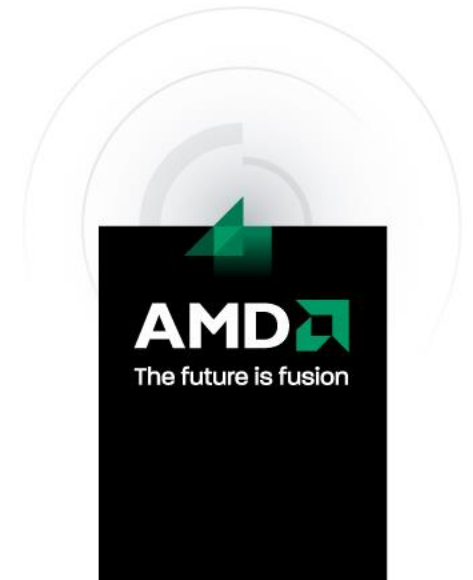


Efficient Compute Shader Programming

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 bilinear 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



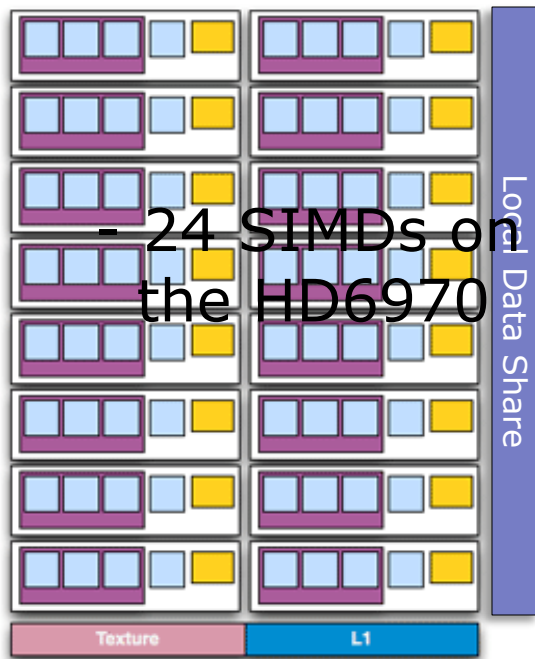
Direct Compute Features

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



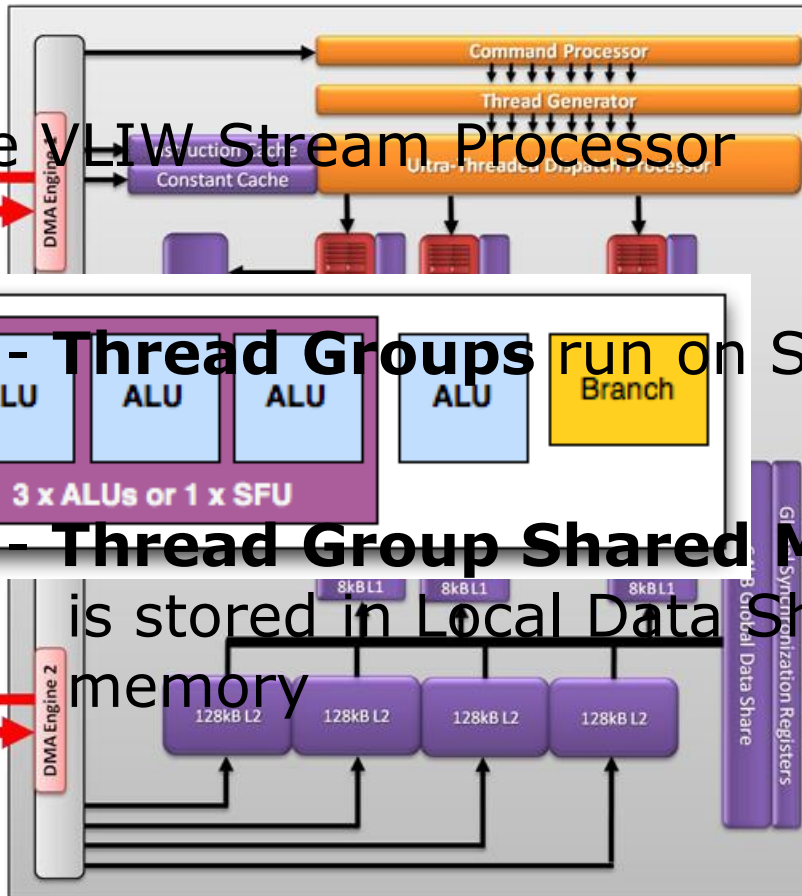
GPU Architecture Overview: HD6970

SIMD



- 24 SIMDs on the HD6970

4 - Wide VLIW Stream Processor



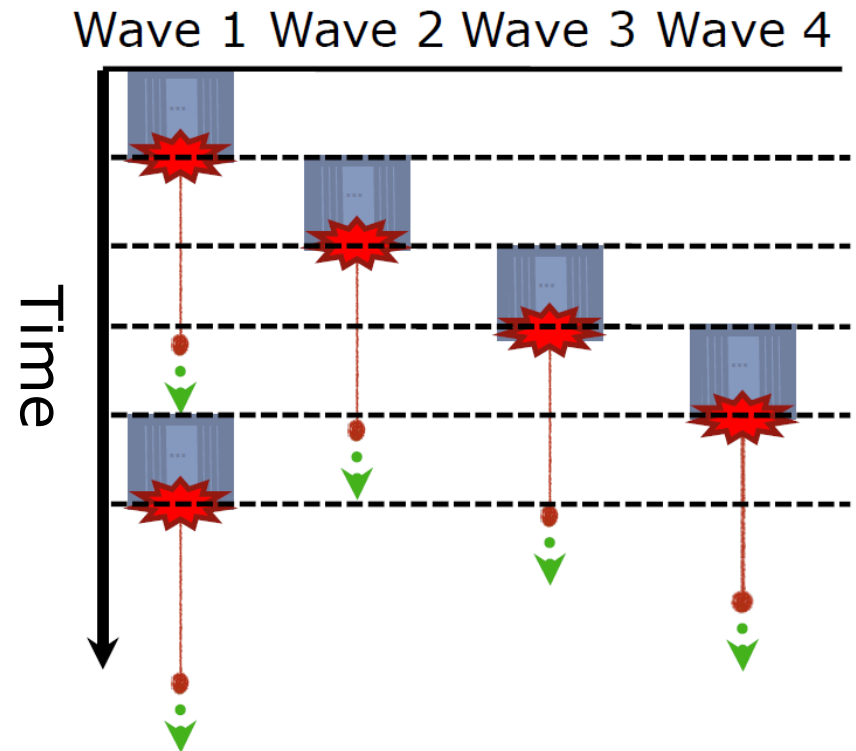
- Thread Groups run on SIMDs

- Thread Group Shared Memory is stored in Local Data Share memory



GPU Architecture Overview: Wavefronts

- GPU Time-slices execution to hide latency
- 1 Wavefront = 4 waves of threads per SP
- 16 SPs per SIMD, so $16 \times 4 = 64$ 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, $[(\text{Thread Group Size}) \bmod 64]$ threads go unused!
- Thread Group Shared Memory (LDS)
 - Limited to 32K per SIMD, so 32K per thread group
 - Memory is addressed in 32 banks. Addressing the same location, or $\text{loc} + (n \times 32)$ may cause bank conflicts.
- Vectorize your compute shader code
 - 4-way VLIW stream processors



Optimization Considerations

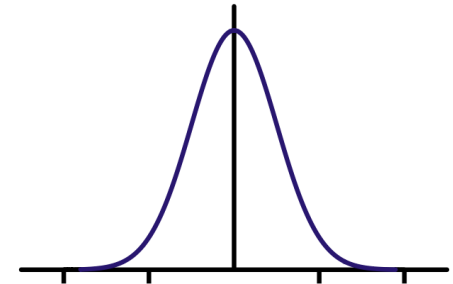
- Know what it is you're trying to optimize
 - TEX, ALU
 - GPUPerfStudio and GPU Shader Analyzer can help with this.
- Try lots of different configurations
 - Avoid hard-coding variables
 - Use GPUPerfStudio to edit in-place
- Avoid divergent dynamic flow control
 - Wastes shader processor cycles
- Know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



Example 1: Gaussian Blur

- Low-pass filter
 - Approximation of an ideal sinc
 - 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:

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

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

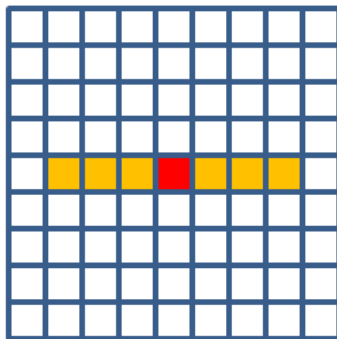
- Fewer TEX and ALU operations



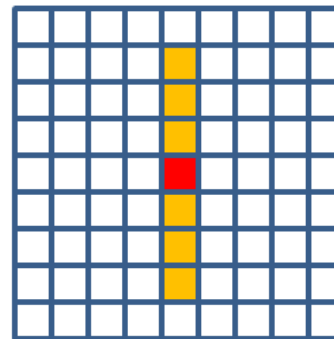
Typical Pipeline Steps



Horizontal Pass



Vertical Pass



Use Bilinear HW filtering?

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

- Just need to compute the correct sampling offsets

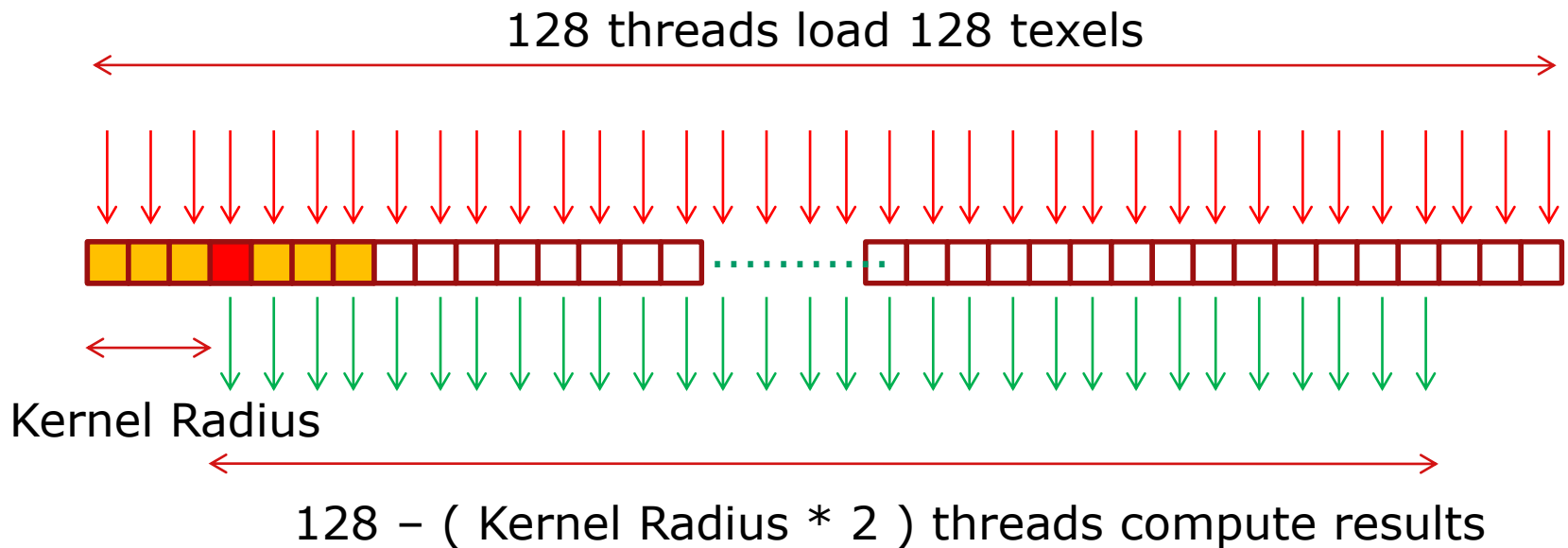
Not possible with more advanced filters

- 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 ☹



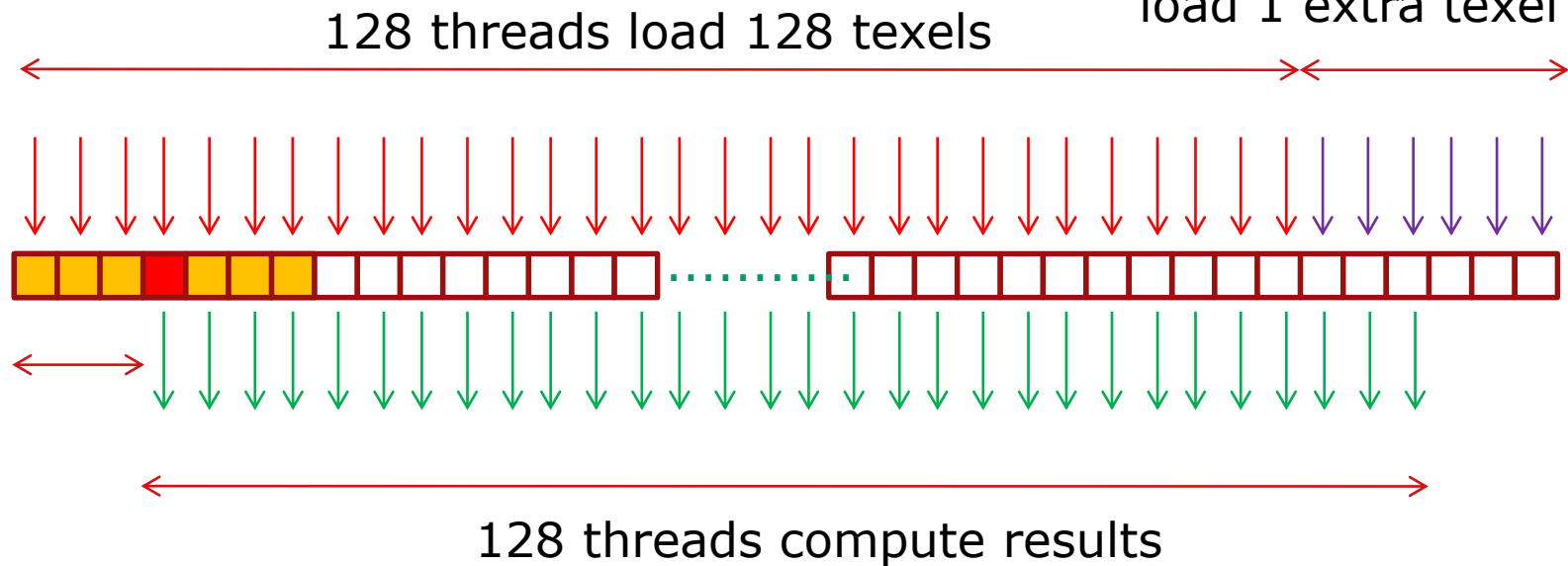
Avoid Redundant Threads

- 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 a lot of redundancy for a large kernel diameter



A better use of Thread Group Shared Memory

Kernel Radius * 2 threads
load 1 extra texel each



No redundant compute threads 😊



GPUPerfStudio: Separable Filter

GPU PerfStudio 2 - D:\sv_desktop\SDK\Samples\D3D11\GaussianBlurCS11\Release\GaussianBlurCS11.exe

File Windows Help

Welcome **Frame Analyzer**

Frame Debugger Profile GaussianBl...:44 AM

Shader Linkage

- Hull Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
 - Shader Linkage
- Domain Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
 - Shader Linkage
- Geometry Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
 - Shader Linkage
- Pixel Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
 - Shader Linkage
- Output Merger
 - Render States
 - Render Targets
 - All UAVs
 - Stencil Buffer
 - Depth Buffer

CS Code

HLSL Assembly

```
248 void CSHorizontalGaussianBlur( uint3 Gid : SV_GroupID, uint3 GTid : SV_GroupThreadID )
249 {
250     // Loop counters
251     int i, j;
252     float4 f4LDSValue;
253     float fWeights[KERNEL_DIAMETER];
254     float fWeightSum = 0.0f;
255
256     // Line, pixel, and LDS offsets from group thread IDs
257     int iPixelOffset = GTid.x * PIXELS_PER_THREAD;
258     int iLineOffset = GTid.y;
259
260     // Group, pixel, and clamped coords from group IDs
261     int2 i2GroupCoord = int2( ( Gid.x * RUN_SIZE ) - KERNEL_RADIUS, ( Gid.y * RUN_LINES )
262     int2 i2Coord = int2( i2GroupCoord.x + iPixelOffset, i2GroupCoord.y );
263
264     // ...
```

Name	X	Y	Z	W
cb0	0.000000	0.000000	0.000000	0.000000

Drawcall 3 / 33 (GPUTime: 0.5302 ms): Dispatch (8, 157, 1)

localhost GaussianBlurCS11.exe Profile Completed

Profile GaussianBlurCS11.exe 12:37 AM

Data Options Analysis Info

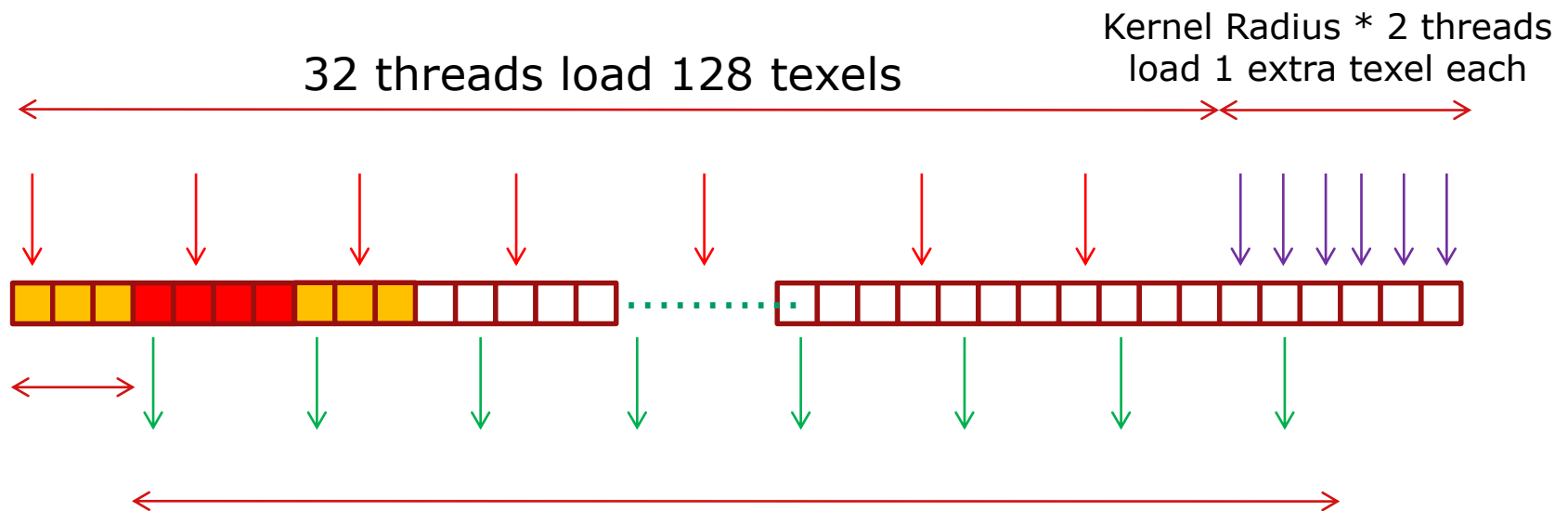
State ID	Draw Call Count	GPUTime (ms)	CSThreads
0	0	0	0
1	1	0.324	160,768
2	1	0.330	163,840
3	28	0.027	0
4	1	0.127	0
5	1	0.018	0

API Call	State ID	Index	GPUTime (ms)
ClearRenderTargetView	5	1	0.018
Draw	6	2	0.173
Dispatch	1	3	0.324
Dispatch	2	4	0.330
Draw	4	5	0.127
Draw	3	6	0.004
Draw	3	7	0.000
Draw	3	8	0.001
Draw	3	9	0
Draw	3	10	0.002
Draw	3	11	0.000
Draw	3	12	0.002
Draw	3	13	0.002
Draw	3	14	0.001
Draw	3	15	0
Draw	3	16	0.003



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)



32 threads compute 128 results

Compute threads not a multiple of 64 ☹️



GPUPerfStudio: 4 Pixels Per Thread

GPU PerfStudio 2 - D:\sv_desktop\SDK\Samples\D3D11\GaussianBlurCS11\Release\GaussianBlurCS11.exe

File Windows Help

Welcome **Frame Analyzer**

Frame Debugger Profile GaussianBl...:44 AM

Shader Linkage

- Hull Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
- Shader Linkage
- Domain Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
- Geometry Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
- Pixel Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
- Output Merger
 - Render States
 - Render Targets
 - All UAVs
 - Stencil Buffer
 - Depth Buffer

CS Code

HLSL Assembly

```
248 void CSHorizontalGaussianBlur( uint3 Gid : SV_GroupID, uint3 GTid : SV_GroupThreadID )
249 {
250     // Loop counters
251     int i, j;
252     float4 f4LDSValue;
253     float fWeights[KERNEL_DIAMETER];
254     float fWeightSum = 0.0f;
255
256     // Line, pixel, and LDS offsets from group thread IDs
257     int iPixelOffset = GTid.x * PIXELS_PER_THREAD;
258     int iLineOffset = GTid.y;
259
260     // Group, pixel, and clamped coords from group IDs
261     int2 i2GroupCoord = int2( ( Gid.x * RUN_SIZE ) - KERNEL_RADIUS, ( Gid.y * RUN_LINES )
262     int2 i2Coord = int2( i2GroupCoord.x + iPixelOffset, i2GroupCoord.y );
263
264     // ...
265 }
```

Table 1: State ID, Draw Call Count, GPU Time, CS Threads

State ID	Draw Call Count	GPU Time (ms)	CS Threads
0	0	0	0
1	1	0.324	160,768
2	1	0.330	163,840
3	28	0.027	0
4	1	0.127	0
5	1	0.018	0

Table 2: API Call, State ID, Index, GPU Time

API Call	State ID	Index	GPU Time (ms)
ClearRenderTargetView	5	1	0.018
Draw	6	2	0.173
Dispatch	1	3	0.324
Dispatch	2	4	0.330
Draw	4	5	0.127
Draw	3	6	0.004
Draw	3	7	0.000
Draw	3	8	0.001
Draw	3	9	0
Draw	3	10	0.002
Draw	3	11	0.000
Draw	3	12	0.002
Draw	3	13	0.002
Draw	3	14	0.001
Draw	3	15	0
Draw	3	16	0.003

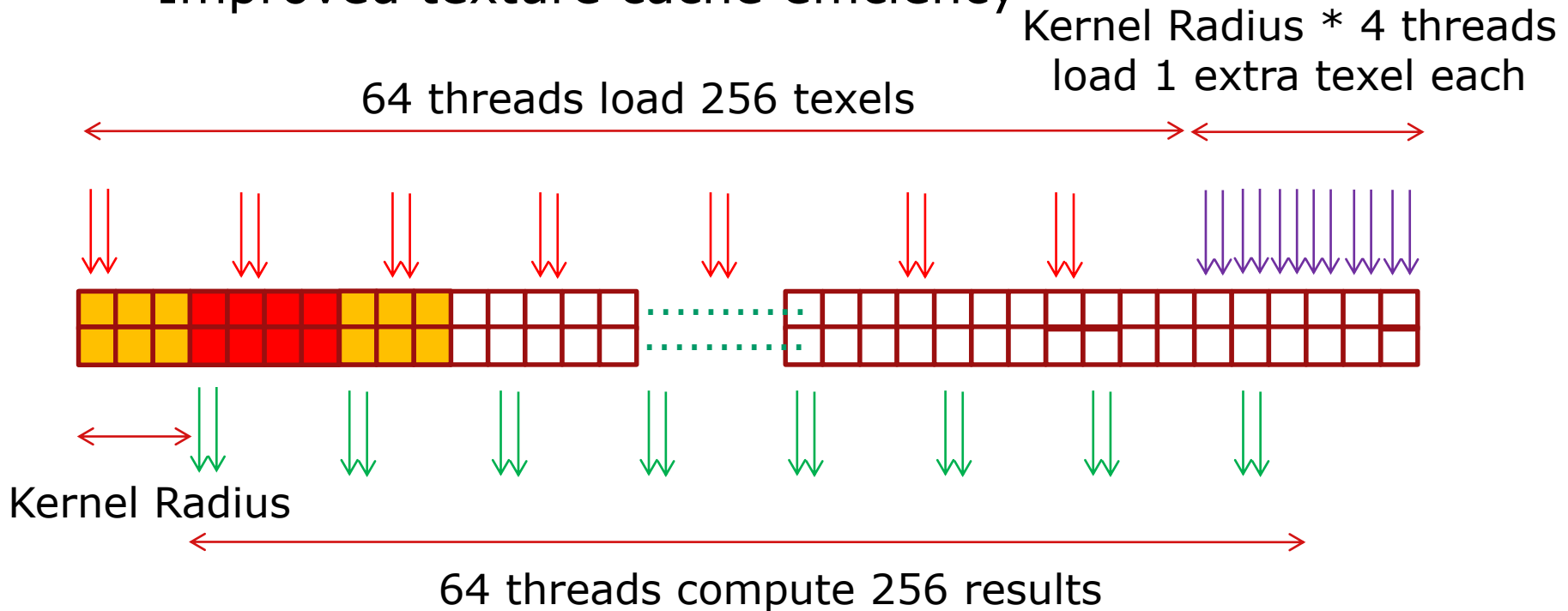
Drawcall 3 / 33 (GPU Time: 0.5302 ms): Dispatch (8, 157, 1)

localhost GaussianBlurCS11.exe Profile Completed



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

GPU PerfStudio 2 - D:\isv_desktop\SDK\Samples\D3D11\GaussianBlurCS11\Release\GaussianBlurCS11.exe

File Windows Help

Welcome Frame Analyzer

Frame Debugger Profile GaussianBl...:44 AM

Shader Linkage

- Hull Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
 - Shader Linkage
- Domain Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
 - Shader Linkage
- Geometry Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
 - Shader Linkage
- Pixel Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
 - Shader Linkage
- Output Merger
 - Render States
 - Render Targets
 - All UAVs
 - Stencil Buffer
 - Depth Buffer

CS Code

HLSL Assembly

```
248 void CSHorizontalGaussianBlur( uint3 Gid : SV_GroupID, uint3 GTid : SV_GroupThreadID )
249 {
250     // Loop counters
251     int i, j;
252     float4 f4LDSValue;
253     float fWeights[KERNEL_DIAMETER];
254     float fWeightSum = 0.0f;
255
256     // Line, pixel, and LDS offsets from group thread IDs
257     int iPixelOffset = GTid.x * PIXELS_PER_THREAD;
258     int iLineOffset = GTid.y;
259
260     // Group, pixel, and clamped coords from group IDs
261     int2 i2GroupCoord = int2( ( Gid.x * RUN_SIZE ) - KERNEL_RADIUS, ( Gid.y * RUN_LINES )
262     int2 i2Coord = int2( i2GroupCoord.x + iPixelOffset, i2GroupCoord.y );
263
264     // ...
```

Name	X	Y	Z	W
cb0	0.000000	0.000000	0.000000	0.000000

Drawcall 3 / 33 (GPUTime: 0.5302 ms): Dispatch (8, 157, 1)

localhost GaussianBlurCS11.exe Profile Completed

Profile GaussianBlurCS11.exe 12:37 AM

Data Options Analysis Info

State ID	Draw Call Count	GPUTime (ms)	CSThreads
0	0	0	0
1	1	0.324	160,768
2	1	0.330	163,840
3	28	0.027	0
4	1	0.127	0
5	1	0.018	0

API Call	State ID	Index	GPUTime (ms)
earRenderTargetView	5	1	0.018
Draw	6	2	0.173
Dispatch	1	3	0.324
Dispatch	2	4	0.330
Draw	4	5	0.127
Draw	3	6	0.004
Draw	3	7	0.000
Draw	3	8	0.001
Draw	3	9	0
Draw	3	10	0.002
Draw	3	11	0.000
Draw	3	12	0.002
Draw	3	13	0.002
Draw	3	14	0.001
Draw	3	15	0
Draw	3	16	0.003



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

GPU PerfStudio 2 - D:\jsv_desktop\SDK\Samples\D3D11\GaussianBlurCS11\Release\GaussianBlurCS11.exe

File Windows Help

Welcome | **Frame Analyzer**

Frame Debugger | Profile GaussianBl...:44 AM

Shader Linkage

- Hull Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
 - Shader Linkage
- Domain Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
 - Shader Linkage
- Geometry Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
 - Shader Linkage
- Pixel Shader
 - Code
 - Samplers
 - All Textures
 - Active Textures
 - Shader Linkage
- Output Merger
 - Render States
 - Render Targets
 - All UAVs
 - Stencil Buffer
 - Depth Buffer

CS Code

HLSL Assembly

```
248 void CSHorizontalGaussianBlur( uint3 Gid : SV_GroupID, uint3 GTid : SV_GroupThreadID )
249 {
250     // Loop counters
251     int i, j;
252     float4 f4LDSValue;
253     float fWeights[KERNEL_DIAMETER];
254     float fWeightSum = 0.0f;
255
256     // Line, pixel, and LDS offsets from group thread IDs
257     int iPixelOffset = GTid.x * PIXELS_PER_THREAD;
258     int iLineOffset = GTid.y;
259
260     // Group, pixel, and clamped coords from group IDs
261     int2 i2GroupCoord = int2( ( Gid.x * RUN_SIZE ) - KERNEL_RADIUS, ( Gid.y * RUN_LINES )
262     int2 i2Coord = int2( i2GroupCoord.x + iPixelOffset, i2GroupCoord.y );
263
264     // ...
```

Drawcall 3 / 33 (GPUPTime: 0.5302 ms): Dispatch (8, 157, 1)

localhost | GaussianBlurCS11.exe | Profile Completed

Profile GaussianBlurCS11.exe 12:37 AM

Data | Options | Analysis | Info

State ID	Draw Call Count	GPUPTime (ms)	CSThreads
0	0	0	0
1	1	0.324	160,768
2	1	0.330	163,840
3	28	0.027	0
4	1	0.127	0
5	1	0.018	0

API Call	State ID	Index	GPUPTime (ms)
earRenderTargetView	5	1	0.018
Draw	6	2	0.173
Dispatch	1	3	0.324
Dispatch	2	4	0.330
Draw	4	5	0.127
Draw	3	6	0.004
Draw	3	7	0.000
Draw	3	8	0.001
Draw	3	9	0
Draw	3	10	0.002
Draw	3	11	0.000
Draw	3	12	0.002
Draw	3	13	0.002
Draw	3	14	0.001
Draw	3	15	0
Draw	3	16	0.003



Example 2: High Definition Ambient Occlusion

Depth + Normals



*



=



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

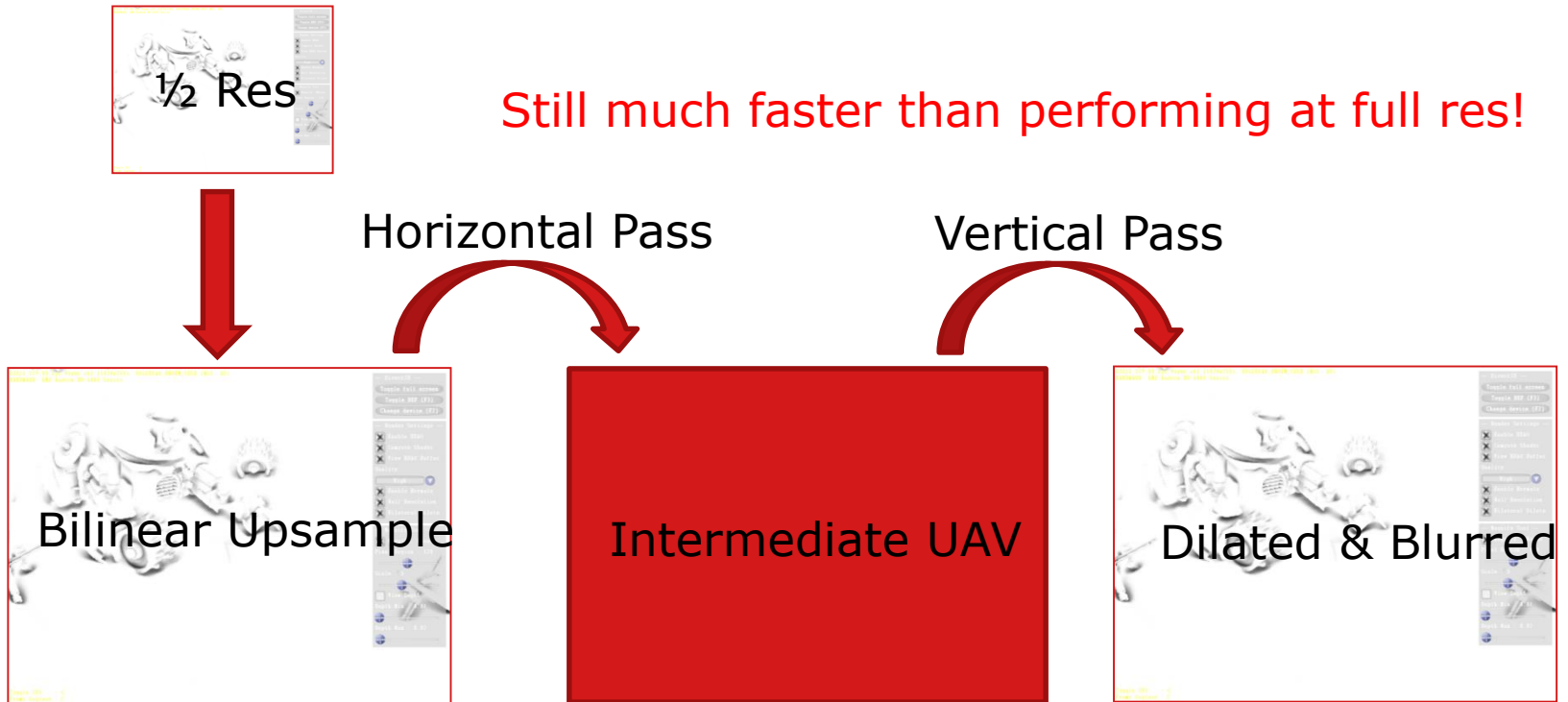


HDAO buffer doesn't match with scene

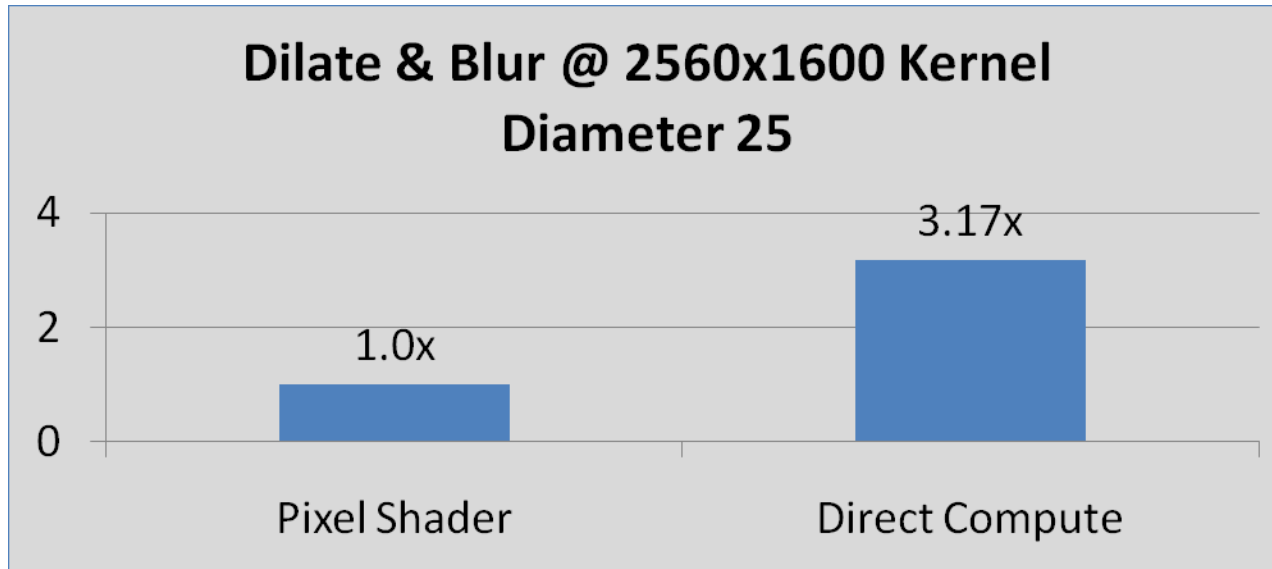
A bilateral dilate & blur fixes the issue



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



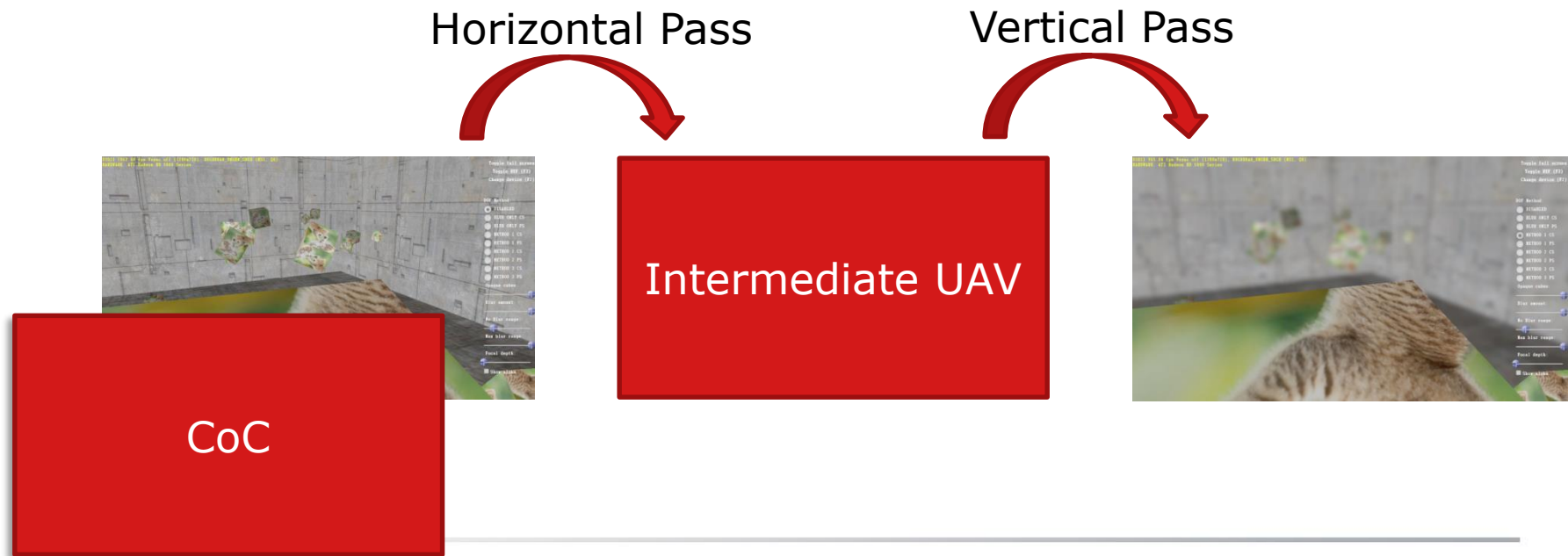
Example 3: Depth of Field

- Many techniques exist to solve this problem
- A common technique is to figure out how blurry a pixel should be
 - Often called the Circle 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



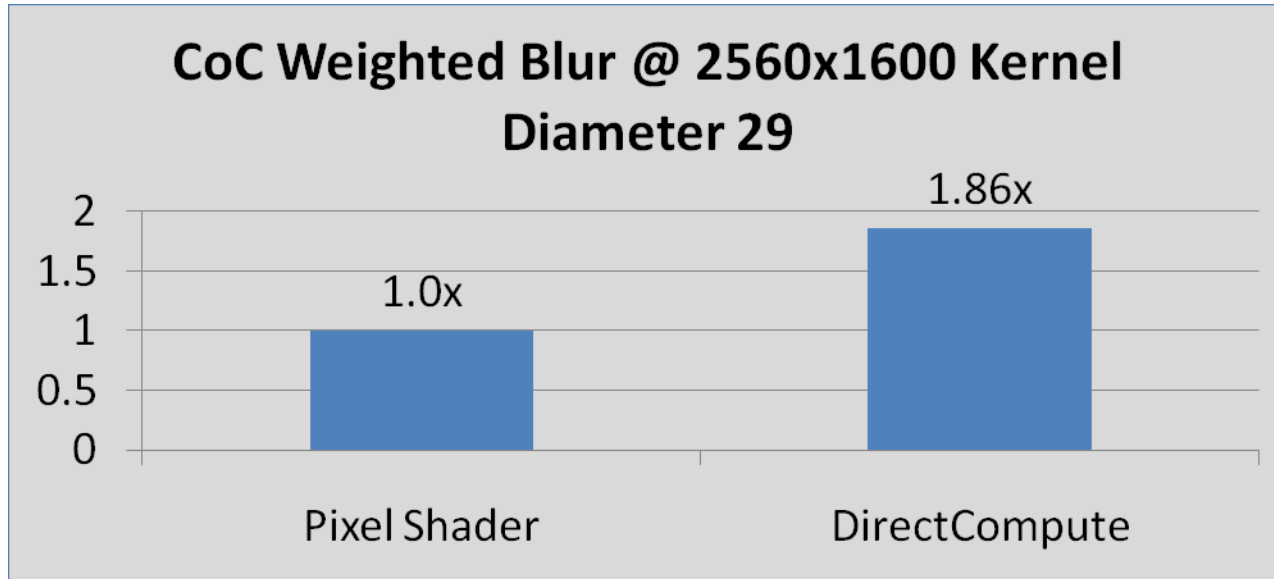
Shogun 2 – DOF Off



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.



Acknowledgements

Jon Story, AMD

- Slides, examples, and research



Questions?

bill.bilodeau@amd.com

