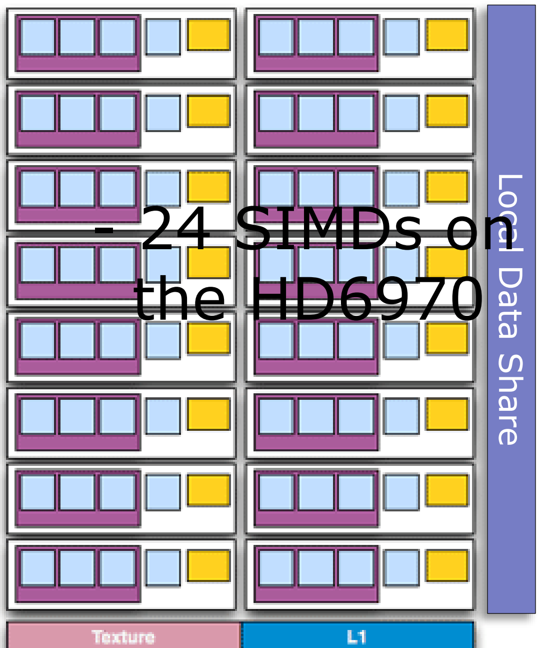
Direct Compute Features

- Thread Groups
  – Threads can be grouped for compute shader execution
- Thread Group Shared Memory
  – Fast local memory shared between threads within the thread group.

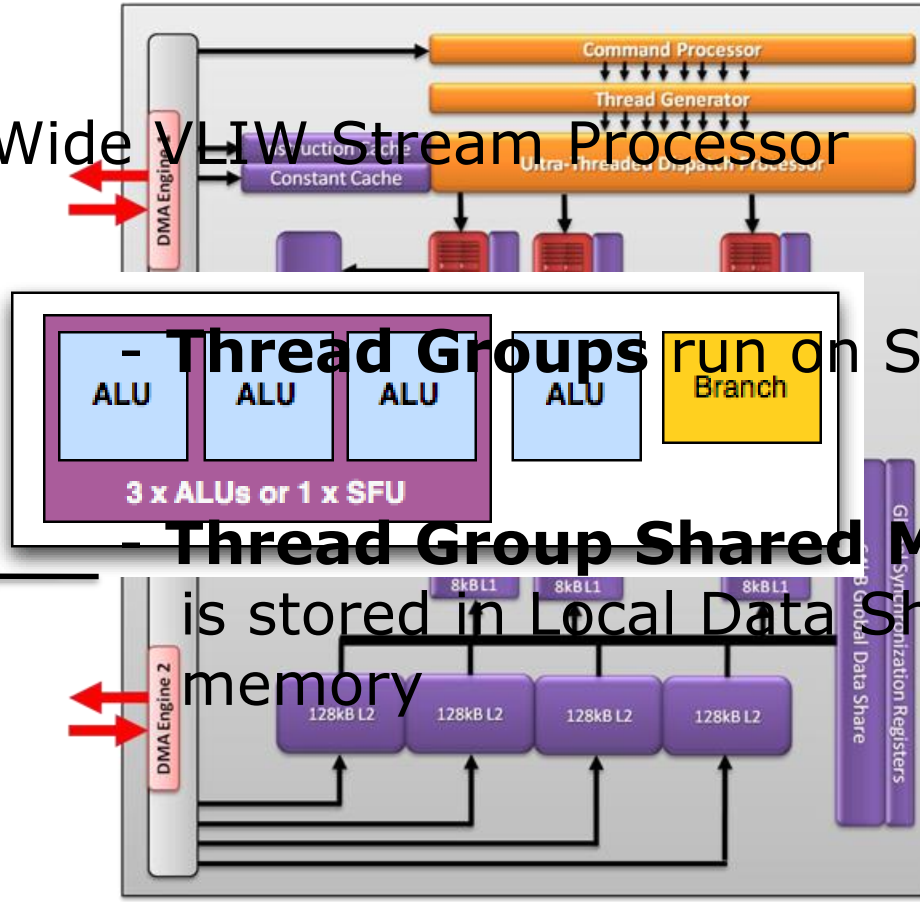
GPU Archtecture Overview: HD6970

SIMD

4 –Wide VLIW Stream Processor



- 24 SIMDs on the HD6970



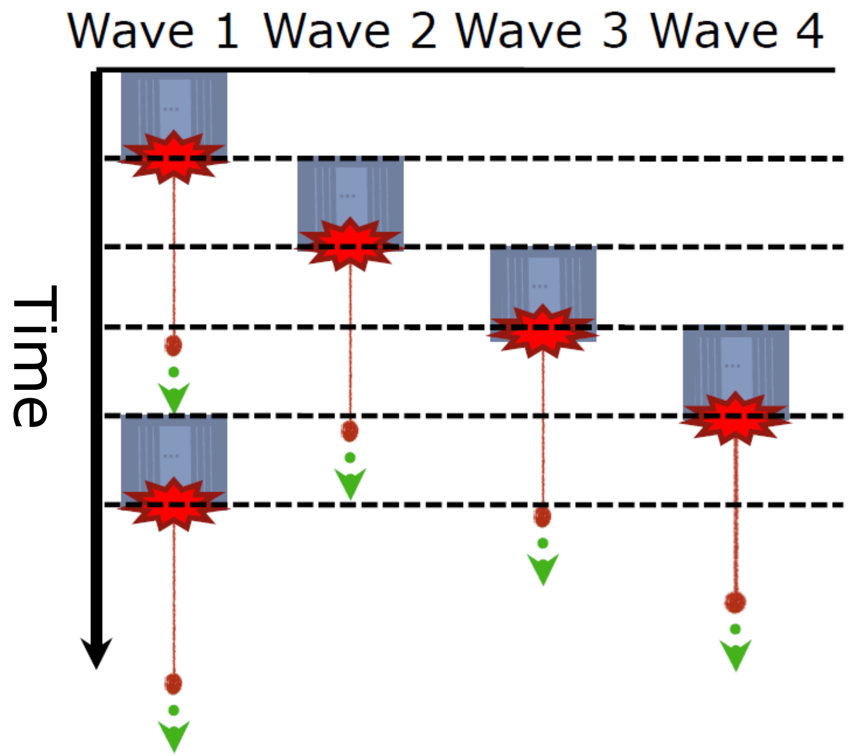
- **Thread Groups** run on SIMDs
- **Thread Group Shared Memory** is stored in Local Data Share memory

GPU Archtecture Overview: Wavefronts

- GPU Time-slices execution to hide latency

- 1 Wavefront = 4 waves of threads per SP

- 16 SPs per SIMD, so 16 x 4 = 64 threads per Wavefront



What does this mean for Direct Compute?

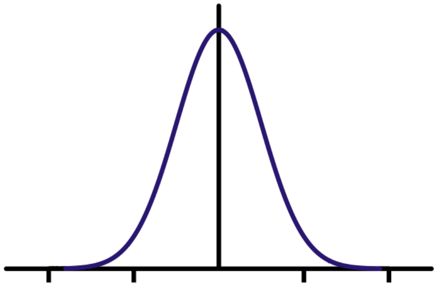
- Thread Groups
  - Threads are always executed in wavefronts on each SIMD
  - Thread group size should be a multiple of the wavefront size (64)
    - Otherwise, [(Thread Group Size) mod 64] threads go unused!
- Thread Group Shared Memory (LDS)
  - Limited to 32K per SIMD, so 32K per thead group
  - Memory is addressed in 32 banks. Addressing the same location, or loc + (n x 32) may cause bank conflicts.
- Vectorize your compute shader code
  - 4-way VLIW stream processors

Optimization Considerations

- Know what it is you're trying to optimize
  - TEX, ALU
  - GPUPerfStudio and GPU Shader Analyzer can help with this.
- Try lots of different configurations
  - Avoid hard-coding variables
  - Use GPUPerfStudio to edit in-place
- Avoid divergent dynamic flow control
  - Wastes shader processor cycles
- Know the hardware

Example 1: Gaussian Blur

- Low-pass filter
  - Approximation of an ideal sync
  - Impulse Response in 2D:



$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- For images, implemented as a 2D discrete convolution

$$f(m,n) = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j]$$

Optimization 1: Separable Gaussian Filter

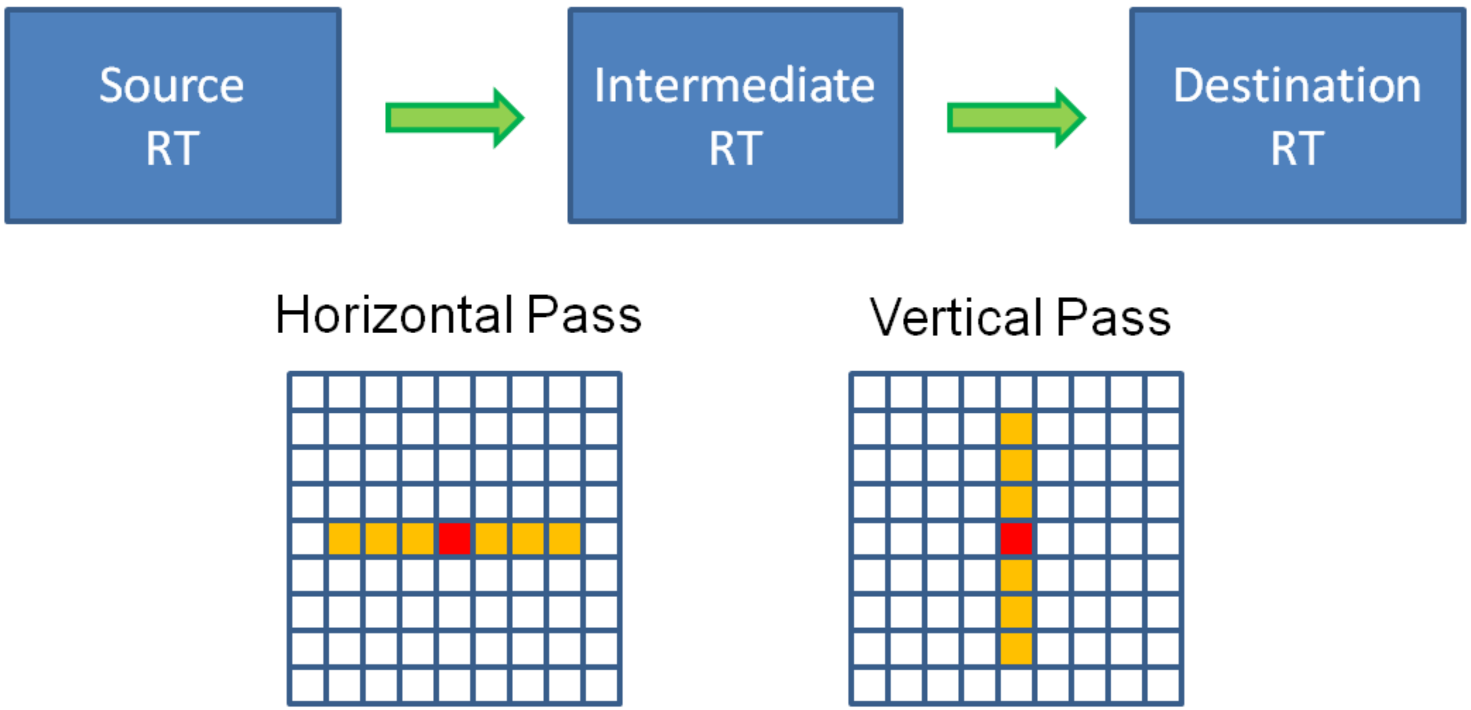
- Some 2D filters can be separated in to independent horizontal and vertical convolutions, i.e. “separable”
  - Can use separable passes even for non-separable filters
- Reduces to 1D filter with 1D convolutions:

$$h(x,y) = \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma} \cdot e^{-\frac{x^2}{2\sigma^2}}$$

$$f(n) = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

- Fewer TEX and ALU operations

Typical Pipeline Steps



Use Bilinear HW filtering?

Bilinear filter HW can halve the number of ALU and TEX instructions

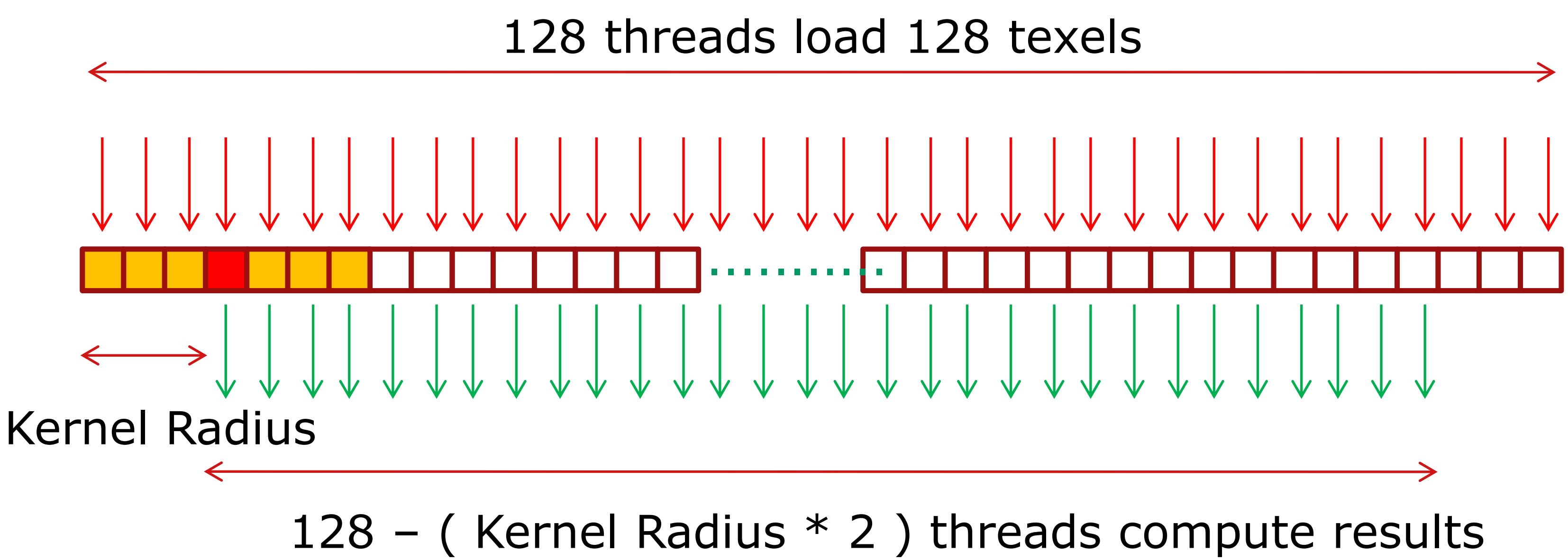
- Just need to compute the correct sampling offsets

Not possible with more advanced filters

- Usually because weighting is a dynamic operation
- Think about bilateral cases...

Optimization 2: Thread Group Shared Memory

- Use the TGSM as a cache to reduce TEX and ALU ops
- Make sure thread group size is a multiple of 64

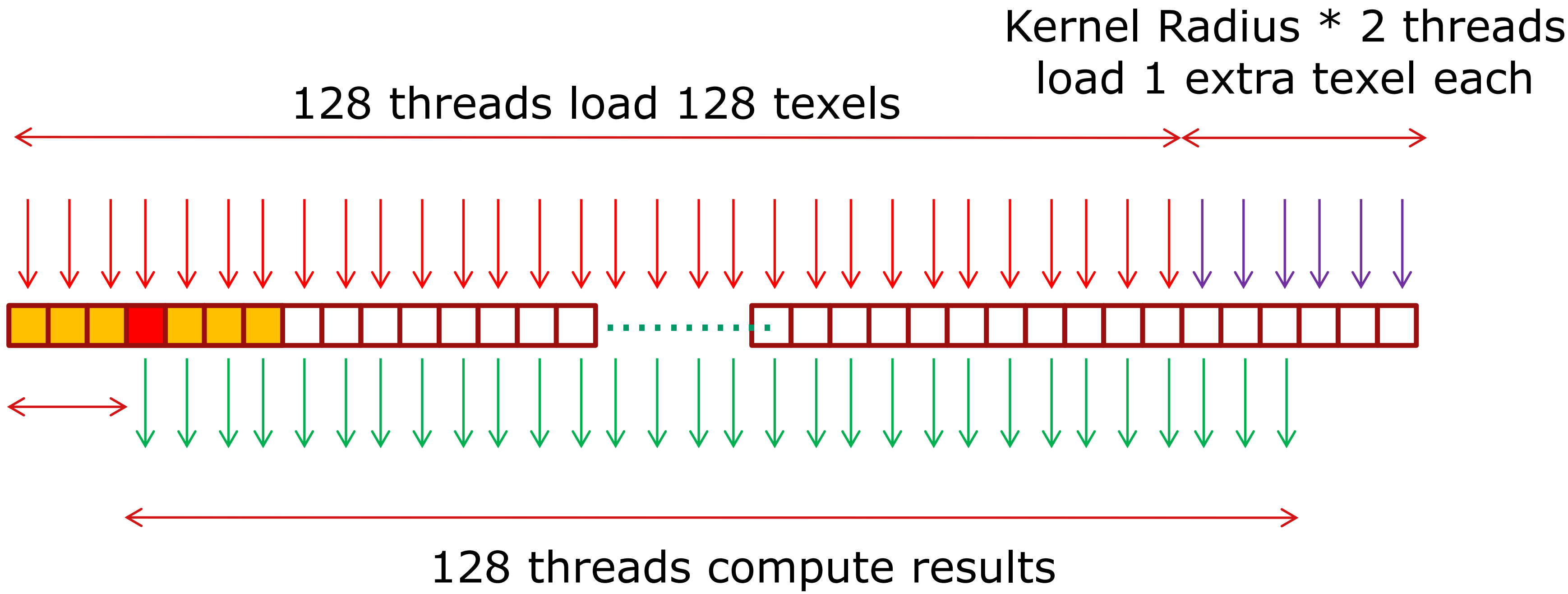


Redundant compute threads ☹

Avoid Redundant Threads

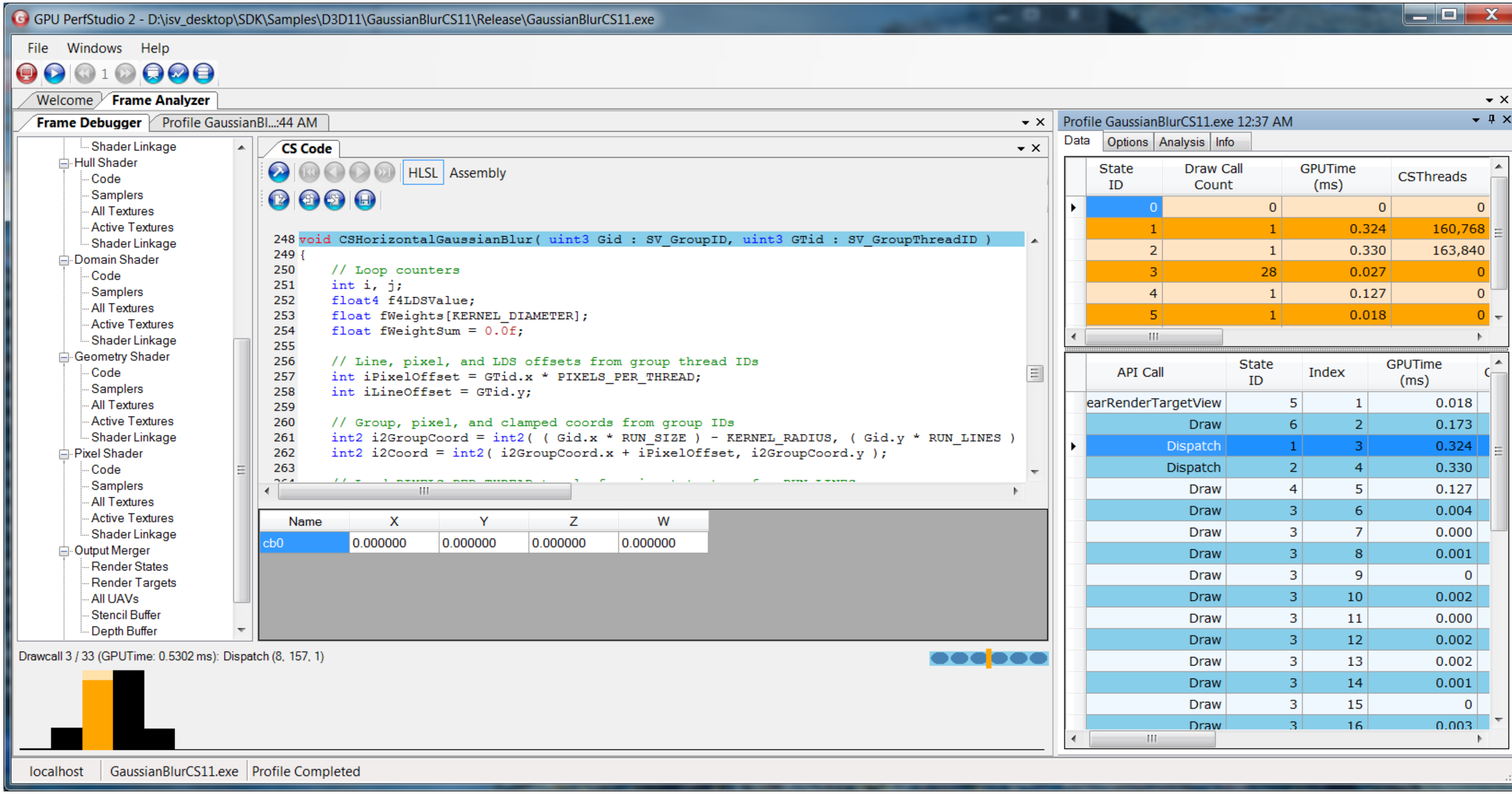
- Should ensure that all threads in a group have useful work to do – wherever possible
- Redundant threads will not be reassigned work from another group
- This would involve alot of redundancy for a large kernel diameter

A better use of Thread Group Shared Memory



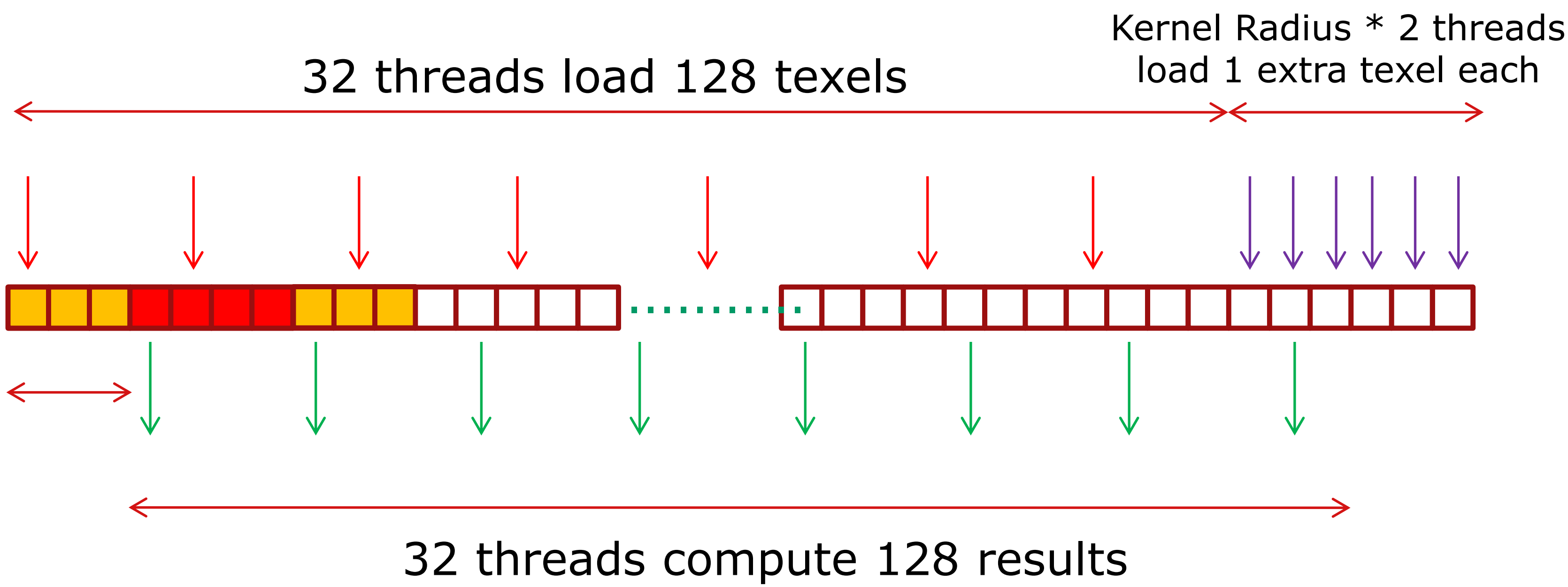
No redundant compute threads ☺

GPUPerfStudio: Separable Filter



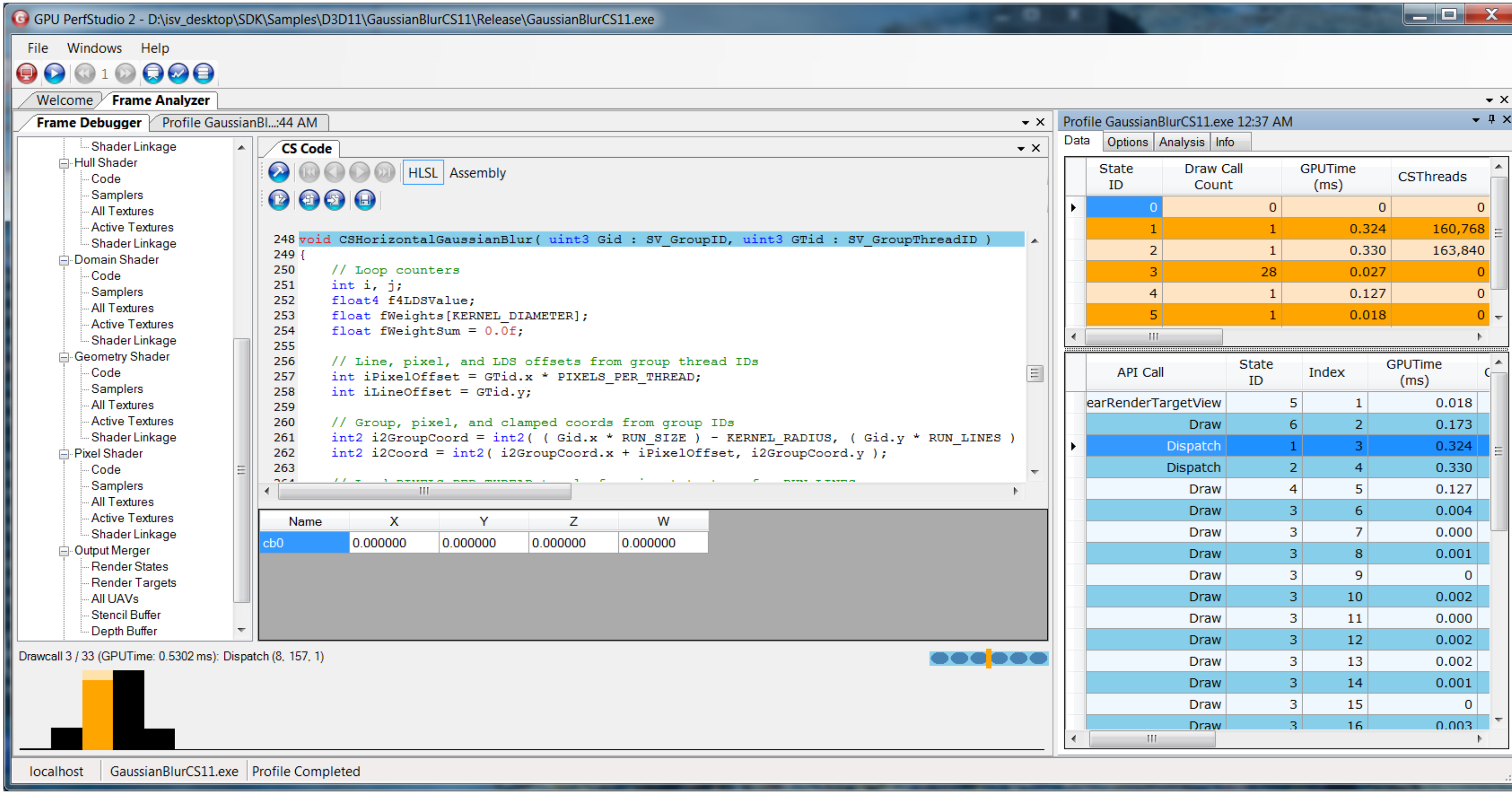
Optimization 3: Multiple Pixels per Thread

- Allows for natural vectorization
  - 4 works well on AMD HW (OK for scalar hardware too)
- Possible to cache TGSM reads on General Purpose Registers (GPRs)



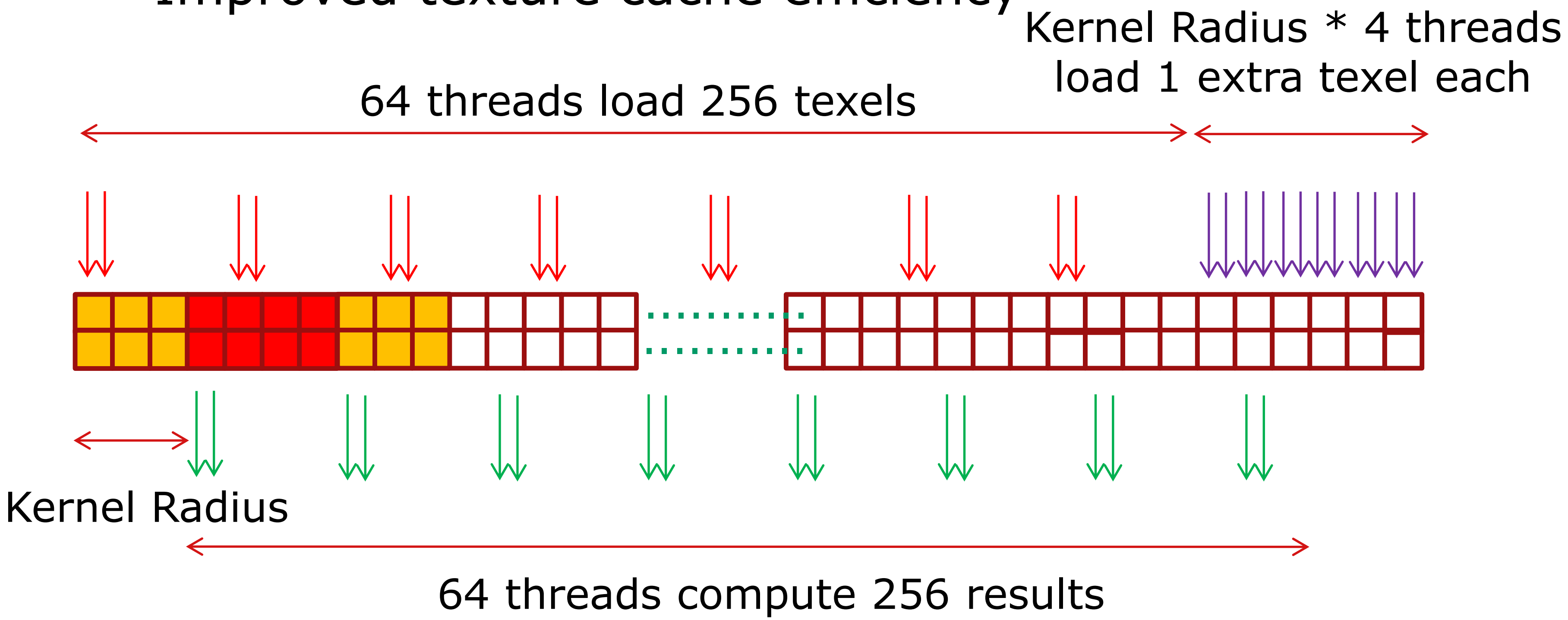
Compute threads not a multiple of 64 ☹

GPUPerfStudio: 4 Pixels Per Thread

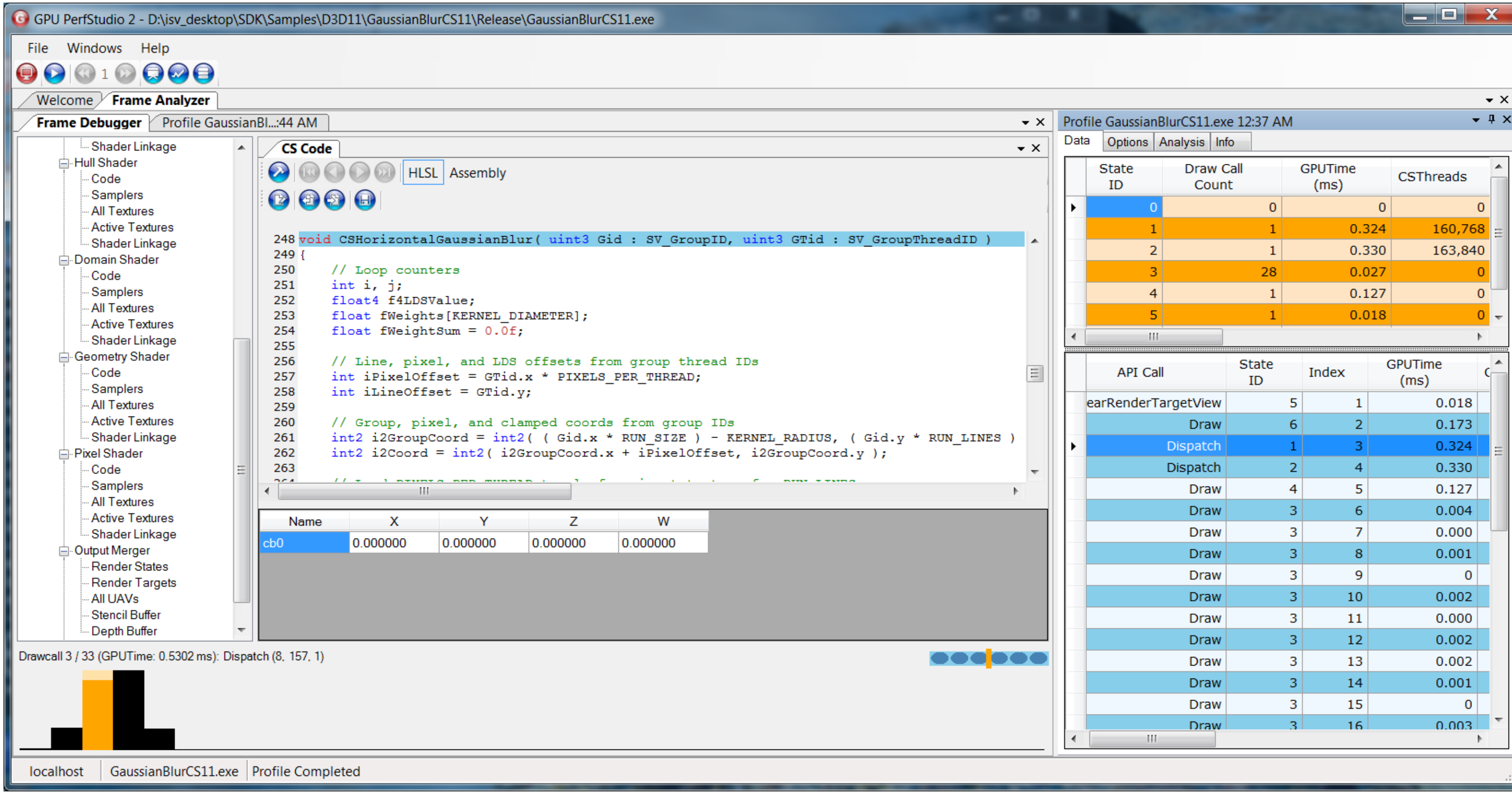


Optimization 4: 2D Thread Groups

- Process multiple lines per thread group
  - Thread group size is back to a multiple of 64
  - Better than one long line (2 or 4 works well)
- Improved texture cache efficiency



GPUPerfStudio: 2D Thread Groups



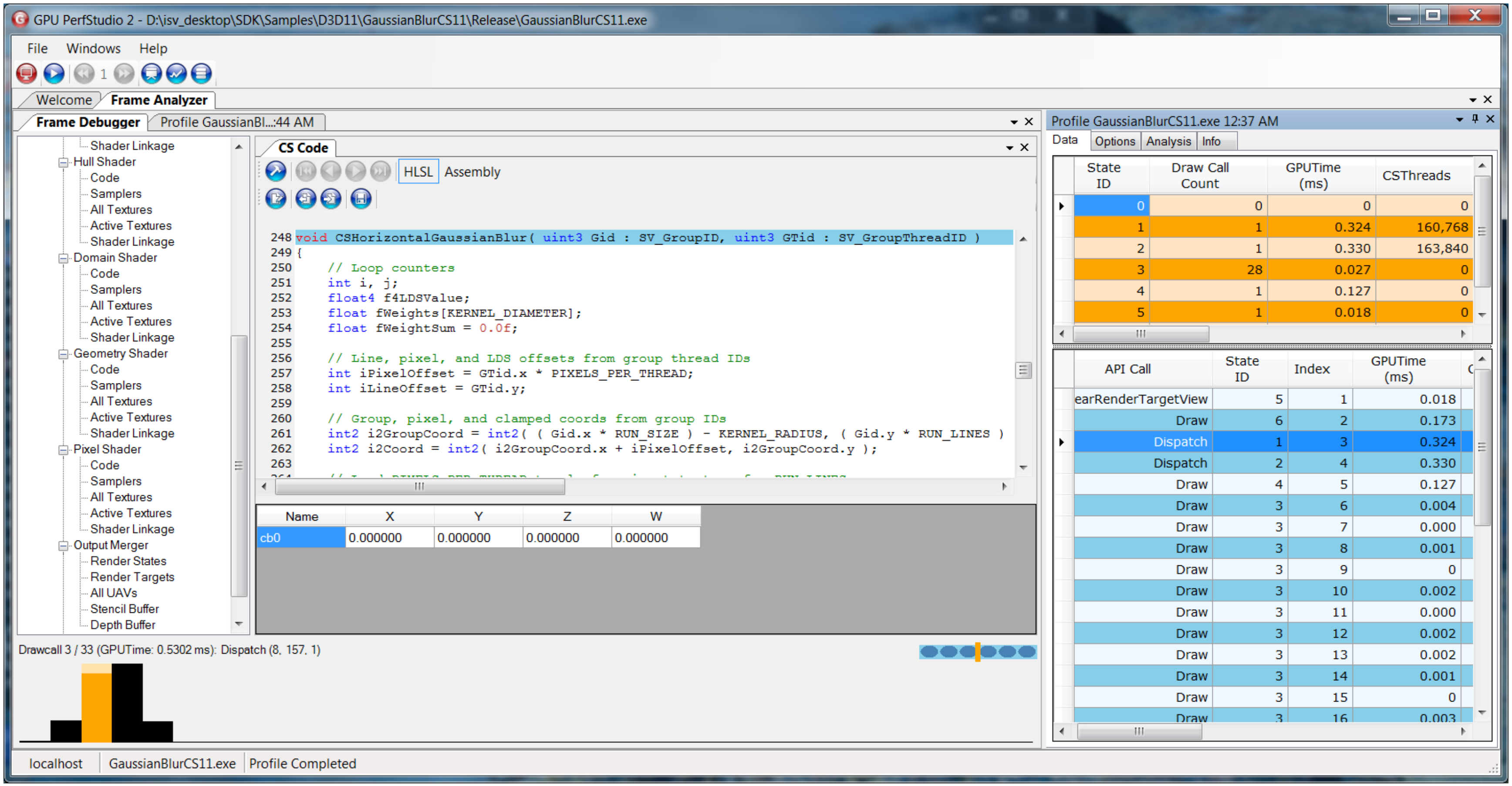
Kernel Diameter

- Kernel diameter needs to be > 7 to see a DirectCompute win
  - Otherwise the overhead cancels out the advantage
- The larger the kernel diameter the greater the win
- Large kernels also require more TGSM

Optimization 5: Use Packing in TGSM

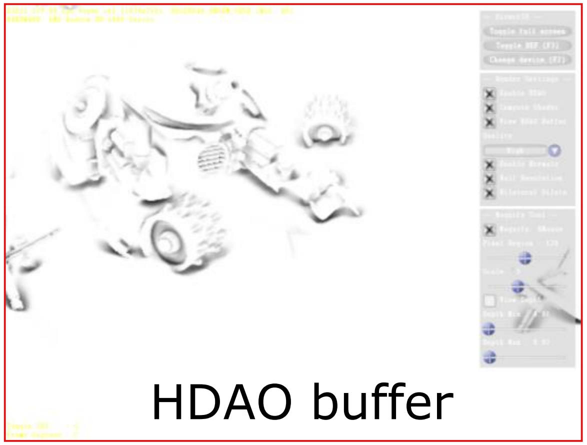
- Use packing to reduce storage space required in TGSM
  - Only have 32k per SIMD
- Reduces reads/writes from TGSM
- Often a uint is sufficient for color filtering
- Use SM5.0 instructions f32tof16(), f16tof32()

GPUPerfStudio: TGSM Packing



Example 2: High Definition Ambient Occlusion

Depth + Normals


HDAO buffer

*


Original Scene

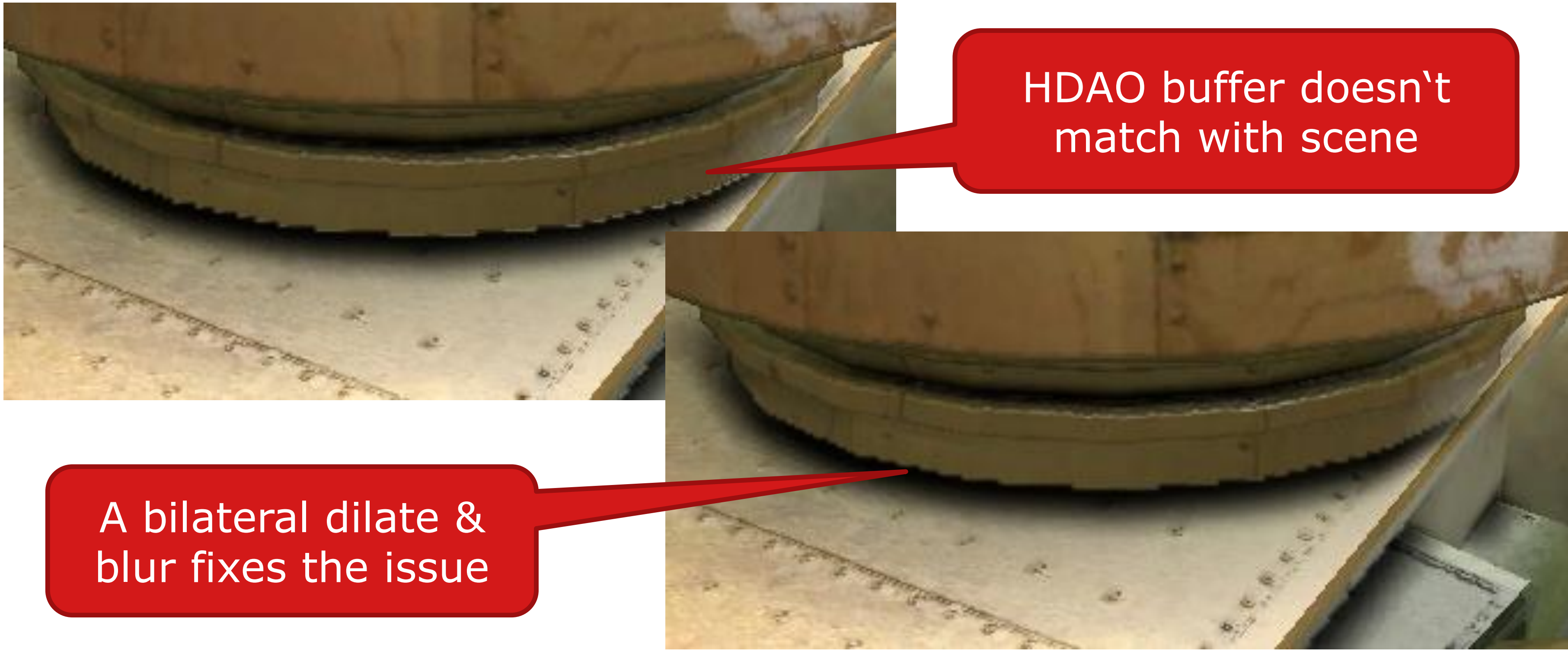
=


Final Scene

Optimization 6: Perform at Half Resolution

- HDAO at full resolution is expensive
- Running at half resolution captures more occlusion – and is obviously much faster
- Problem: Artifacts are introduced when combined with the full resolution scene

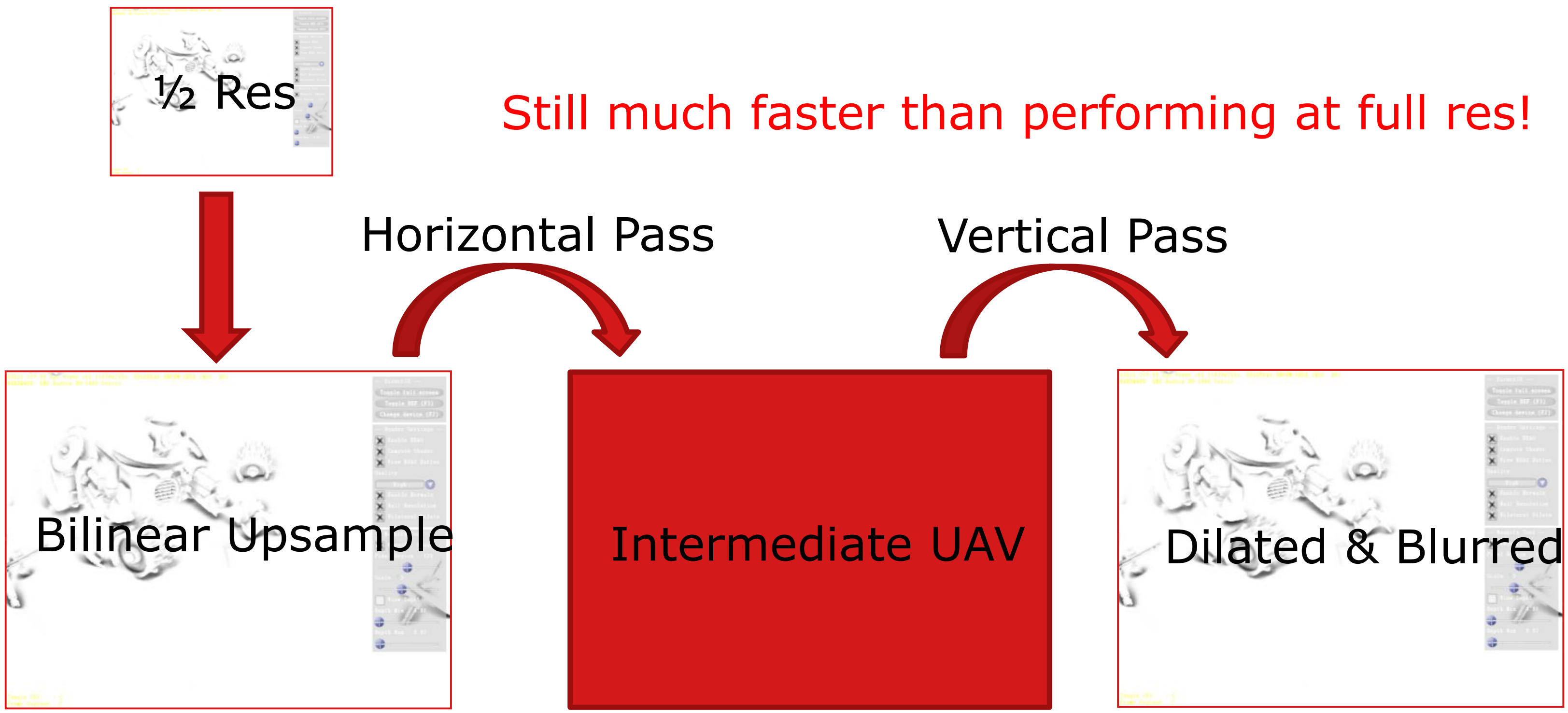
Bilateral Dilate & Blur



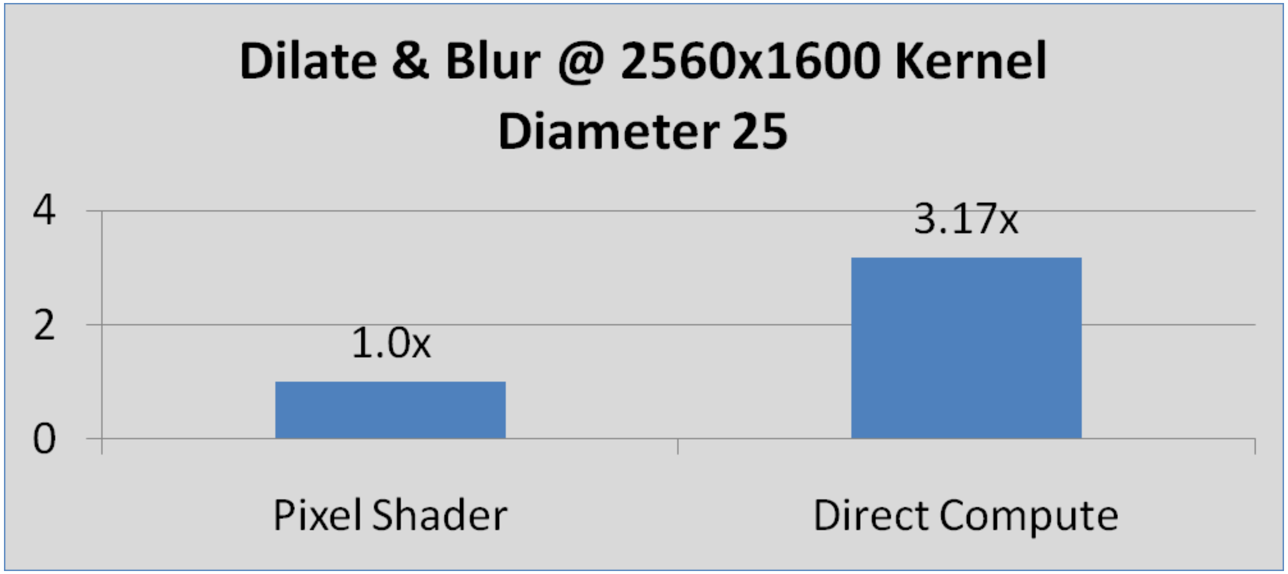
HDAO buffer doesn't match with scene

A bilateral dilate & blur fixes the issue

New Pipeline...



Pixel Shader vs DirectCompute



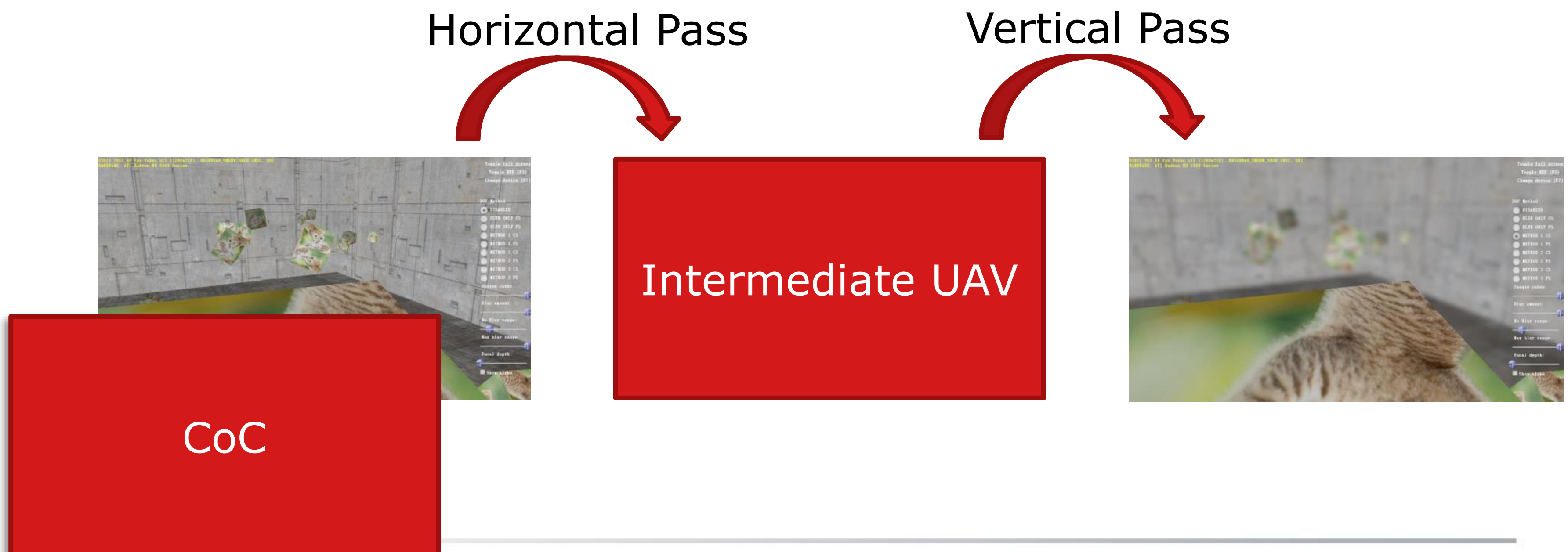
*Tested on a range of AMD and NVIDIA DX11 HW, DirectCompute is between ~2.53x to ~3.17x faster than the Pixel Shader

Example 3: Depth of Field

- Many techniques exist to solve this problem
- A common technique is to figure out how blurry a pixel should be
  - Often called the Cirle of Confusion (CoC)
- A Gaussian blur weighted by CoC is a pretty efficient way to implement this effect

Optimization 7: Combine filters

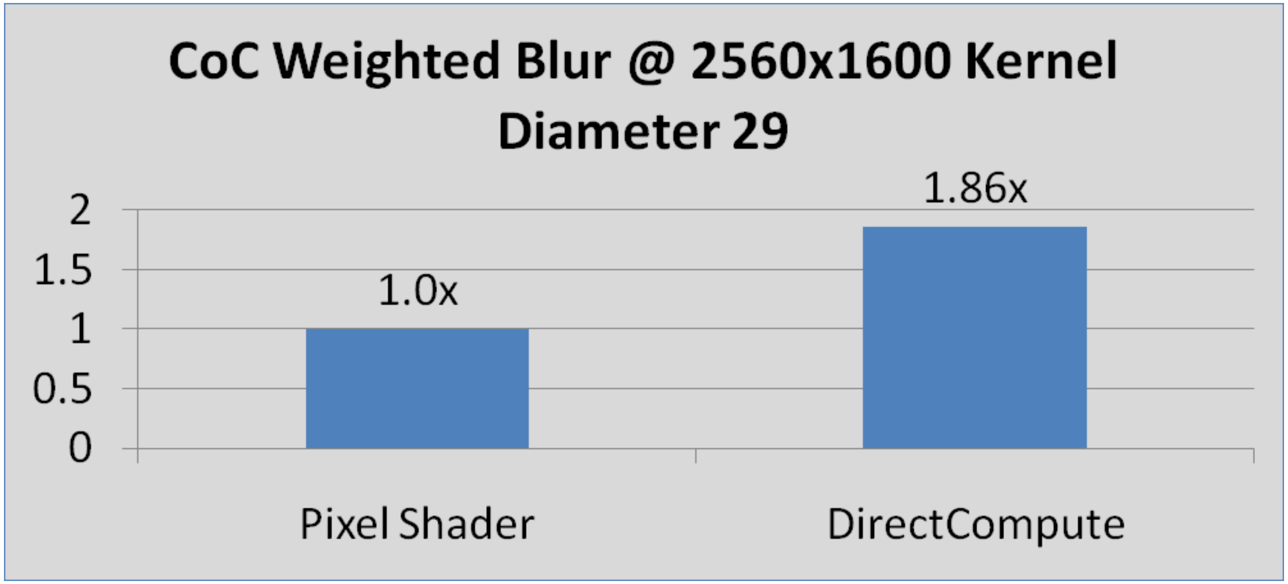
- Combined Gaussian Blur and CoC weighting isn't a separable filter, but we can still use a separate horizontal and vertical 1D pass
  - The result is acceptable in most cases



Shogun 2 – DOF Off

Shogun 2: DOF On

Pixel Shader vs DirectCompute



*Tested on a range of AMD and NVIDIA DX11 HW, DirectCompute is between ~1.48x to ~1.86x faster than the Pixel Shader

Summary

- Compute Shaders can provide big optimizations over pixel shaders if optimized correctly
- 7 Filter Optimizations presented
  - Separable Filters
  - Thread Group Shared Memory
  - Multiple Pixels per Thread
  - 2D Thread Groups
  - Packing in Thread Group Shared Memory
  - Half Res Filtering
  - Combined non-separable filter using separate passes
- AMD can provide examples for you to use.

Aknowledgements

Jon Story, AMD

- Slides, examples, and research

AMD The future is fusion

Questions?

bill.bilodeau@amd.com