Game Developers Conference® | March 9-13, 2010 | Moscone Center | San Francisco, CA

THINK™ SERVICES
A DIVISION OF UNITED BUSINESS MEDIA LLC

GDC 10

Learn. Network. Inspire.

www.GDConf.com

Game Developers Conference®
March 9-13, 2010
Moscone Center
San Francisco, CA
www.GDConf.com

Direct3D 11 Performance Tips & Tricks

Holger Gruen AMD ISV Relations
Cem Cebenoyan NVIDIA ISV Relations

Agenda

- Introduction
- Shader Model 5
- Resources and Resource Views
- Multithreading
- Miscellaneous
- Q&A

Introduction

- Direct3D 11 has numerous new features
- However these new features need to be used wisely for good performance
- For generic optimization advice please refer to last year‘s talk http://developer.amd.com/gpu_assets/The A to Z of DX10 Performance.pps

Shader Model 5 (1)

- Use Gather*/GatherCmp*() for fast multi-channel texture fetches
  - Use smaller number of RTs while still fetching efficiently
    - Store depth to FP16 alpha for SSAO
      - Use Gather*() for region fetch of alpha/depth
  - Fetch 4 RGB values in just three ops
    - Image post processing

Fetch 4 RGB values in just three texture ops

red0 green0 blue0 alpha0	red1 green1 blue1 alpha1
red2 green2 blue2 alpha2	red3 green3 blue3 alpha3

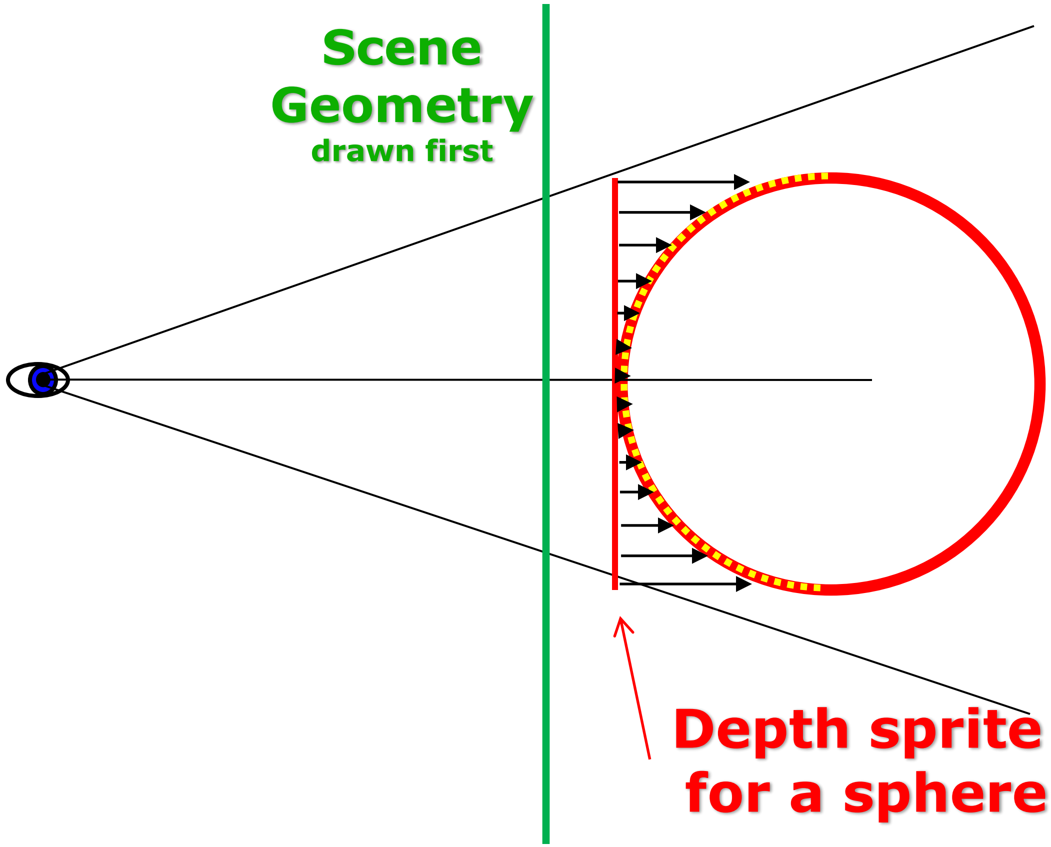
SampleOp0	red0	green0	blue0	alpha0
SampleOp1	red1	green1	blue1	alpha1
SampleOp2	red2	green2	blue2	alpha2
SampleOp3	red3	green3	blue3	alpha3

GatherRed	red2	red3	red1	red0
GatherGreen	green2	green3	green1	green0
GatherBlue	blue2	blue3	blue1	blue0

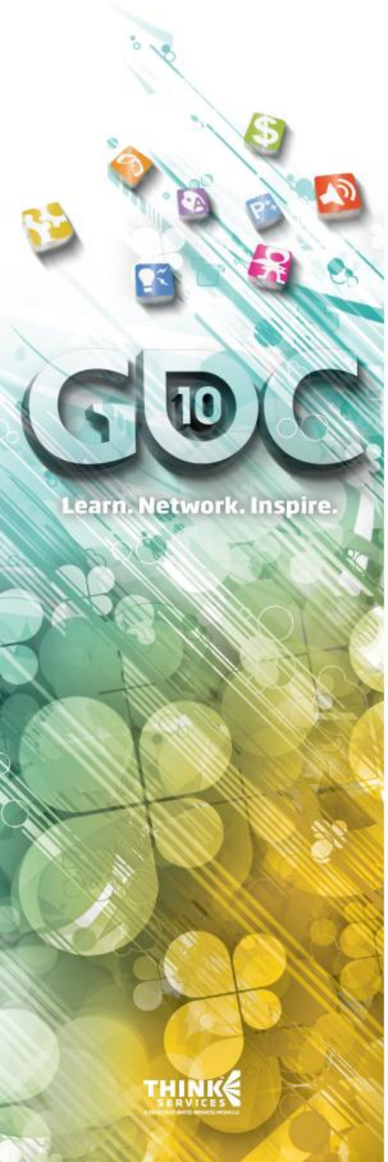
Shader Model 5 (2)

- Use ‘Conservative Depth’ to keep early depth rejection active for fast depth sprites
  - Output SV_DepthGreater/LessEqual instead of SV_Depth from your PS
    - Keeps early depth rejection active even with shader-modified Z
  - The hardware/driver will enforce legal behavior
    - If you write an invalid depth value it will be clamped to the rasterized value

Depth Sprites under Direct3D 11



Direct3D 11 can fully cull this depth sprite if SV_DepthGreaterEqual is output by the PS



Shader Model 5 (3)

- Use EvaluateAttribute*() for fast shader AA without super sampling
  - Call EvaluateAttribute*() at subpixel positions
    - Simpler shader AA for procedural materials
  - Input SV_COVERAGE to compute a color for each covered subsample and write average color
    - Slightly better image quality than pure MSAA
  - Output SV_Coverage for MSAA alpha-test
    - This feature has been around since 10.1
    - EvaluateAttribute*() makes implementation simpler
    - But check if alpha to coverage gives you what you need already, as it should be faster.

Shader Model 5 (4)

- A quick Refresher on UAVs and Atomics
  - Use PS scattering and UAVs wisely
  - Use Interlocked*() Operations wisely
  - See DirectCompute performance presentation!

Shader Model 5 (5)

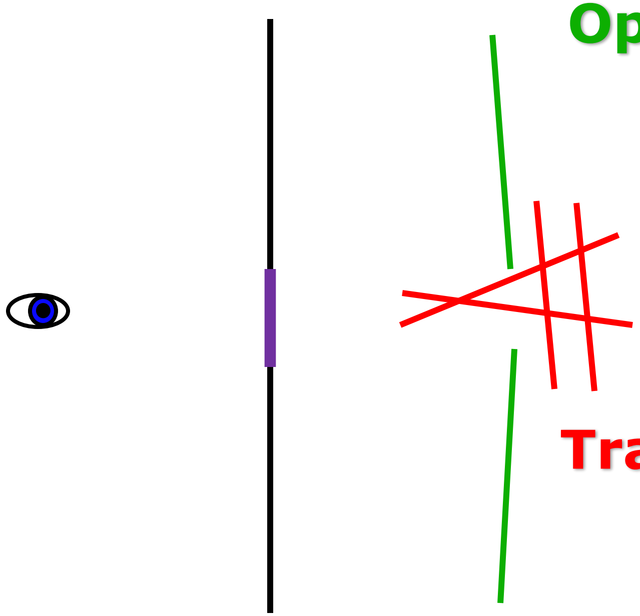
- Reduce stream out passes
  - Addressable stream output
  - Output to up to 4 streams in one pass
  - All streams can have multiple elements
- Write simpler code using Geometry shader instancing
  - Use SV_SInstanceID instead of loop index

Shader Model 5 (6)

- Force early depth-stencil testing for your PS using [earlydepthstencil]
  - Can introduce significant speedup specifically if writing to UAVs or AppendBuffers
    - AMD‘s OIT demo uses this
  - Put ‘[earlydepthstencil]’ above your pixel shader function declaration to enable it

Early Depth Stencil and OIT

Projection Plane



Opaque Geometry
drawn first

Transparent Geometry
Drawn after all
opaque Geometry

A ‘[earlydepthstencil]’ pixel shader that writes OIT color layers to a UAV only will cull all pixels outside the purple area!

Shader Model 5 (7)

- Use the numerous new intrinsics for faster shaders
  - Fast bitops – countbits(), reversebits() (needed in FFTs), etc.
  - Conversion instructions - fp16 to fp32 and vice versa (f16to32() and f32to16())
    - Faster packing/unpacking
  - Fast coarse deriatives (ddx/y_coarse)
  - …

Shader Model 5 (8)

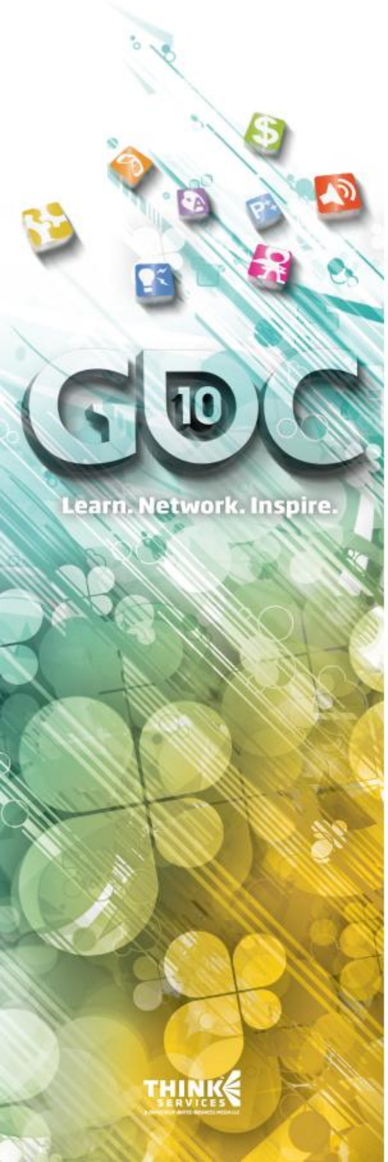
- Use Dynamic shader linkage of subroutines wisely
  - Subroutines are not free
    - No cross function boundary optimizations
  - Only use dynamic linkage for large subroutines
    - Avoid using a lot of small subroutines

Resources and Resource Views (1)

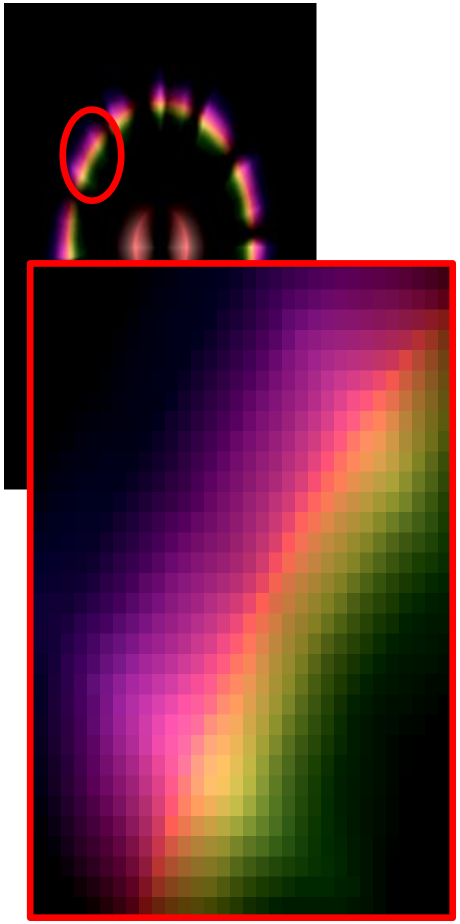
- Reduce memory size and bandwidth for more performance

  BC6 and BC7 provide new capabilities

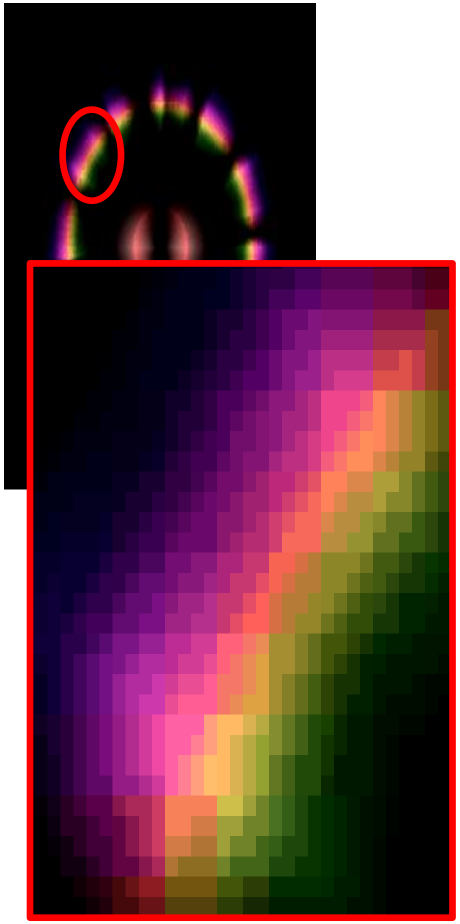
  - Very high quality, and HDR support
  - All static textures should now be compressible



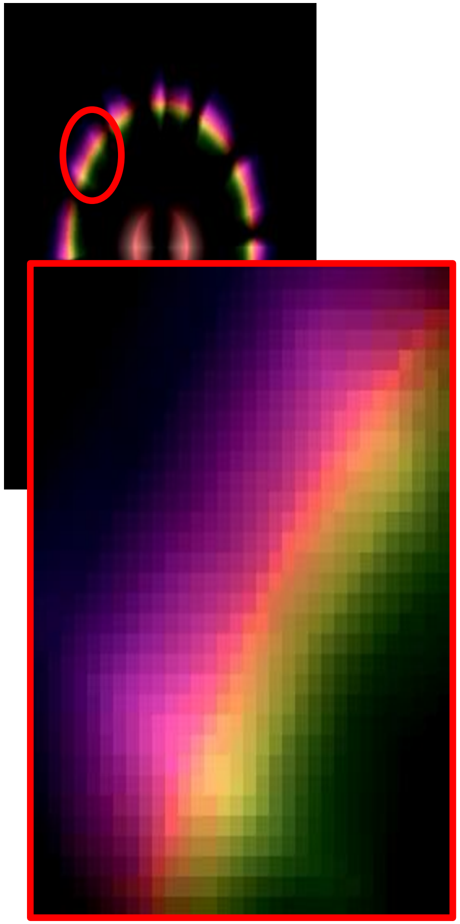
BC7 image quality



Original Image



BC1 Compressed



BC7 Compressed

Resources and Resource Views (2)

- Use Read-Only depth buffers to avoid copying the depth buffer

    Direct3D 11 allows the sampling of a depth buffer still bound for depth testing

    - Useful for deferred lighting if depth is part of the g-buffer
    - Useful for soft particles

    AMD: Using a depth buffer as a SRV may trigger a decompression step

    - Do it as late in the frame as possible

Free Threaded Resource Creation

- Use fast Direct3D 11 asynchronous resource creation

  In general it should just be faster and more parallel

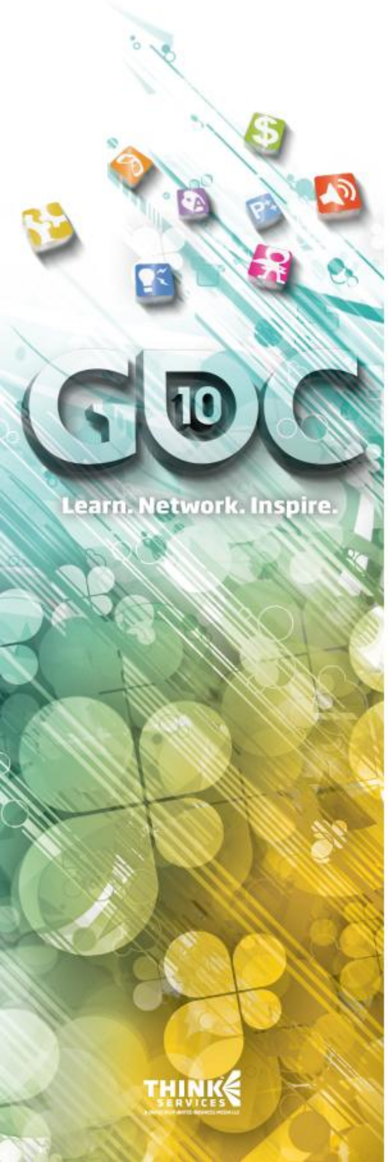
- Do not destroy a resource in a frame in which it’s used

  Destroying resources would most likely cause synchronizing events

- Avoid create-render-destroy sequences

Display Lists (aka command lists created from a deferred context)

- First make sure your app is multi-threaded well
- Only use display lists if command construction is a large enough bottleneck
- Now consider display lists to express parallelism in GPU command construction
    - Avoid fine grained command lists
- Drivers are already multi-threaded



Deferred Contexts

- On deferred contexts Map() and UpdateSubResource() will use extra memory
  - Remember, all initial Maps need to use the DISCARD semantic
- Note that on a single core system a deferred context will be slower than just using the immediate context
  - For dual core, it is also probably best to just use the immediate context
- Don’t use Deferred Contexts unless there is significant parallelism

Miscellaneous

- Use DrawIndirect to further lower your CPU overhead

  Kick off instanced draw calls/dispatch using args from a GPU written buffer

  - Could use the GPU for limited scene traversal and culling
- Use Append/Consume Buffers for fast ’stream out‘
  - Faster than GS as there are no input ordering constraints
  - One pass SO with ’unlimited‘ data amplification

Questions?

holger.gruen@amd.com
cem@nvidia.com