High Quality Graphic Effects using DX11

杨雪青(Young Yang), NVIDIA

第四届中国游戏开发者大会
China Game Developers Conference 2011
中国游戏开发者大会
汉威信恒 HOWELL EXPO

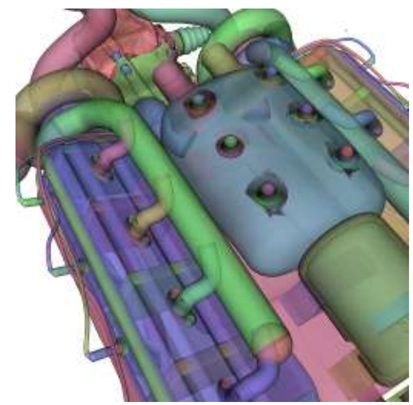
Topics Covered

SSAO using Compute Shaders

Opacity Mapping

Stochastic Transparency

FXAA



The topics in this talk and more great samples can be found here
http://developer.nvidia.com/nvidia-graphics-sdk-11

SSAO using Compute Shaders

gameworks.nvidia.com

Features Covered

- Using CS to implement high quality post process effects

DirectCompute

- For general purpose computing
- New shader mode that operates on arbitrary threads
- Frees processing from restrictions of gfx pipeline
- Full access to conventional Direct3D resources

New Resource Types

- Buffers / Structured Buffers
- Unordered Access Views (UAV)
  - RWTexture/RWBuffer
  - Allows arbitrary reads and writes from PS and CS

New Intrinsic Operations

- Atomic operations

  Such as, InterlockedAdd, InterlockedAnd, InterlockedMax, InterlockedMin, InterlockedOr …

  – Allow parallel workloads to combine results easily
  – Not free! Cost increases if a value is hit often

- Append/Consume

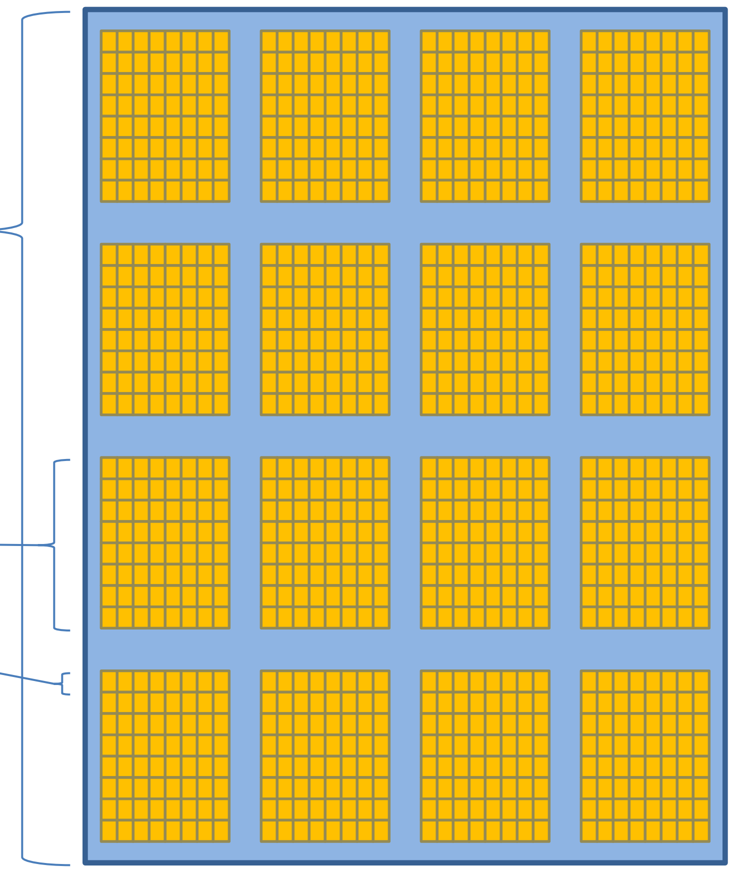
Threads, Groups, and Dispatches

- Shaders run as a set of several thread blocks that execute in parallel
  - “Threads” runs the code given in the shader
  - “Groups” are sets of threads that can communicate using on-chip memory
  - “Dispatches” are sets of groups
- Compute Shaders are invoked by API Dispatch(), just like the draw calls invoke the rendering.

Dispatch Example

```
// CPU Code
pContext->CSSetUnorderedAccessViews(
          0, 1, &pOutputUAV, NULL);
pContext->CSSetShader(pSimpleCS);
pContext->Dispatch(4,4,1);
```

```
// HLSL Code
RWTexture<float3> uavOut : register(u0);
[numthreads(8,8,1)]
void SimpleCS(uint3 tID :
                    SV_DispatchThreadID)
{
    uavOut[tID] = float3(0,1,0);
}
```



Memory Hierarchy

Memory Space	Speed	Visibility
Global Memory (Buffers, Textures, Constants)	Longest Latency	All Threads
Shared Memory (groupshared)	Fast	Single Group
Local Memory (Registers)	Very Fast	Single Thread

Take Advantage Of Shared Memory

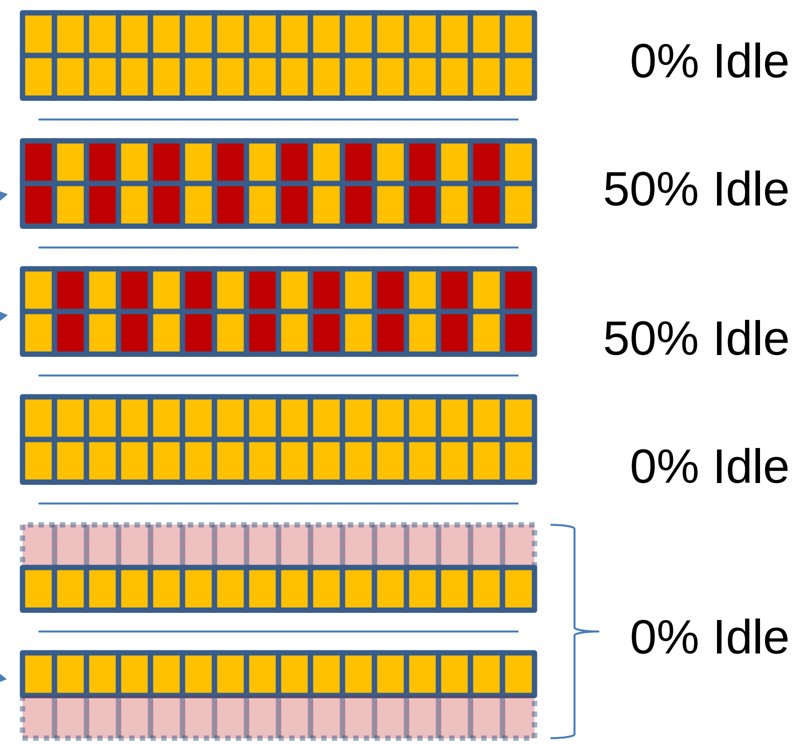
- Compute threads can communicate and share data via "`groupshared`" memory
  - Pre-load data used by every thread in a group
    - Save bandwidth and computation
  - Share workload for common tasks

Optimization Concepts

- Caching behavior depends on access mode
  - Buffers may hit cache better for linear accesses
  - Textures work better for less predictable/more 2D access within a group
- Divergence
  - Theoretically, threads execute independantly, Practically, they execute in parallel warps
  - Threads are “masked” for instructions in untaken branches while warp executes
  - Warps size is hardware specific , NV : 32

Warp Divergence

```
// Assume WARP_SIZE = warp size for
// the hardware (NV:32)
[numthreads(WARP_SIZE,2,1)]
void DivergeCS(uint3 tID :
          SV_DispatchThreadID) {
  float val;
  // Divergent
  if (tID.x%2)
    val = ComplexFuncA(tID);
  else
    val = ComplexFuncB(tID);
  // Not divergent
  if (tID.y%2)
    val =+ ComplexFuncC(tID);
  else
    val =+ ComplexFuncD(tID);
  Output(tID, val);
}
```



Group Dispatch Calls

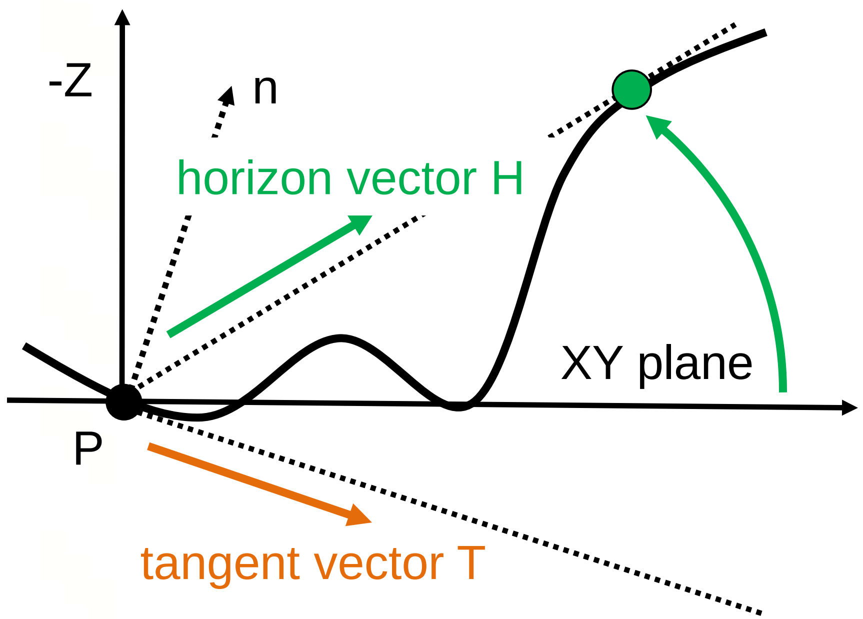
- Have to be aware of context switch cost
  - Penalty switching between Graphics and Compute
  - Usually minimal unless repeatedly hit
- Re-binding a resource used as UAV may stall HW to avoid data hazards
  - Have to make sure all writes complete so they are visible to next dispatch
  - Driver may re-order unrelated Dispatch calls to hide this latency

HBAO

- AO: Calculate how much sky can be seen from a point
  - HBAO: Horizon Based Ambient Occlusion
    Figure out the hight of the horizons around the point, and use these values to decide how much the point's lighted by the sky.

gameworks.nvidia.com

HBAO



horizon angle in [-π/2, π/2]

$h(H) = atan(H.z / ||H.xy||)$

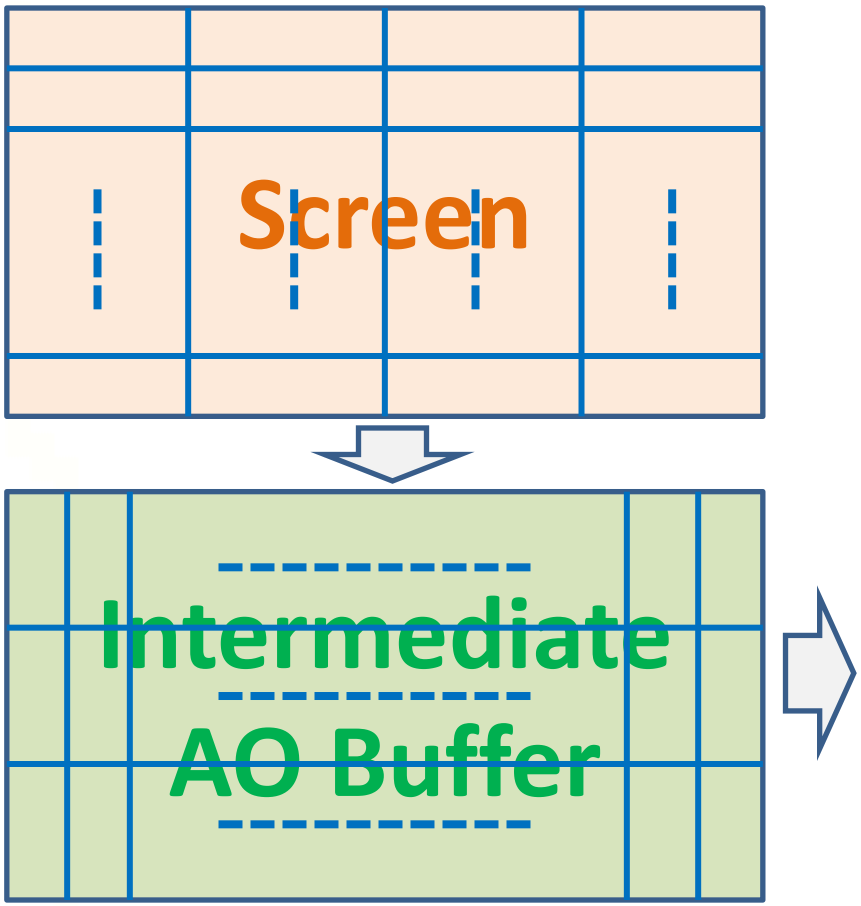
tangent angle in [-π/2, π/2]

$t(T) = atan(T.z / ||T.xy||)$

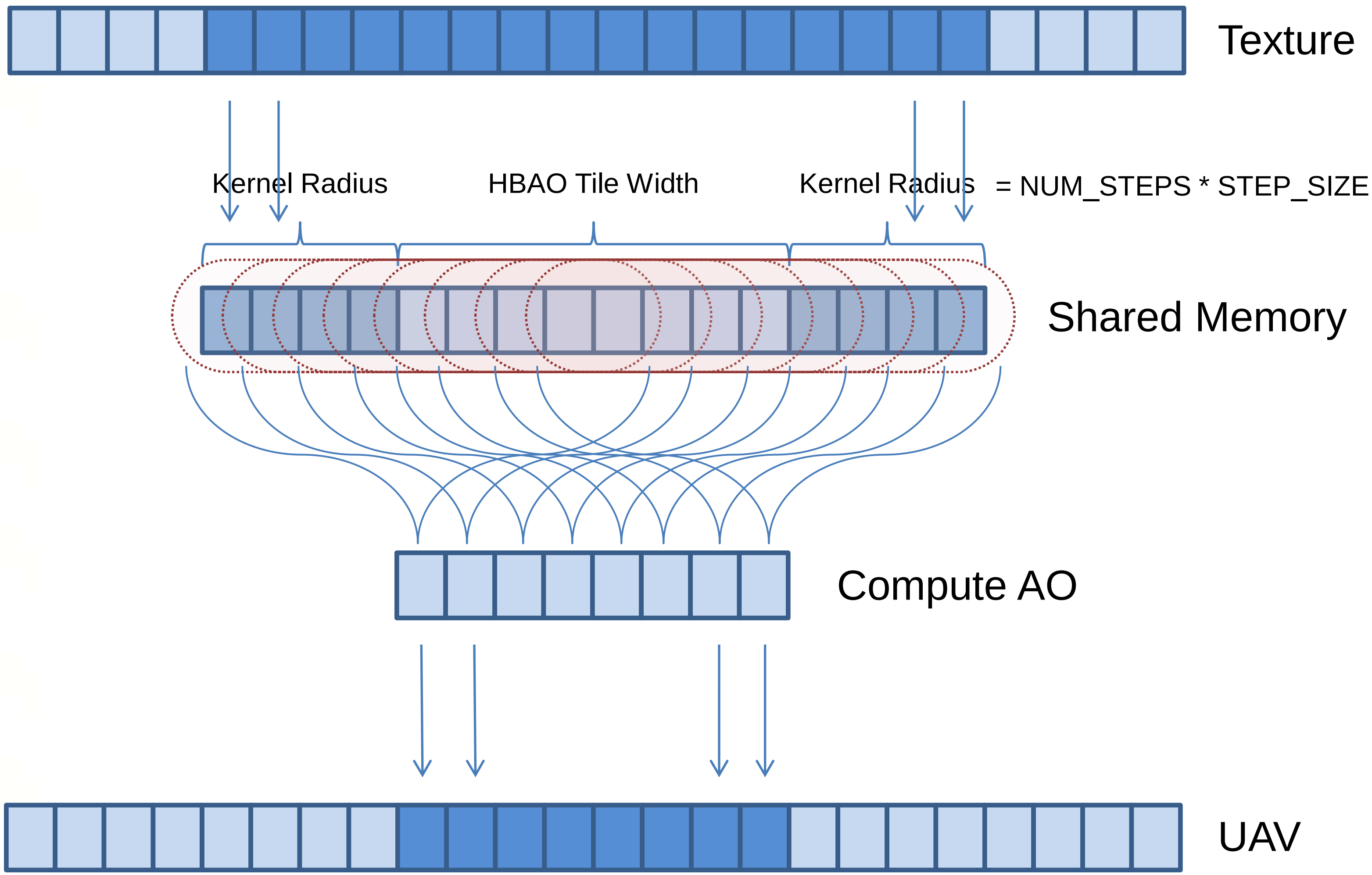
AO = sin h – sin t

Compute Shaders in HBAO

- Step 1, Compute HBAO along X , GroupSize=Tile_Width X 1
- Step 2, Compute HBAO along Y , GroupSize= 1 X Tile_Width



In each Group



Performance Comparison

Geforce GTX 480

1280x720	CS	PS	CS/PS
Full-res AO	426 fps	255 fps	1.67x
Half-res AO	593 fps	496 fps	1.20x

1600x900	CS	PS	CS/PS
Full-res AO	311 fps	168 fps	1.85x
Half-res AO	461 fps	368 fps	1.25x

SSAO using CS : Wrapping up

- Compute Shaders can be widely used for the post process effects
  - Summed Area Table
  - Scattered Bokeh
  - Blur, Dynamic Tone Mapping, etc.

Opacity Mapping

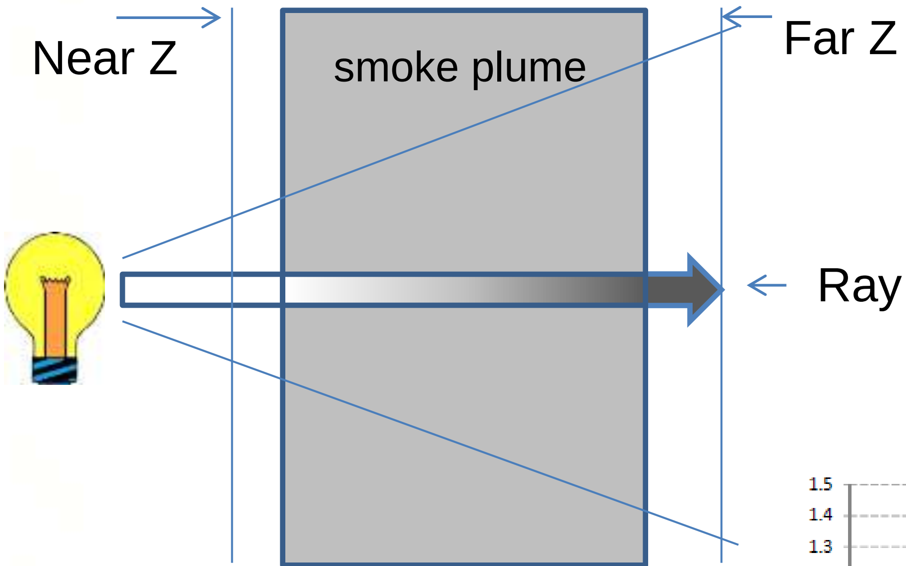
Features Covered

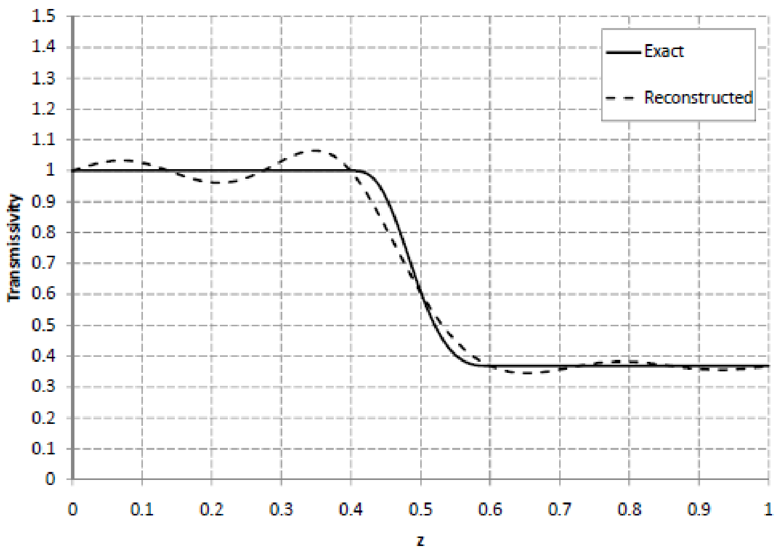
- Using tessellation to accelerate lighting effects
- Accelerating up-sampling with GatherRed()
- Playing nice with AA using SV_SampleIndex
- Read-only depth for soft particles

GOALS

- Plausible lighting for a game-typical particle system (16K largish translucent billboards)
- Self-shadowing (using opacity mapping)
  – Volumetric Smoke
  – Also receive shadows from opaque objects
- 3 light sources (all with shadows)

不透明度映射算法简介(1)

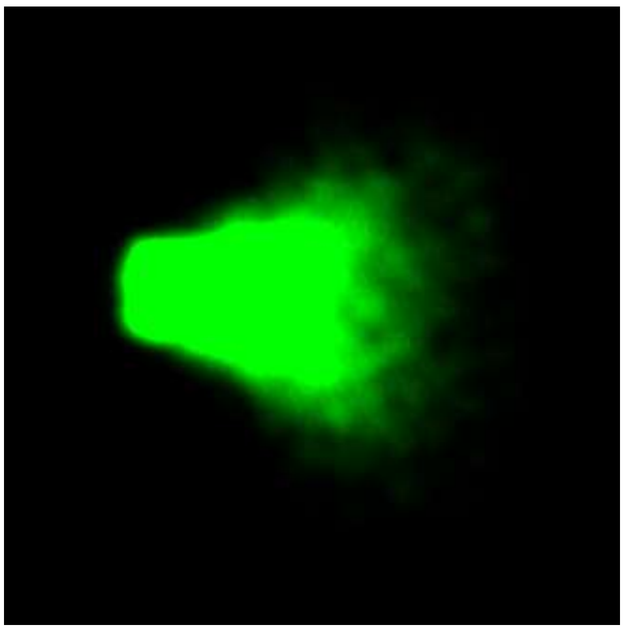


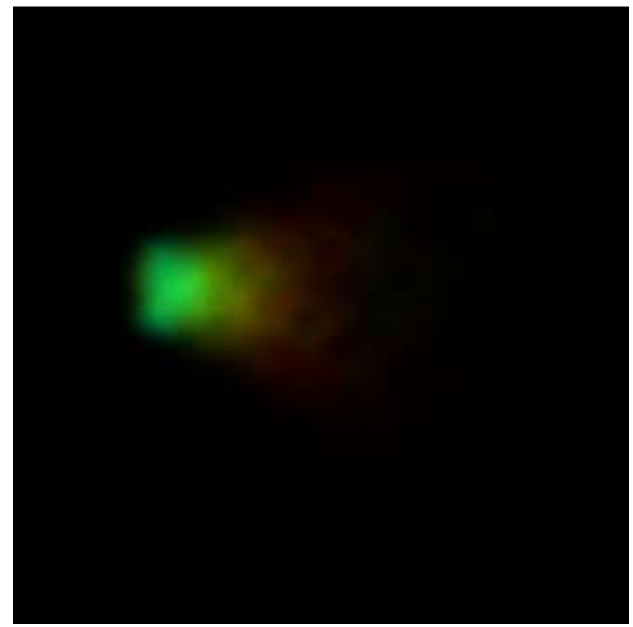


gameworks.nvidia.com

Opacity Mapping (2)

- Rendering the coefficients to generate the Fourier Opacity Map(FOM)





Opacity Mapping (3)

+ opacity mapping

=

Not performant with conventional methods

- Brute force (per-pixel lighting/shadowing) is not performant
  - 5 to 10 FPS on GTX 560 Ti* or HD 6950*
- Not surprising considering amount of overdraw...

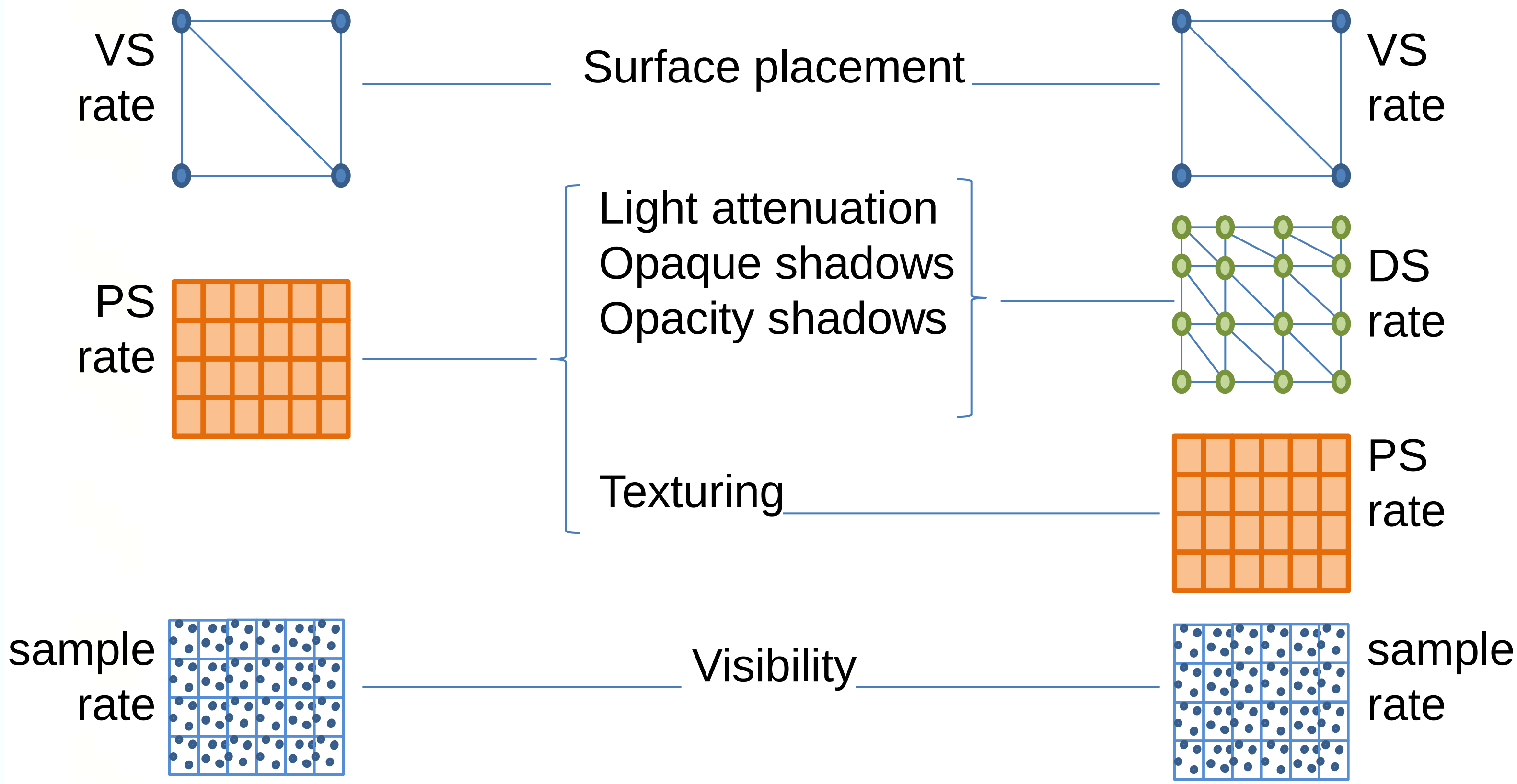
*1680x1050, 4xAA

Solution (Part 1)

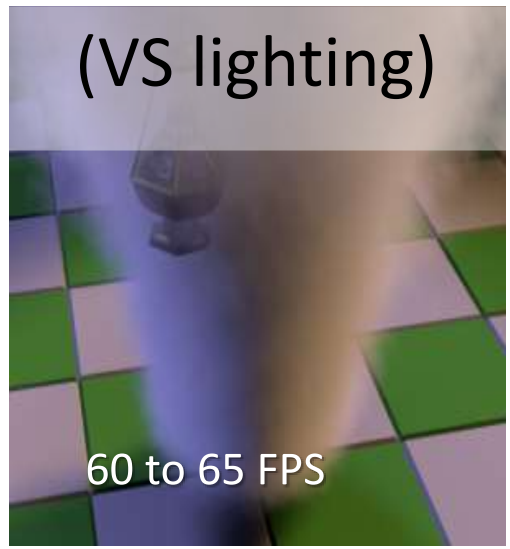
- Use DX11 tessellation to calculate lighting at an intermediate ‘sweet-spot’ rate in the DS
- High-frequency components can remain at per-pixel or per-sample rates, as required

Solution (Part 1)

- PS lighting
- DS lighting



Solution (Part 1)



Solution (Part 2)

- Main bottleneck is fill-rate following tess-opt
- So... render particles to low-res offscreen buffer*
  - significant benefit, even with tess opt (1.2x to 1.5x for GTX 560 Ti / HD 6950)
  - **BUT:** simple bilinear up-sampling from low-res can lead to artifacts at edges...

*[Cantlay, 2007]

Solution (Part 2)

- Ground truth (full res)

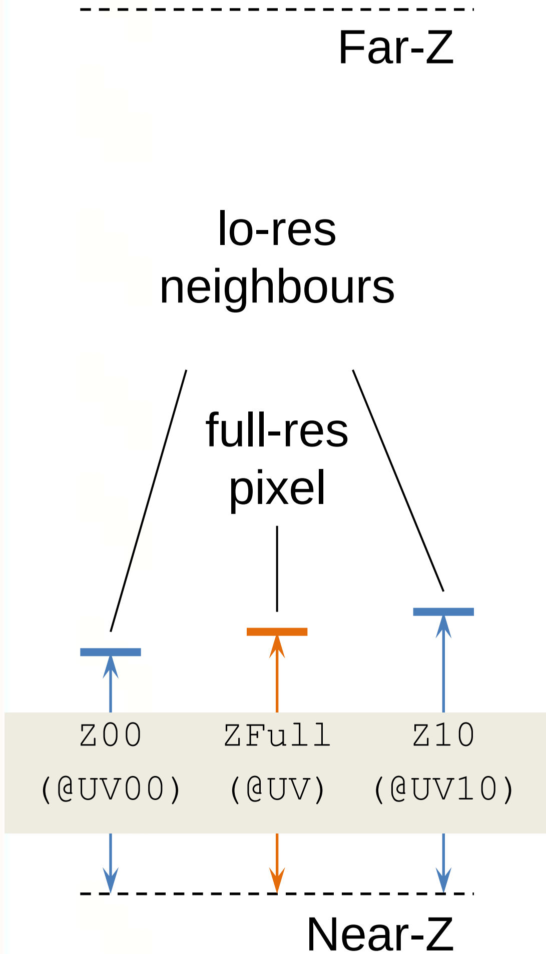
- Bilinear up-sample (half-res)

Solution (Part 2)

- Instead, we use nearest-depth up-sampling*
  - compares high-res depth with neighbouring low-res depths
  - samples from closest matching neighbour at depth discontinuities (bilinear otherwise)

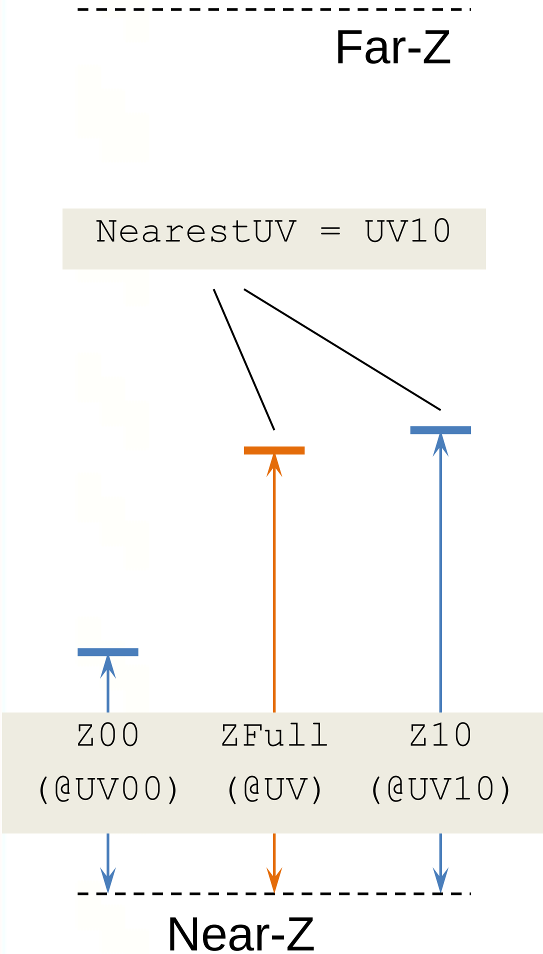
*[Bavoil, 2010]

Solution (Part 2)




```
if( abs(Z00-ZFull) < kDepthThreshold &&
    abs(Z10-ZFull) < kDepthThreshold &&
    abs(Z01-ZFull) < kDepthThreshold &&
    abs(Z11-ZFull) < kDepthThreshold )
{
   return loResColTex.Sample(sBilin,UV);
}
    else
{
   return loResColTex.Sample(sPoint,NearestUV);
}
```

Solution (Part 2)




```
if( abs(Z00-ZFull) < kDepthThreshold &&
    abs(Z10-ZFull) < kDepthThreshold &&
    abs(Z01-ZFull) < kDepthThreshold &&
    abs(Z11-ZFull) < kDepthThreshold )
{
   return loResColTex.Sample(sBilin,UV);
}
    else
{
   return loResColTex.Sample(sPoint,NearestUV);
}
```

Solution (Part 2)

- Use SM5 GatherRed() to efficiently fetch 2x2 low-res depth neighbourhood in one go

```
float4 zg = g_DepthTex.GatherRed(g_Sampler,UV);
float z00 = zg.w;    // w: floor(uv)
float z10 = zg.z;    // z: ceil(u),floor(v)
float z01 = zg.x;    // x: floor(u),ceil(v)
float z11 = zg.y;    // y: ceil(uv)
```

Solution (Part 2)

- Nearest-depth up-sampling plays nice with AA when run per-sample
  - and surprisingly performant! (FPS hit < 5%)

```
float4 UpsamplePS(  VS_OUTPUT In,
                    uint uSID : SV_SampleIndex
                    ) : SV_Target
```

Solution (Part 2)

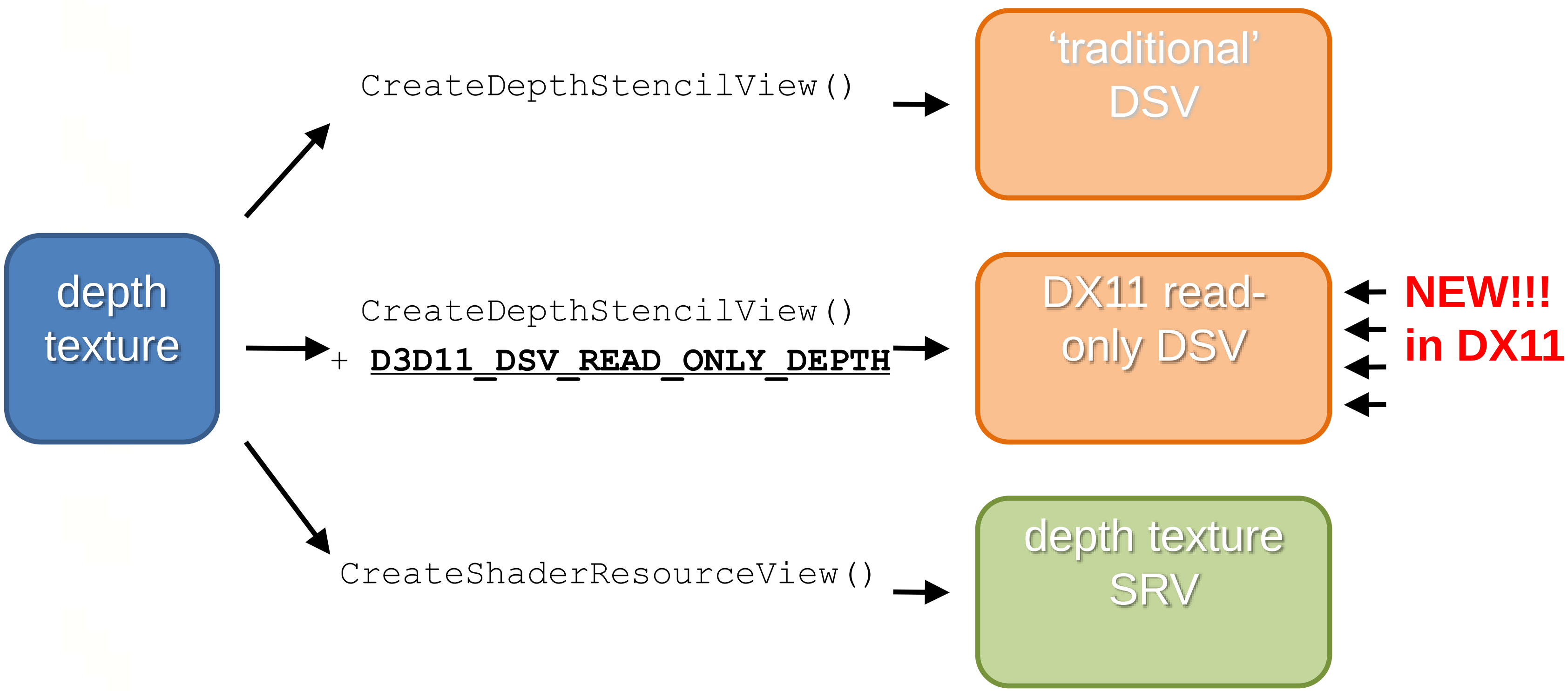
- Ground truth (full res)

- Nearest-depth up-sample

Soft Particles

- Soft particles (depth-based alpha fade) requires read from scene depth

DX11 solution:



Soft Particles

STEP 1: render opaque objects to depth texture

‘traditional’ DSV

DX11 read-only DSV

depth texture SRV

```
pDC->OMSetRenderTargets(...)

// Render opaque objects
```

Soft Particles

STEP 2: render soft particles with depth-test

‘traditional’ DSV

DX11 read-only DSV

depth texture SRV

```
pDC->OMSetRenderTargets(...)
pDC->PSSetShaderResources(...)
// (Valid D3D state!)

// Render soft particles
```

Soft Particles

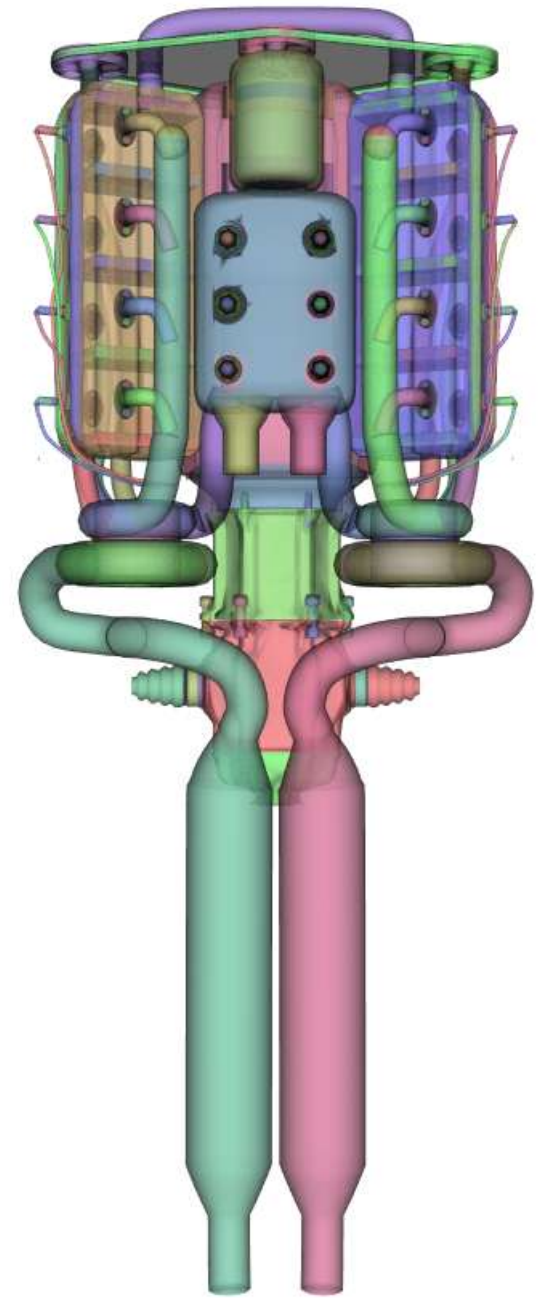
- 'Hard' particles
- Soft particles

Final image

Opacity Mapping : Wrapping up

- 5x to 10x overall speedup
- DX11 tessellation gave us most of it
- But rendering at reduced-res alleviates fill-rate and lets tessellation shine thru
- GatherRed() and RO DSV also saved cycles

Stochastic Transparency



Features covered

- Use the output coverage mask feature to implement Order-Independent Transparency (OIT) rendering

Basic Idea

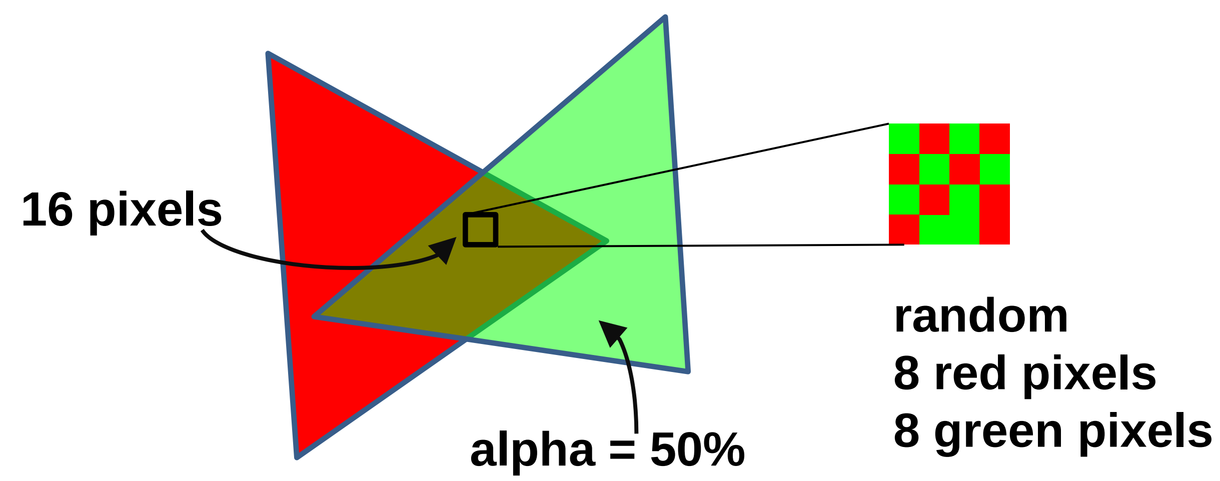
- Screen door transparency is simple & fast, but lots of noise

A green triangle over a red triangle → random 50% green pixels and 50% red pixels

SDT is OIT !

- How about: Split each pixel into many sub-pixels (samples)

Half of sub-pixels are red. The other half of sub-pixels are green
Then we can average these sub-pixels to get blended color!

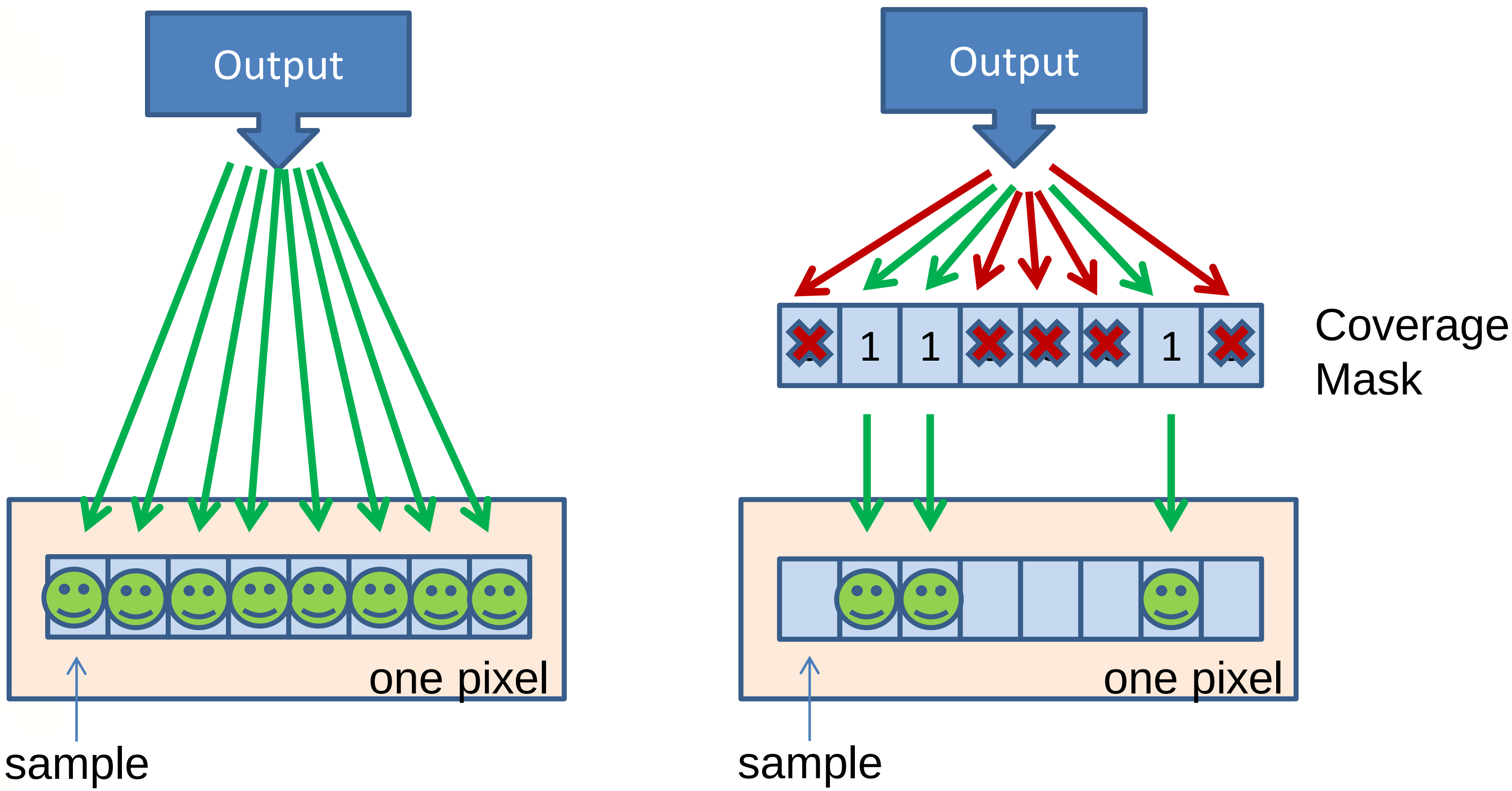


Implementation

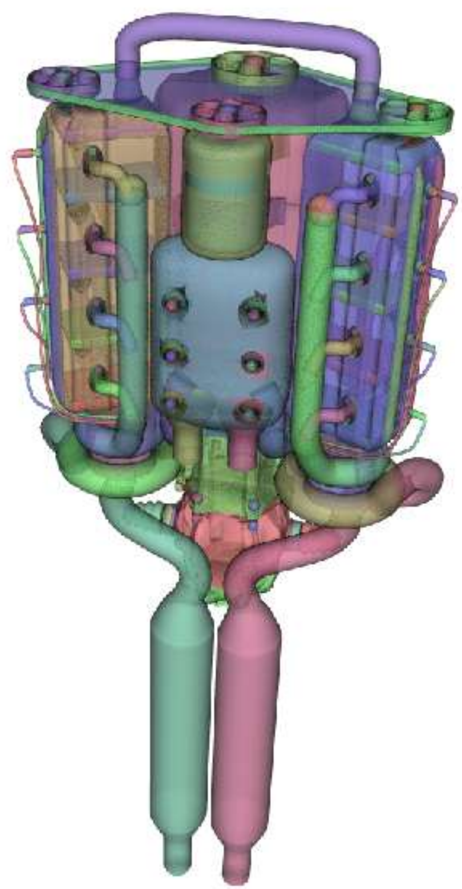
- How to split a pixel then?
  - Create an MSAA buffer, not for AA though
  - 8x MSAA, each pixel can have 8 samples

How SV_Coverage works

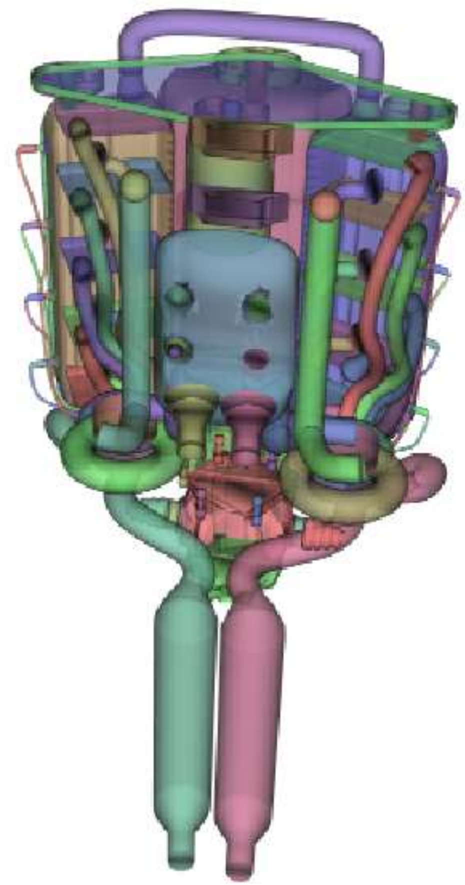
- Without Coverage Mask
- With Coverage Mask



Quality comparison



Stochastic Transparency



Without Coverage Mask

Stochastic Transparency : Wrapping up

- The conventional Order-dependent Transparency rendering can't deal with the complex transparent objects

- Stochastic Transparency can render the complex transparent objects correctly, and the algorithm's relatively simple

FXAA

Without FXAA

FXAA

Advantages

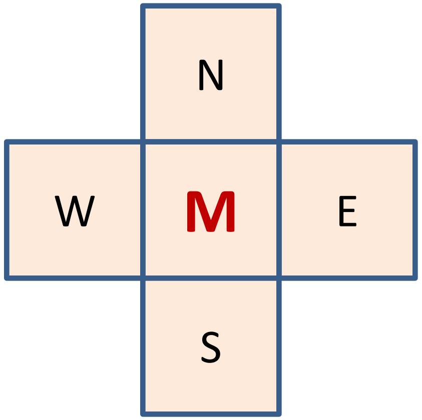
- easy to integrate into a single pixel shader. FXAA runs as a single-pass filter on a single-sample color image.
- easy to integrate into any engines
- provide a memory advantage over MSAA and avoid artifacts caused by MSAA
- a high performance and high quality screen-space software approximation to anti-aliasing, can balance the performance and quality by selecting different preset

Algorithm Overview

- Check local contrast to avoid processing non-edges.
- Classify the direction of the edge (horizontal or vertical)
- Search for end-of-edge
- Given the ends of the edge and the pixel position on the edge, get a sub-pixel offset, and re-sample the input texture with this offset.

Edge detection

- use the pixel and its North, South, East, and West neighbors
- Figure out the local max and min luma
- If the difference in local max and min luma (contrast) is lower than a threshold then it's on the edge



Vertical/Horizontal Edge Test

- Use the following 3X3 high-pass filter kernel

NW	N	NE
W	**M**	E
SW	S	SE

edgeVert =

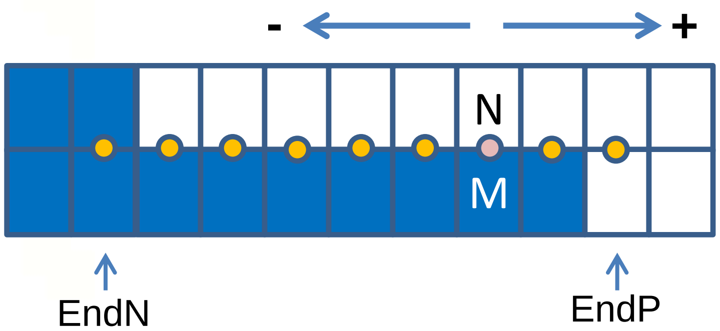
0.25	-0.5	0.25
0.5	-1.0	0.5
0.25	-0.5	0.25

edgeHorz =

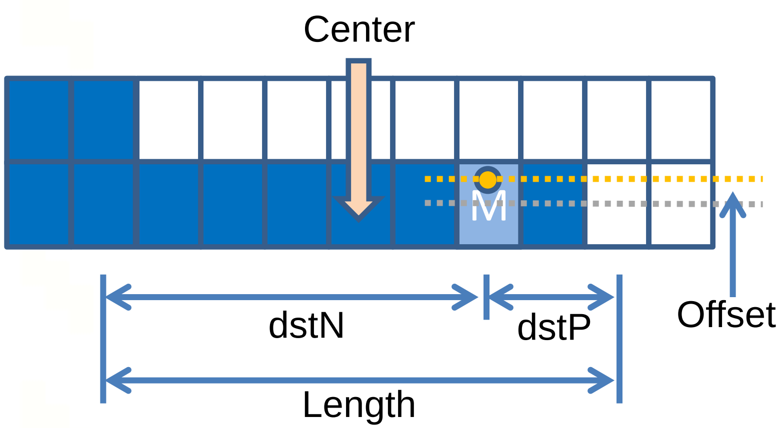
0.25	0.5	0.25
-0.5	-1.0	-0.5
0.25	0.5	0.25

```
bool horzSpan = edgeHorz >= edgeVert;
```

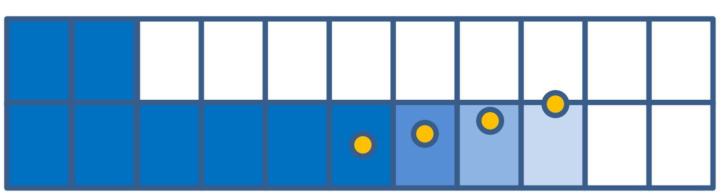
End-of-edge Search



- Search for end-of-edge in both the negative and positive, directions along the edge. Checking for a significant change in average luminance of the high contrast pixel pair along the edge .



- Position on span is used to compute sub-pixel filter offset using simple ramp, the maximum offset is 0.5



- The result of applying FXAA on the pixels along the edge

FXAA : Wrapping up

- FXAA has been used by many games
  - Crysis2
  - Age of Conan
  - Duke Nukem Forever
  - FEAR3
- Easy to port to DX9 & OpenGL

Summary

- DirectCompute introduces more powerful ability on image processing
- Some trivial new features can also give significant improvement on rendering

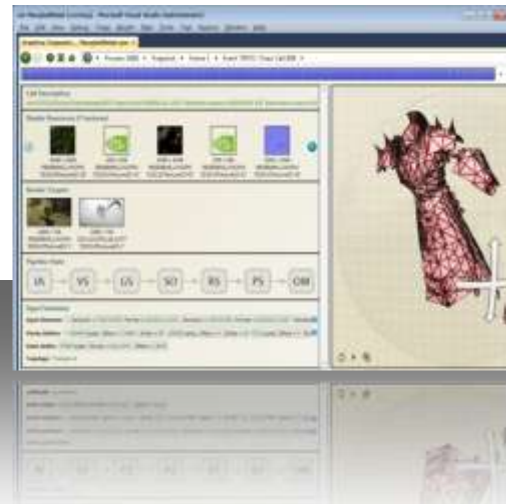
NVIDIA Parallel Nsight for Graphics

Graphics Debugger

Debug HLSL graphics shaders directly on GPU hardware inside Visual Studio

Examine shaders executing in parallel

Identify problem areas with conditional breakpoints

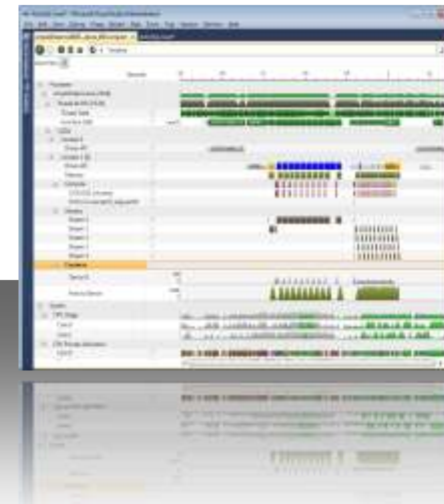


Graphics Inspector

Real-time inspection of DirectX API calls

Investigate GPU pipeline state

See contributing fragments with Pixel History



System Analysis

View CPU and GPU events on a single timeline

Examine workload dependencies

DirectX3D and OpenGL API Trace

gameworks.nvidia.com

Parallel Nsight Update

For Game and Graphics Development

1.51

Version 2.0

Beyond

- View all graphics resources at a glance
- Numerous usability and workflow improvements
- Graphics profiler performance and accuracy
- Driver independence
- Stability improvements
- Support for r270 driver and latest hardware

Available Q2 2011 http://parallelnsight.nvidia.com/

Parallel Nsight Update

For Compute and Parallel Development

1.51

Version 2.0

Beyond

- CUDA Toolkit 4.0 Support
- Full Visual Studio 2010 Platform Support
- Tesla Compute Cluster (TCC) Analysis
- PTX Assembly Debugging
- CUDA Attach to Process
- CUDA Derived Metrics and Experiments
- CUDA Concurrent Kernel Trace
- CUDA Runtime API Trace
- Advanced Conditional Breakpoints
- Support for r270 driver and latest hardware

Available Q2 2011 http://parallelnsight.nvidia.com/

Questions?

youngy@nvidia.com

References

- DX11 Performance Gems, Jon Jansen, GDC 2011
- High Performance Post-Processing, Nathan Hoobler, GDC 2011
- Fourier Opacity Mapping, Jansen & Bavoil, 2010
- Image-Space Horizon-Based Ambient Occlusion, Bryan Dudash, 2008