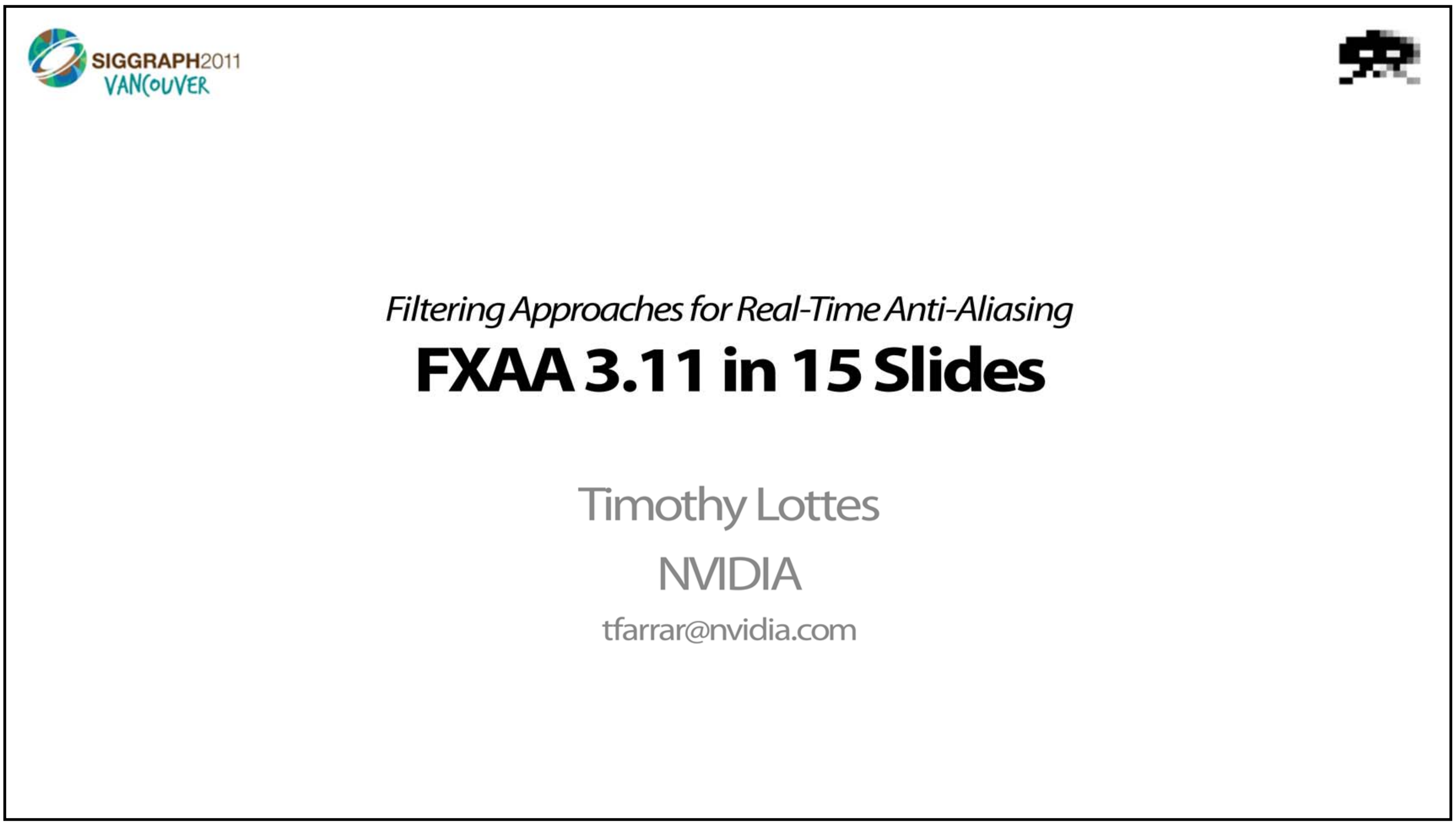
SIGGRAPH2011 VANCOUVER

Filtering Approaches for Real-Time Anti-Aliasing

http://www.iryoku.com/aacourse/



SIGGRAPH2011 VANCOUVER

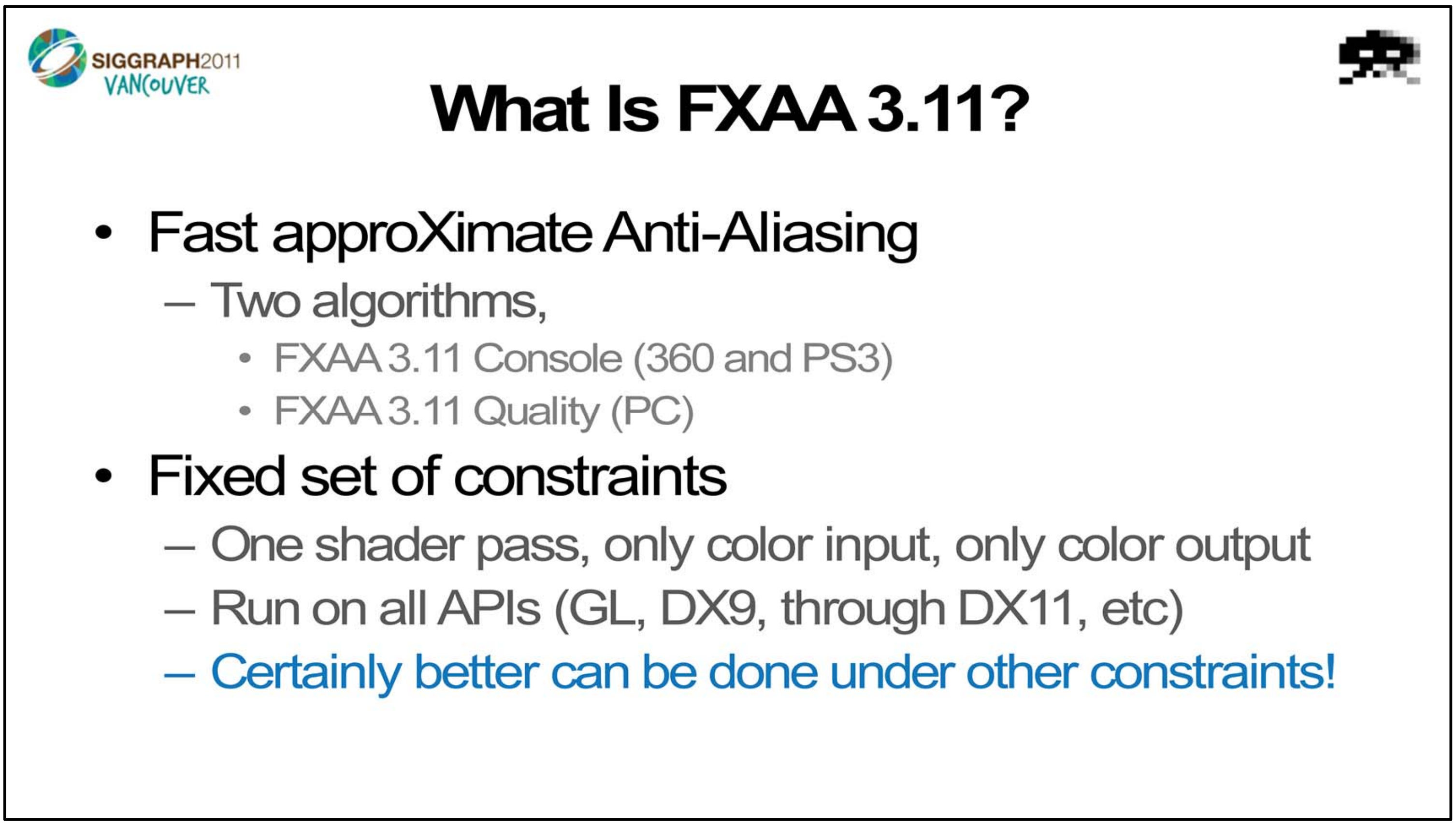
Filtering Approaches for Real-Time Anti-Aliasing

FXAA 3.11 in 15 Slides

Timothy Lottes

NVIDIA

tfarrar@nvidia.com



SIGGRAPH2011 VANCOUVER

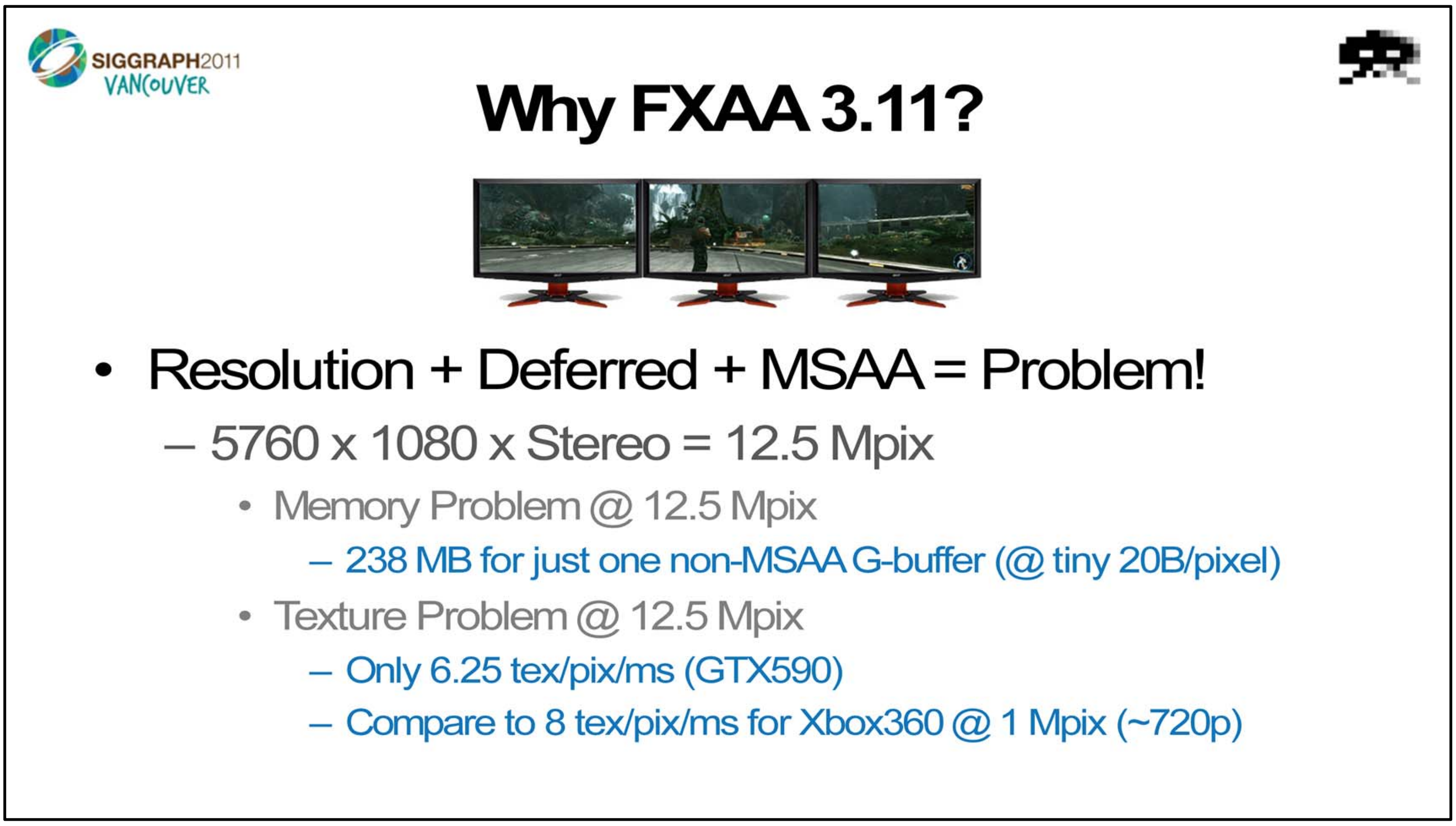
What Is FXAA 3.11?

- Fast approXimate Anti-Aliasing
  - Two algorithms,
    - FXAA 3.11 Console (360 and PS3)
    - FXAA 3.11 Quality (PC)
- Fixed set of constraints
  - One shader pass, only color input, only color output
  - Run on all APIs (GL, DX9, through DX11, etc)
  - Certainly better can be done under other constraints!

Note “approXimate”!

FXAA does not attempt anything close to the correct solution.

Rather attempts something fast that looks visually good enough.



SIGGRAPH2011 VANCOUVER

Why FXAA 3.11?

- Resolution + Deferred + MSAA = Problem!
  - 5760 x 1080 x Stereo = 12.5 Mpix
    - Memory Problem @ 12.5 Mpix
      - 238 MB for just one non-MSAA G-buffer (@ tiny 20B/pixel)
    - Texture Problem @ 12.5 Mpix
      - Only 6.25 tex/pix/ms (GTX590)
      - Compare to 8 tex/pix/ms for Xbox360 @ 1 Mpix (~720p)

(1.) MEMORY PROBLEM

At huge resolutions with deferred rendering, memory used for render targets and back buffers can be very large without MSAA.

Even 4xMSAA might not be practical (for example with larger FP16 precision G-buffers).

Software post-process filtering AA relieves this memory pressure.

(2.) TEXTURE PERFORMANCE PROBLEM

PC GPU texture performance/pixel for huge resolutions can be under what is common on the 5-year-old consoles.

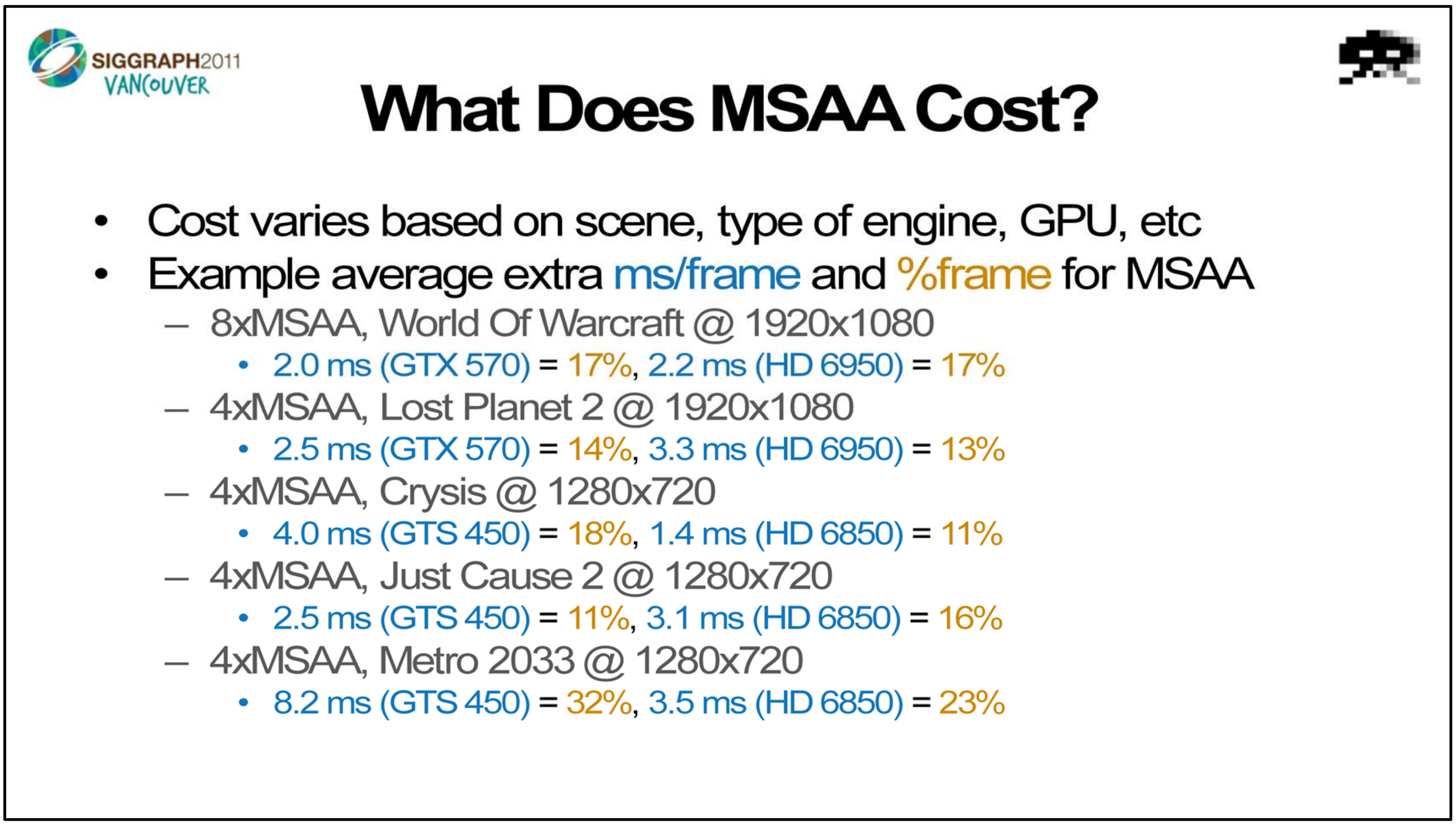
Need something faster than what is required performance-wise for consoles.

(3.) RESOLUTION SET TO RISE MORE

Could see another huge bump in resolution when iPhone4 pixels/inch levels reach desktop displays.

Estimating unlike mobile phones, desktop has not seen the end of the resolution race.

Even mobile phones might not have seen the end of the resolution race.



SIGGRAPH2011 VANCOUVER

What Does MSAA Cost?

- Cost varies based on scene, type of engine, GPU, etc
- Example average extra ms/frame and %frame for MSAA
  - 8xMSAA, World Of Warcraft @ 1920x1080
    - 2.0 ms (GTX 570) = 17%, 2.2 ms (HD 6950) = 17%
  - 4xMSAA, Lost Planet 2 @ 1920x1080
    - 2.5 ms (GTX 570) = 14%, 3.3 ms (HD 6950) = 13%
  - 4xMSAA, Crysis @ 1280x720
    - 4.0 ms (GTS 450) = 18%, 1.4 ms (HD 6850) = 11%
  - 4xMSAA, Just Cause 2 @ 1280x720
    - 2.5 ms (GTS 450) = 11%, 3.1 ms (HD 6850) = 16%
  - 4xMSAA, Metro 2033 @ 1280x720
    - 8.2 ms (GTS 450) = 32%, 3.5 ms (HD 6850) = 23%

How much does MSAA actually cost?

These examples cover multiple types of graphics engines.

Numbers taken from taking difference of AA vs MSAA numbers from various www.tomshardware.com reviews,

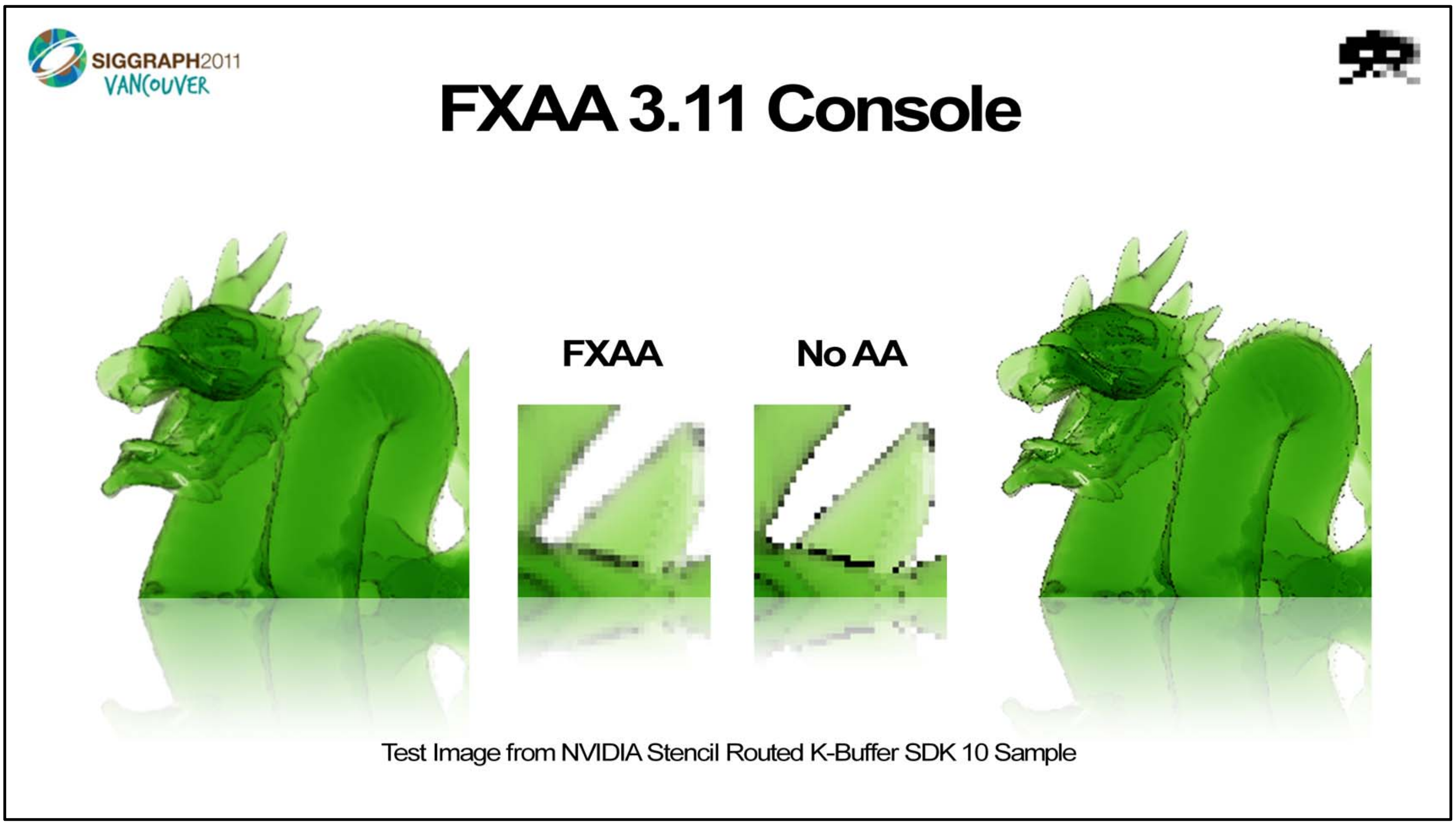
http://www.tomshardware.com/reviews/nvidia-geforce-gtx-560-ti-gf114,2845-12.html

http://www.tomshardware.com/reviews/nvidia-geforce-gtx-560-ti-gf114,2845-7.html

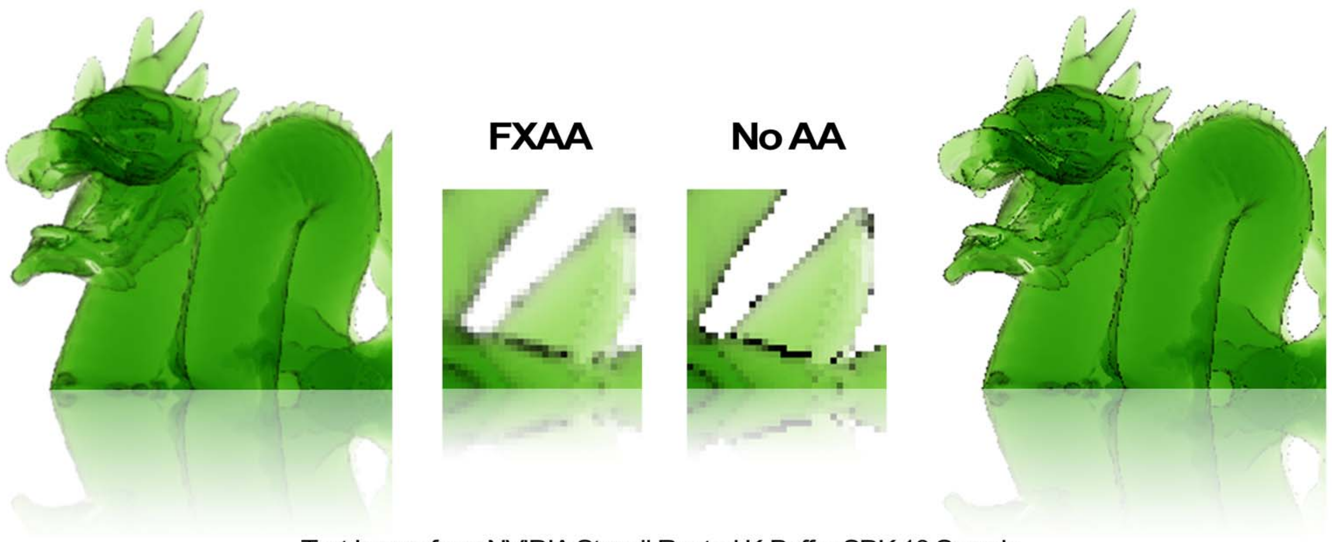
http://www.tomshardware.com/reviews/af6850-1024d5s1-ngt440-1gqi-f1-n450gts-m2d1gd5,2949-6.html

http://www.tomshardware.com/reviews/af6850-1024d5s1-ngt440-1gqi-f1-n450gts-m2d1gd5,2949-8.html

http://www.tomshardware.com/reviews/af6850-1024d5s1-ngt440-1gqi-f1-n450gts-m2d1gd5,2949-9.html



FXAA 3.11 Console



Test Image from NVIDIA Stencil Routed K-Buffer SDK 10 Sample

FXAA Console is a Local Contrast Adaptive Directional Edge Blur

(In short) 2|4-tap variable-length bi-directional filter

(Advantages) Very fast, reduces contrast on pixel and sub-pixel aliasing

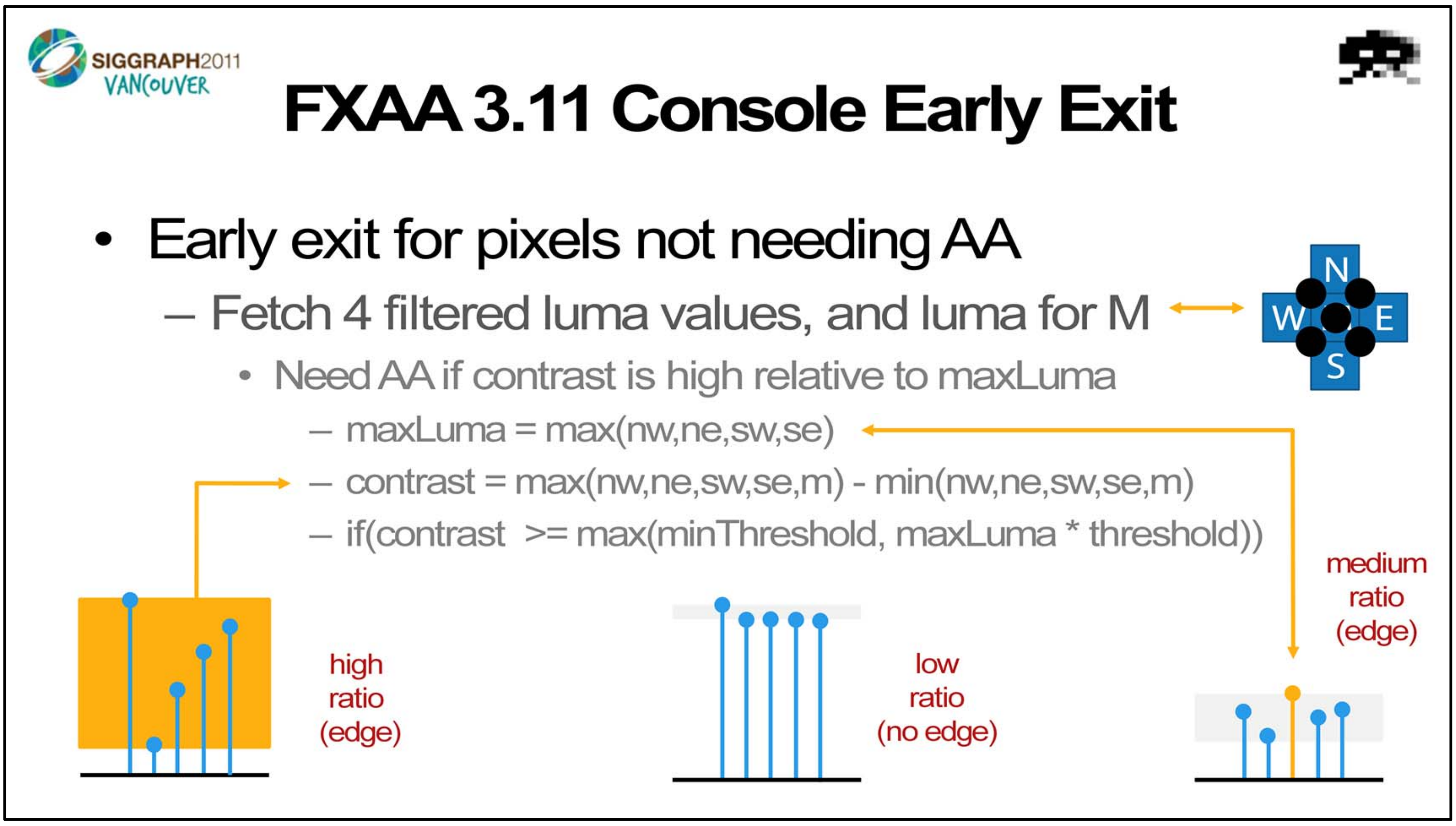
(Disadvantages) Not very good on near horizontal or vertical edges

```
/*============================================================================

                         FXAA3 CONSOLE - PC VERSION

------------------------------------------------------------------------------
Instead of using this on PC, I'd suggest just using FXAA Quality with
    #define FXAA_QUALITY__PRESET 10
Or
    #define FXAA_QUALITY__PRESET 20
Either are higher qualilty and almost as fast as this on modern PC GPUs.
============================================================================*/
#if (FXAA_PC_CONSOLE == 1)
/*--------------------------------------------------------------------------*/
FxaaFloat4 FxaaPixelShader(
    // See FXAA Quality FxaaPixelShader() source for docs on Inputs!
    FxaaFloat2 pos,
    FxaaFloat4 fxaaConsolePosPos,
    FxaaTex tex,
    FxaaTex fxaaConsole360TexExpBiasNegOne,
    FxaaTex fxaaConsole360TexExpBiasNegTwo,
    FxaaFloat2 fxaaQualityRcpFrame,
    FxaaFloat4 fxaaConsoleRcpFrameOpt,
    FxaaFloat4 fxaaConsoleRcpFrameOpt2,
    FxaaFloat4 fxaaConsole360RcpFrameOpt2,
    FxaaFloat fxaaQualitySubpix,
    FxaaFloat fxaaQualityEdgeThreshold,
    FxaaFloat fxaaQualityEdgeThresholdMin,
    FxaaFloat fxaaConsoleEdgeSharpness,
    FxaaFloat fxaaConsoleEdgeThreshold,
    FxaaFloat fxaaConsoleEdgeThresholdMin,
    FxaaFloat4 fxaaConsole360ConstDir
) {
/*--------------------------------------------------------------------------*/
    FxaaFloat lumaNw = FxaaLuma(FxaaTexTop(tex, fxaaConsolePosPos.xy));
    FxaaFloat lumaSw = FxaaLuma(FxaaTexTop(tex, fxaaConsolePosPos.xw));
    FxaaFloat lumaNe = FxaaLuma(FxaaTexTop(tex, fxaaConsolePosPos.zy));
    FxaaFloat lumaSe = FxaaLuma(FxaaTexTop(tex, fxaaConsolePosPos.zw));
/*--------------------------------------------------------------------------*/
    FxaaFloat4 rgbyM = FxaaTexTop(tex, pos.xy);
    #if (FXAA_GREEN_AS_LUMA == 0)
        FxaaFloat lumaM = rgbyM.w;
    #else
        FxaaFloat lumaM = rgbyM.y;
    #endif
/*--------------------------------------------------------------------------*/
    FxaaFloat lumaMaxNwSw = max(lumaNw, lumaSw);
    lumaNe += 1.0/384.0;
    FxaaFloat lumaMinNwSw = min(lumaNw, lumaSw);
/*--------------------------------------------------------------------------*/
    FxaaFloat lumaMaxNeSe = max(lumaNe, lumaSe);
    FxaaFloat lumaMinNeSe = min(lumaNe, lumaSe);
/*--------------------------------------------------------------------------*/
    FxaaFloat lumaMax = max(lumaMaxNeSe, lumaMaxNwSw);
    FxaaFloat lumaMin = min(lumaMinNeSe, lumaMinNwSw);
/*--------------------------------------------------------------------------*/
    FxaaFloat lumaMaxScaled = lumaMax * fxaaConsoleEdgeThreshold;
/*--------------------------------------------------------------------------*/
    FxaaFloat lumaMinM = min(lumaMin, lumaM);
    FxaaFloat lumaMaxScaledClamped = max(fxaaConsoleEdgeThresholdMin, lumaMaxScaled);
    FxaaFloat lumaMaxM = max(lumaMax, lumaM);
    FxaaFloat dirSwMinusNe = lumaSw - lumaNe;
    FxaaFloat lumaMaxSubMinM = lumaMaxM - lumaMinM;
    FxaaFloat dirSeMinusNw = lumaSe - lumaNw;
    if(lumaMaxSubMinM < lumaMaxScaledClamped) return rgbyM;
/*--------------------------------------------------------------------------*/
    FxaaFloat2 dir;
    dir.x = dirSwMinusNe + dirSeMinusNw;
    dir.y = dirSwMinusNe - dirSeMinusNw;
/*--------------------------------------------------------------------------*/
    FxaaFloat2 dir1 = normalize(dir.xy);
    FxaaFloat4 rgbyN1 = FxaaTexTop(tex, pos.xy - dir1 * fxaaConsoleRcpFrameOpt.zw);
    FxaaFloat4 rgbyP1 = FxaaTexTop(tex, pos.xy + dir1 * fxaaConsoleRcpFrameOpt.zw);
/*--------------------------------------------------------------------------*/
```

```
    FxaaFloat dirAbsMinTimesC = min(abs(dir1.x), abs(dir1.y)) * fxaaConsoleEdgeSharpness;
    FxaaFloat2 dir2 = clamp(dir1.xy / dirAbsMinTimesC, -2.0, 2.0);
/*--------------------------------------------------------------------------*/
    FxaaFloat4 rgbyN2 = FxaaTexTop(tex, pos.xy - dir2 * fxaaConsoleRcpFrameOpt2.zw);
    FxaaFloat4 rgbyP2 = FxaaTexTop(tex, pos.xy + dir2 * fxaaConsoleRcpFrameOpt2.zw);
/*--------------------------------------------------------------------------*/
    FxaaFloat4 rgbyA = rgbyN1 + rgbyP1;
    FxaaFloat4 rgbyB = ((rgbyN2 + rgbyP2) * 0.25) + (rgbyA * 0.25);
/*--------------------------------------------------------------------------*/
    #if (FXAA_GREEN_AS_LUMA == 0)
        FxaaBool twoTap = (rgbyB.w < lumaMin) || (rgbyB.w > lumaMax);
    #else
        FxaaBool twoTap = (rgbyB.y < lumaMin) || (rgbyB.y > lumaMax);
    #endif
    if(twoTap) rgbyB.xyz = rgbyA.xyz * 0.5;
    return rgbyB; }
/*==========================================================================*/
#endif
```



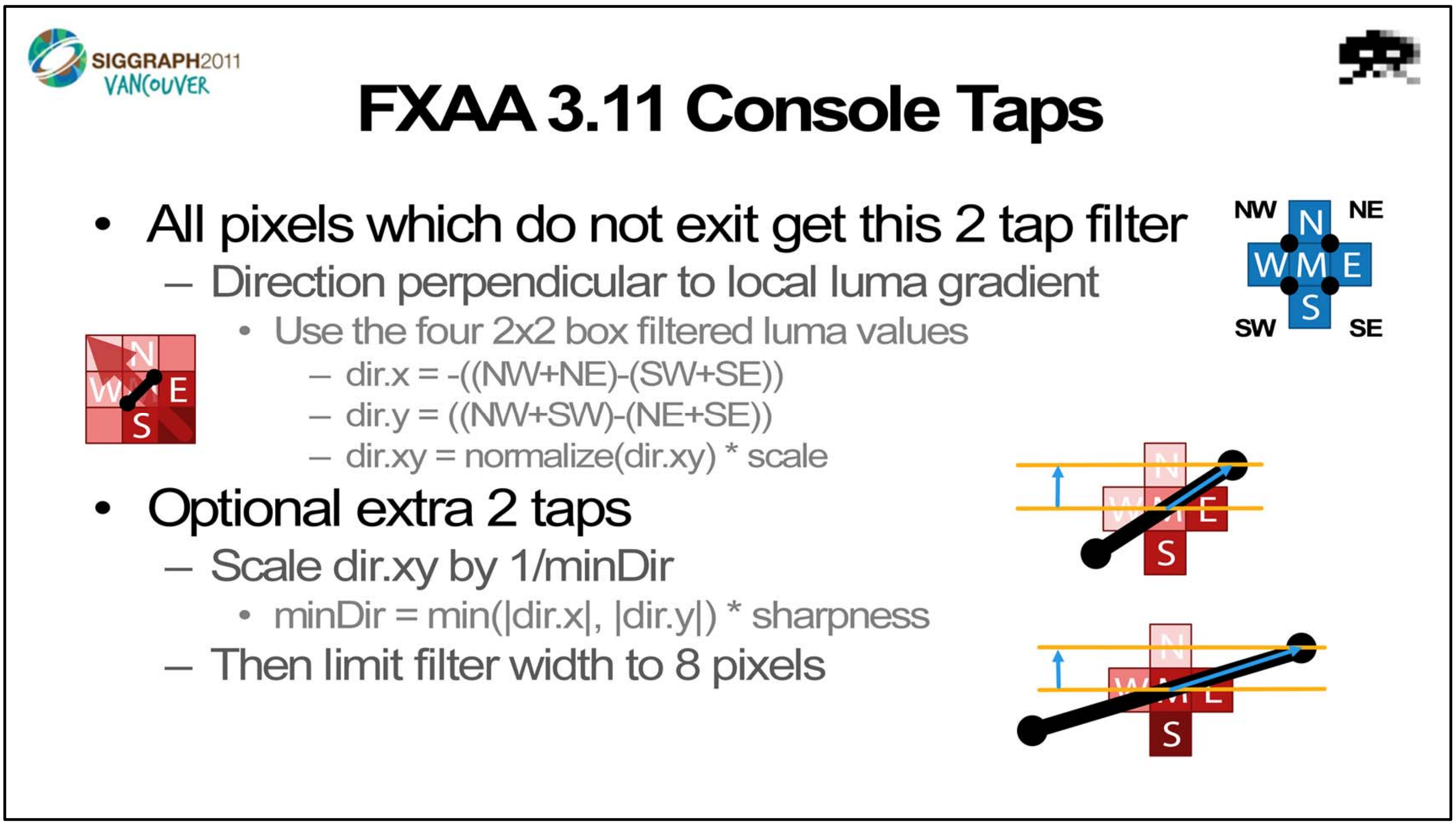
FXAA 3.11 Console Early Exit

- Early exit for pixels not needing AA
  - Fetch 4 filtered luma values, and luma for M
    - Need AA if contrast is high relative to maxLuma
      - maxLuma = max(nw,ne,sw,se)
      - contrast = max(nw,ne,sw,se,m) - min(nw,ne,sw,se,m)
      - if(contrast >= max(minThreshold, maxLuma * threshold))

On PS3 there is no early exit.

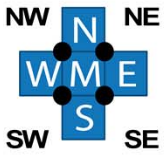
The “minThreshold” factor helps early exit more in the darks.

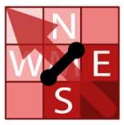
Luma is pre-packed in A or FXAA_GREEN_AS_LUMA is used to use G instead of A.

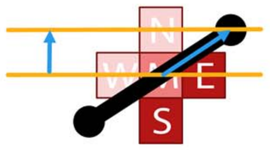


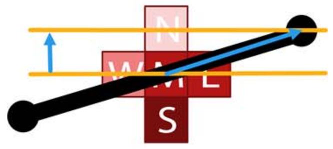
FXAA 3.11 Console Taps

- All pixels which do not exit get this 2 tap filter
  - Direction perpendicular to local luma gradient
    - Use the four 2x2 box filtered luma values
      - dir.x = -((NW+NE)-(SW+SE))
      - dir.y = ((NW+SW)-(NE+SE))
      - dir.xy = normalize(dir.xy) * scale
- Optional extra 2 taps
  - Scale dir.xy by 1/minDir
    - minDir = min(|dir.x|, |dir.y|) * sharpness
  - Then limit filter width to 8 pixels



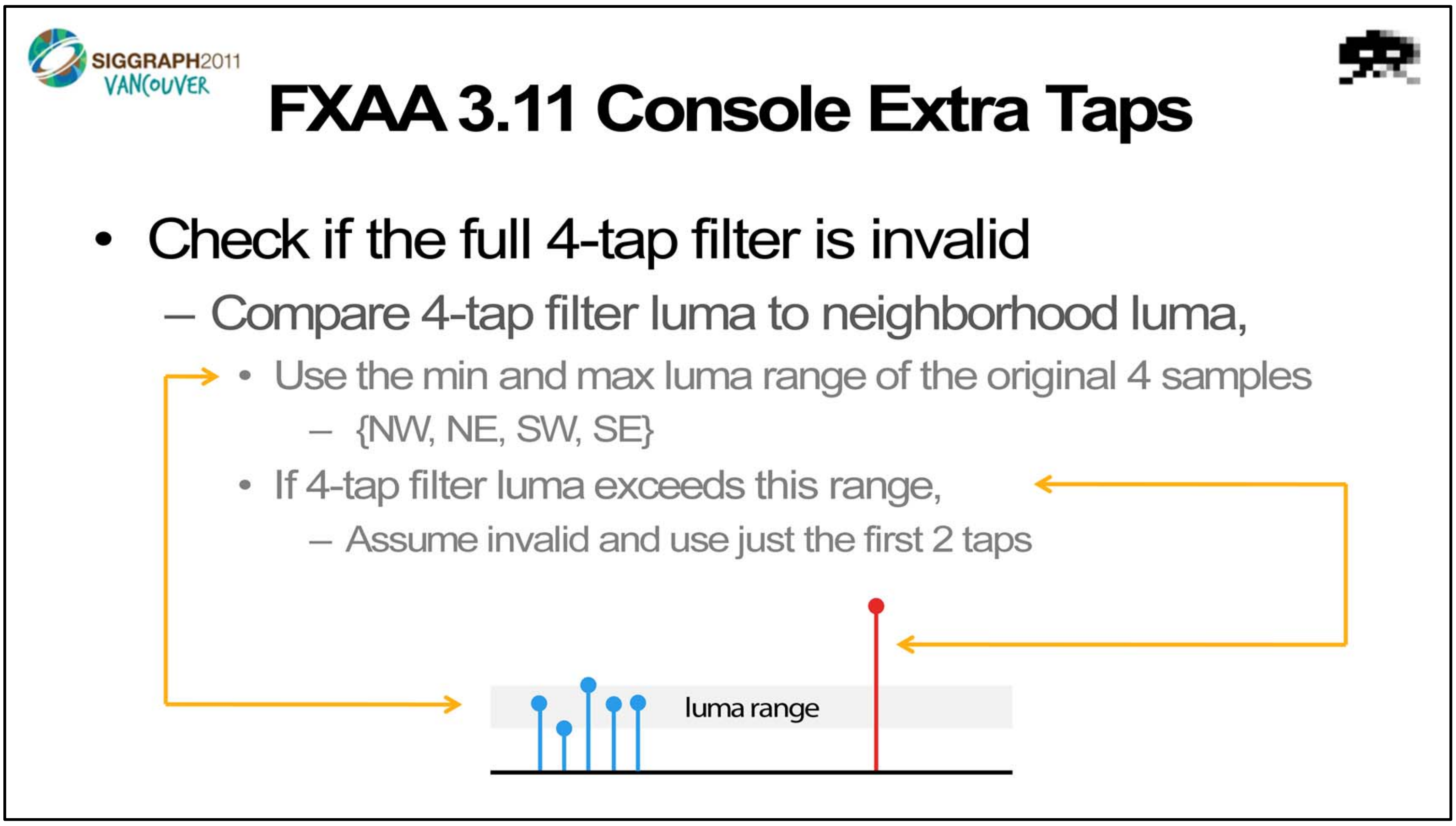






The “scale” factor controls the amount of blur.
This 2-tap filter handles the near diagonal aliasing, and sub-pixel aliasing.

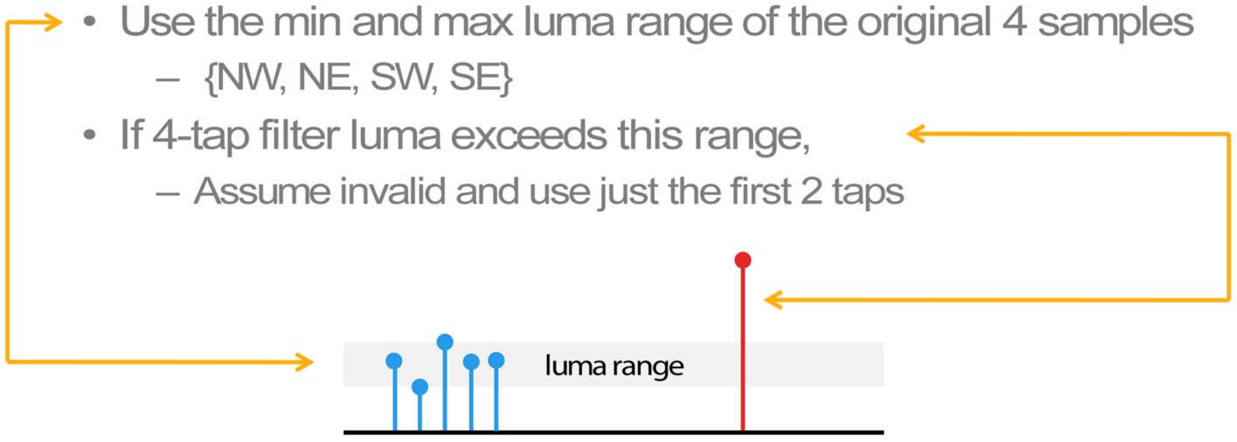
There is a hidden “NE += 1.0/384.0” factor which insures a fixed blur direction in the case of a single hot pixel, which otherwise would have zero gradient.



SIGGRAPH2011 VANCOUVER

FXAA 3.11 Console Extra Taps

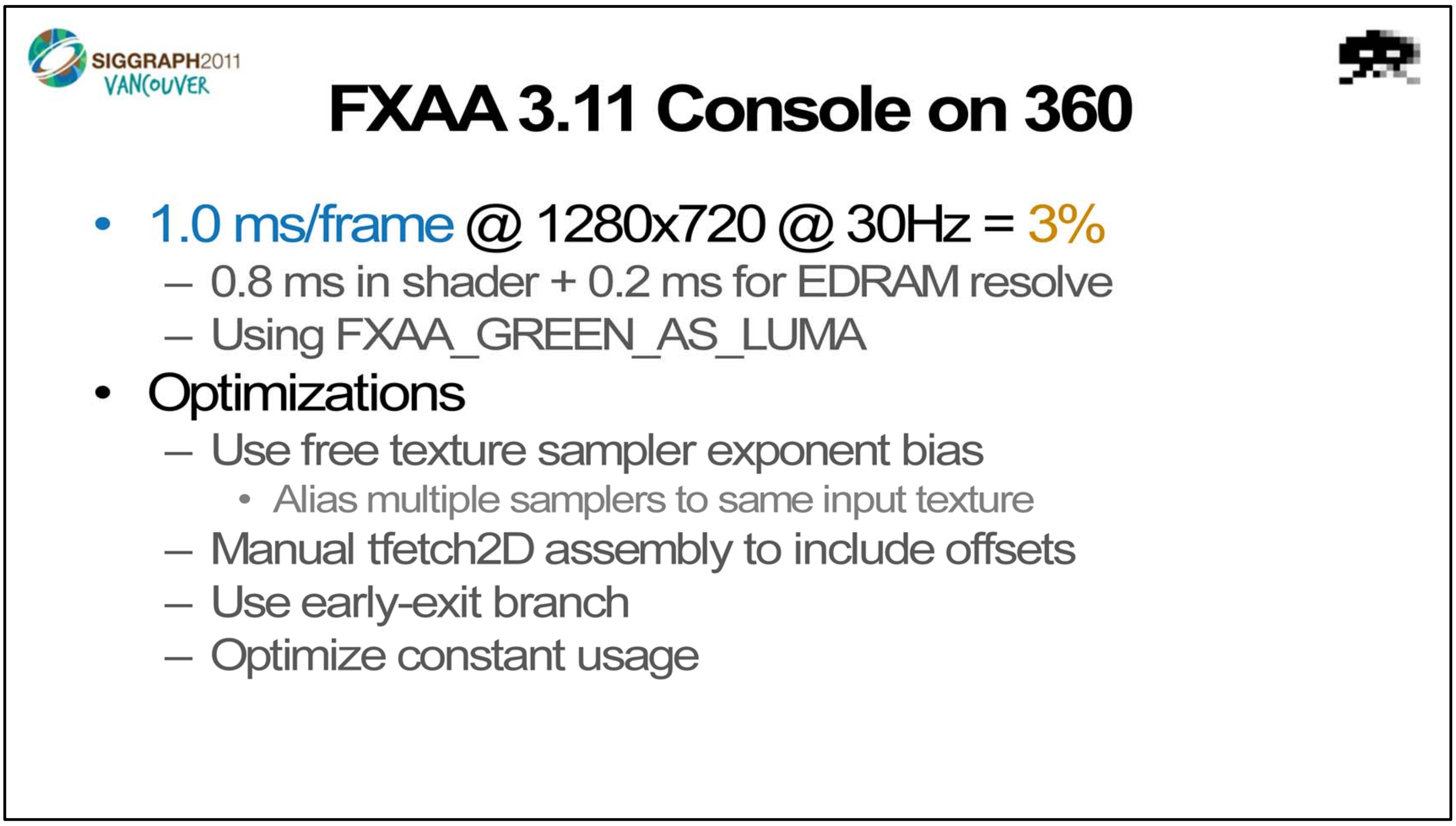
- Check if the full 4-tap filter is invalid
  - Compare 4-tap filter luma to neighborhood luma,
    - Use the min and max luma range of the original 4 samples
      - {NW, NE, SW, SE}
    - If 4-tap filter luma exceeds this range,
      - Assume invalid and use just the first 2 taps



If full-width filter width estimation is too large, then there is a chance the filter kernel will sample from regions off the local edge.

In this case noise will be introduced by the filter kernel.

This step attempts to remove this noise.



FXAA 3.11 Console on 360

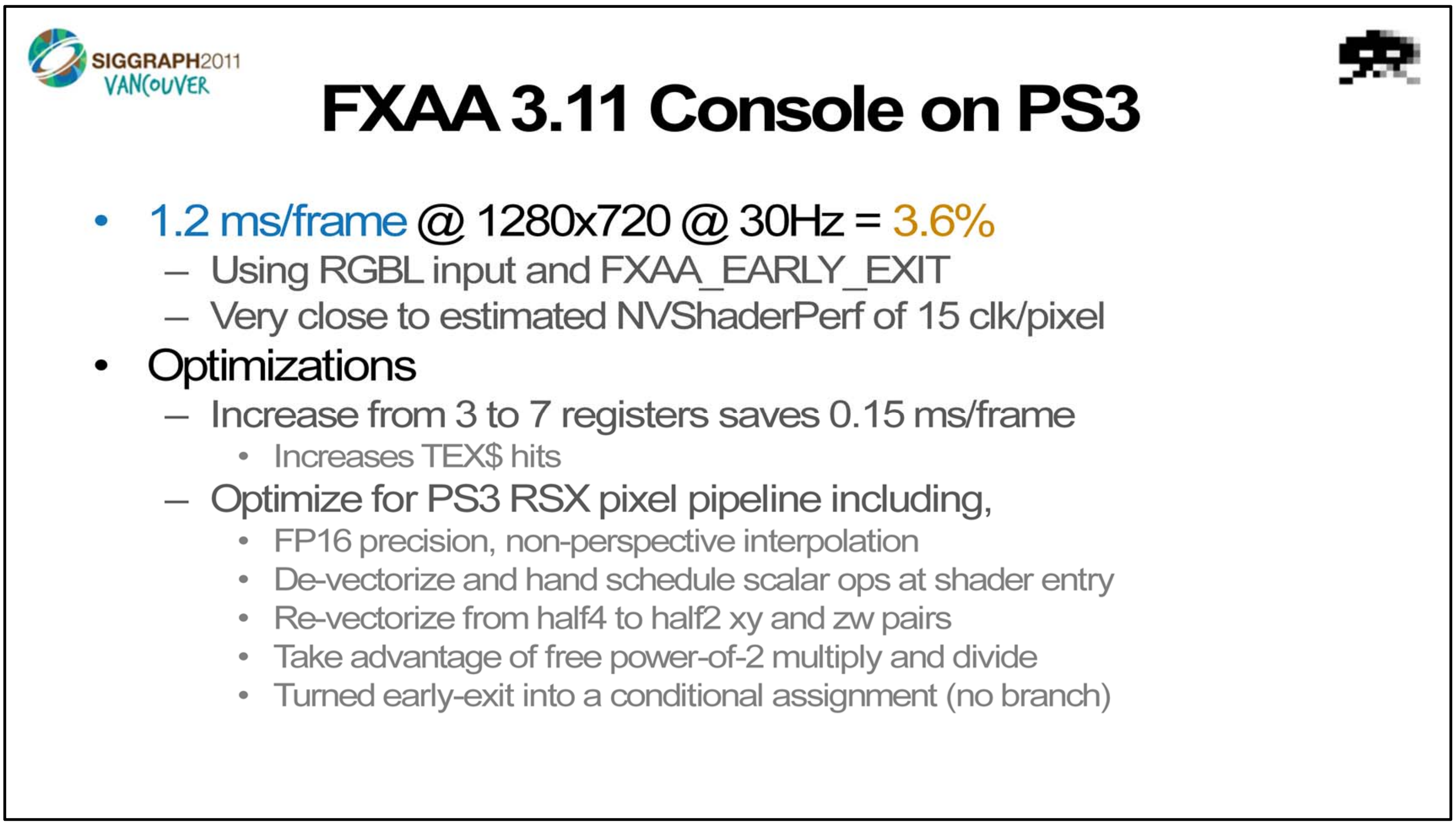
- 1.0 ms/frame @ 1280x720 @ 30Hz = 3%
  - 0.8 ms in shader + 0.2 ms for EDRAM resolve
  - Using FXAA_GREEN_AS_LUMA
- Optimizations
  - Use free texture sampler exponent bias
    - Alias multiple samplers to same input texture
  - Manual tfetch2D assembly to include offsets
  - Use early-exit branch
  - Optimize constant usage

```
/*============================================================================

                      FXAA3 CONSOLE - 360 PIXEL SHADER

------------------------------------------------------------------------------
This optimized version thanks to suggestions from Andy Luedke.
Should be fully tex bound in all cases.
============================================================================*/
#if (FXAA_360 == 1)
/*--------------------------------------------------------------------------*/
[reduceTempRegUsage(4)]
float4 FxaaPixelShader(
    // See FXAA Quality FxaaPixelShader() source for docs on Inputs!
    FxaaFloat2 pos,
    FxaaFloat4 fxaaConsolePosPos,
    FxaaTex tex,
    FxaaTex fxaaConsole360TexExpBiasNegOne,
    FxaaTex fxaaConsole360TexExpBiasNegTwo,
    FxaaFloat2 fxaaQualityRcpFrame,
    FxaaFloat4 fxaaConsoleRcpFrameOpt,
    FxaaFloat4 fxaaConsoleRcpFrameOpt2,
    FxaaFloat4 fxaaConsole360RcpFrameOpt2,
    FxaaFloat fxaaQualitySubpix,
    FxaaFloat fxaaQualityEdgeThreshold,
    FxaaFloat fxaaQualityEdgeThresholdMin,
    FxaaFloat fxaaConsoleEdgeSharpness,
    FxaaFloat fxaaConsoleEdgeThreshold,
    FxaaFloat fxaaConsoleEdgeThresholdMin,
    FxaaFloat4 fxaaConsole360ConstDir
) {
/*--------------------------------------------------------------------------*/
    float4 lumaNwNeSwSe;
    #if (FXAA_GREEN_AS_LUMA == 0)
        asm {
            tfetch2D lumaNwNeSwSe.w___, tex, pos.xy, OffsetX = -0.5, OffsetY = -0.5,
UseComputedLOD=false
            tfetch2D lumaNwNeSwSe._w__, tex, pos.xy, OffsetX =  0.5, OffsetY = -0.5,
UseComputedLOD=false
            tfetch2D lumaNwNeSwSe.__w_, tex, pos.xy, OffsetX = -0.5, OffsetY =  0.5,
UseComputedLOD=false
            tfetch2D lumaNwNeSwSe.___w, tex, pos.xy, OffsetX =  0.5, OffsetY =  0.5,
UseComputedLOD=false
        };
    #else
        asm {
            tfetch2D lumaNwNeSwSe.y___, tex, pos.xy, OffsetX = -0.5, OffsetY = -0.5,
UseComputedLOD=false
            tfetch2D lumaNwNeSwSe._y__, tex, pos.xy, OffsetX =  0.5, OffsetY = -0.5,
UseComputedLOD=false
            tfetch2D lumaNwNeSwSe.__y_, tex, pos.xy, OffsetX = -0.5, OffsetY =  0.5,
UseComputedLOD=false
            tfetch2D lumaNwNeSwSe.___y, tex, pos.xy, OffsetX =  0.5, OffsetY =  0.5,
UseComputedLOD=false
        };
    #endif
/*--------------------------------------------------------------------------*/
    lumaNwNeSwSe.y += 1.0/384.0;
    float2 lumaMinTemp = min(lumaNwNeSwSe.xy, lumaNwNeSwSe.zw);
    float2 lumaMaxTemp = max(lumaNwNeSwSe.xy, lumaNwNeSwSe.zw);
    float lumaMin = min(lumaMinTemp.x, lumaMinTemp.y);
    float lumaMax = max(lumaMaxTemp.x, lumaMaxTemp.y);
/*--------------------------------------------------------------------------*/
    float4 rgbyM = tex2Dlod(tex, float4(pos.xy, 0.0, 0.0));
    #if (FXAA_GREEN_AS_LUMA == 0)
        float lumaMinM = min(lumaMin, rgbyM.w);
        float lumaMaxM = max(lumaMax, rgbyM.w);
    #else
        float lumaMinM = min(lumaMin, rgbyM.y);
        float lumaMaxM = max(lumaMax, rgbyM.y);
    #endif
    if((lumaMaxM - lumaMinM) < max(fxaaConsoleEdgeThresholdMin, lumaMax *
fxaaConsoleEdgeThreshold)) return rgbyM;
/*--------------------------------------------------------------------------*/
    float2 dir;
    dir.x = dot(lumaNwNeSwSe, fxaaConsole360ConstDir.yyxx);
    dir.y = dot(lumaNwNeSwSe, fxaaConsole360ConstDir.xyxy);
    dir = normalize(dir);
/*--------------------------------------------------------------------------*/
    float4 dir1 = dir.xyxy * fxaaConsoleRcpFrameOpt.xyzw;
/*--------------------------------------------------------------------------*/
    float4 dir2;
    float dirAbsMinTimesC = min(abs(dir.x), abs(dir.y)) * fxaaConsoleEdgeSharpness;
    dir2 = saturate(fxaaConsole360ConstDir.zzww * dir.xyxy / dirAbsMinTimesC + 0.5);
    dir2 = dir2 * fxaaConsole360RcpFrameOpt2.xyxy + fxaaConsole360RcpFrameOpt2.zwzw;
/*--------------------------------------------------------------------------*/
```

```
    float4 rgbyN1 = tex2Dlod(fxaaConsole360TexExpBiasNegOne, float4(pos.xy + dir1.xy, 0.0, 
0.0));
    float4 rgbyP1 = tex2Dlod(fxaaConsole360TexExpBiasNegOne, float4(pos.xy + dir1.zw, 0.0, 
0.0));
    float4 rgbyN2 = tex2Dlod(fxaaConsole360TexExpBiasNegTwo, float4(pos.xy + dir2.xy, 0.0, 
0.0));
    float4 rgbyP2 = tex2Dlod(fxaaConsole360TexExpBiasNegTwo, float4(pos.xy + dir2.zw, 0.0, 
0.0));
/*--------------------------------------------------------------------------*/
    float4 rgbyA = rgbyN1 + rgbyP1;
    float4 rgbyB = rgbyN2 + rgbyP2 * 0.5 + rgbyA;
/*--------------------------------------------------------------------------*/
    float4 rgbyR = ((rgbyB.w - lumaMax) > 0.0) ? rgbyA : rgbyB;
    rgbyR = ((rgbyB.w - lumaMin) > 0.0) ? rgbyR : rgbyA;
    return rgbyR; }
/*==========================================================================*/
#endif
```



FXAA 3.11 Console on PS3

- 1.2 ms/frame @ 1280x720 @ 30Hz = 3.6%
  - Using RGBL input and FXAA_EARLY_EXIT
  - Very close to estimated NVShaderPerf of 15 clk/pixel
- Optimizations
  - Increase from 3 to 7 registers saves 0.15 ms/frame
    - Increases TEX$ hits
  - Optimize for PS3 RSX pixel pipeline including,
    - FP16 precision, non-perspective interpolation
    - De-vectorize and hand schedule scalar ops at shader entry
    - Re-vectorize from half4 to half2 xy and zw pairs
    - Take advantage of free power-of-2 multiply and divide
    - Turned early-exit into a conditional assignment (no branch)

```
/*============================================================================

       FXAA3 CONSOLE - OPTIMIZED PS3 PIXEL SHADER (WITH EARLY EXIT)

==============================================================================
The code mostly matches the assembly.
I have a feeling that 14 cycles is possible, but was not able to get there.
Might have to increase register count to get full performance.
Note this shader does not use perspective interpolation.

Use the following cgc options,

 --fenable-bx2 --fastmath --fastprecision --nofloatbindings

Use of FXAA_GREEN_AS_LUMA currently adds a cycle (16 clks).
Will look at fixing this for FXAA 3.12.
------------------------------------------------------------------------------
                             NVSHADERPERF OUTPUT
------------------------------------------------------------------------------
For reference and to aid in debug, output of NVShaderPerf should match this,

Shader to schedule:
  0: texpkb h0.w(TRUE), v5.zyxx, #0
  2: addh h2.y(TRUE), h0.w, constant(0.001953, 0.000000, 0.000000, 0.000000).x
  4: texpkb h1.w(TRUE), v5.xwxx, #0
  6: addh h0.x(TRUE), h1.w, -h2.y
  7: texpkb h2.w(TRUE), v5.zwzz, #0
  9: minh h4.w(TRUE), h2.y, h2
 10: maxh h5.x(TRUE), h2.y, h2.w
 11: texpkb h0.w(TRUE), v5, #0
 13: addh h3.w(TRUE), -h0, h0.x
 14: addh h0.x(TRUE), h0.w, h0
 15: addh h0.z(TRUE), -h2.w, h0.x
 16: addh h0.x(TRUE), h2.w, h3.w
 17: minh h5.y(TRUE), h0.w, h1.w
 18: nrmh h2.xz(TRUE), h0_n
 19: minh_m8 h2.w(TRUE), |h2.x|, |h2.z|
 20: divx h4.xy(TRUE), h2_n.xzzw, h2_n.w
 21: movr r1.zw(TRUE), v4.xxxy
 22: maxh h2.w(TRUE), h0, h1
 23: fenct TRUE
 24: madr r0.xy(TRUE), -h2.xzzw, constant(cConst5.x, cConst5.y, cConst5.z, cConst5.w).zwzz,
r1.zwzz
 26: texpkb h0(TRUE), r0, #0
 28: maxh h5.x(TRUE), h2.w, h5
 29: minh h5.w(TRUE), h5.y, h4
 30: madr r1.xy(TRUE), h2.xzzw, constant(cConst5.x, cConst5.y, cConst5.z, cConst5.w).zwzz,
r1.zwzz
 32: texpkb h2(TRUE), r1, #0
 34: addh_d2 h2(TRUE), h0, h2
 35: texpkb h1(TRUE), v4, #0
 37: maxh h5.y(TRUE), h5.x, h1.w
 38: minh h4.w(TRUE), h1, h5
 39: madr r0.xy(TRUE), -h4, constant(cConst5.x, cConst5.y, cConst5.z, cConst5.w).xyxx,
r1.zwzz
 41: texpkb h0(TRUE), r0, #0
 43: addh_m8 h5.z(TRUE), h5.y, -h4.w
 44: madr r2.xy(TRUE), h4, constant(cConst5.x, cConst5.y, cConst5.z, cConst5.w).xyxx,
r1.zwzz
 46: texpkb h3(TRUE), r2, #0
 48: addh_d2 h0(TRUE), h0, h3
 49: addh_d2 h3(TRUE), h0, h2
 50: movh h0(TRUE), h3
 51: slth h3.x(TRUE), h3.w, h5.w
 52: sgth h3.w(TRUE), h3, h5.x
 53: addx.c0 rc(TRUE), h3.x, h3
 54: slth.c0 rc(TRUE), h5.z, h5
 55: movh h0(c0.NE.w), h2
 56: movh h0(c0.NE.x), h1

IPU0 ------ Simplified schedule: --------
Pass |  Unit  |  uOp |  PC:  Op
-----+--------+------+------------------------
   1 | SCT0/1 |  mov |   0:  TXLr h0.w, g[TEX1].zyxx, const.xxxx, TEX0;
     |    TEX |  txl |   0:  TXLr h0.w, g[TEX1].zyxx, const.xxxx, TEX0;
     |   SCB0 |  add |   2:  ADDh h2.y, h0.-w--, const.-x--;
     |        |      |
   2 | SCT0/1 |  mov |   4:  TXLr h1.w, g[TEX1].xwxx, const.xxxx, TEX0;
     |    TEX |  txl |   4:  TXLr h1.w, g[TEX1].xwxx, const.xxxx, TEX0;
     |   SCB0 |  add |   6:  ADDh h0.x, h1.w---,-h2.y---;
     |        |      |
   3 | SCT0/1 |  mov |   7:  TXLr h2.w, g[TEX1].zwzz, const.xxxx, TEX0;
     |    TEX |  txl |   7:  TXLr h2.w, g[TEX1].zwzz, const.xxxx, TEX0;
     |   SCB0 |  max |  10:  MAXh h5.x, h2.y---, h2.w---;
```

```
    |   SCB1 |  min |   9:  MINh h4.w, h2.---y, h2;
    |        |      |
  4 | SCT0/1 |  mov |  11:  TXLr h0.w, g[TEX1], const.xxxx, TEX0;
    |    TEX |  txl |  11:  TXLr h0.w, g[TEX1], const.xxxx, TEX0;
    |   SCB0 |  add |  14:  ADDh h0.x, h0.w---, h0;
    |   SCB1 |  add |  13:  ADDh h3.w,-h0, h0.---x;
    |        |      |
  5 |   SCT0 |  mad |  16:  ADDh h0.x, h2.w---, h3.w---;
    |   SCT1 |  mad |  15:  ADDh h0.z,-h2.--w-, h0.--x-;
    |   SCB0 |  min |  17:  MINh h5.y, h0.-w--, h1.-w--;
    |        |      |
  6 |   SCT1 |  mov |  18:  NRMh h2.xz, h0;
    |    SRB |  nrm |  18:  NRMh h2.xz, h0;
    |   SCB1 |  min |  19:  MINh*8 h2.w, |h2.---x|, |h2.---z|;
    |        |      |
  7 |   SCT0 |  div |  20:  DIVx h4.xy, h2.xz--, h2.ww--;
    |   SCT1 |  mov |  21:  MOVr r1.zw, g[TEX0].--xy;
    |   SCB1 |  max |  22:  MAXh h2.w, h0, h1;
    |        |      |
  8 |   SCT0 |  mad |  24:  MADr r0.xy,-h2.xz--, const.zw--, r1.zw--;
    |   SCT1 |  mov |  26:  TXLr h0, r0, const.xxxx, TEX0;
    |    TEX |  txl |  26:  TXLr h0, r0, const.xxxx, TEX0;
    |   SCB0 |  max |  28:  MAXh h5.x, h2.w---, h5;
    |   SCB1 |  min |  29:  MINh h5.w, h5.---y, h4;
    |        |      |
  9 |   SCT0 |  mad |  30:  MADr r1.xy, h2.xz--, const.zw--, r1.zw--;
    |   SCT1 |  mov |  32:  TXLr h2, r1, const.xxxx, TEX0;
    |    TEX |  txl |  32:  TXLr h2, r1, const.xxxx, TEX0;
    | SCB0/1 |  add |  34:  ADDh/2 h2, h0, h2;
    |        |      |
 10 | SCT0/1 |  mov |  35:  TXLr h1, g[TEX0], const.xxxx, TEX0;
    |    TEX |  txl |  35:  TXLr h1, g[TEX0], const.xxxx, TEX0;
    |   SCB0 |  max |  37:  MAXh h5.y, h5.-x--, h1.-w--;
    |   SCB1 |  min |  38:  MINh h4.w, h1, h5;
    |        |      |
 11 |   SCT0 |  mad |  39:  MADr r0.xy,-h4, const.xy--, r1.zw--;
    |   SCT1 |  mov |  41:  TXLr h0, r0, const.zzzz, TEX0;
    |    TEX |  txl |  41:  TXLr h0, r0, const.zzzz, TEX0;
    |   SCB0 |  mad |  44:  MADr r2.xy, h4, const.xy--, r1.zw--;
    |   SCB1 |  add |  43:  ADDh*8 h5.z, h5.--y-,-h4.--w-;
    |        |      |
 12 | SCT0/1 |  mov |  46:  TXLr h3, r2, const.xxxx, TEX0;
    |    TEX |  txl |  46:  TXLr h3, r2, const.xxxx, TEX0;
    | SCB0/1 |  add |  48:  ADDh/2 h0, h0, h3;
    |        |      |
 13 | SCT0/1 |  mad |  49:  ADDh/2 h3, h0, h2;
    | SCB0/1 |  mul |  50:  MOVh h0, h3;
    |        |      |
 14 |   SCT0 |  set |  51:  SLTh h3.x, h3.w---, h5.w---;
    |   SCT1 |  set |  52:  SGTh h3.w, h3, h5.---x;
    |   SCB0 |  set |  54:  SLThc0 rc, h5.z---, h5;
    |   SCB1 |  add |  53:  ADDxc0_s rc, h3.---x, h3;
    |        |      |
 15 | SCT0/1 |  mul |  55:  MOVh h0(NE0.wwww), h2;
    | SCB0/1 |  mul |  56:  MOVh h0(NE0.xxxx), h1;
```

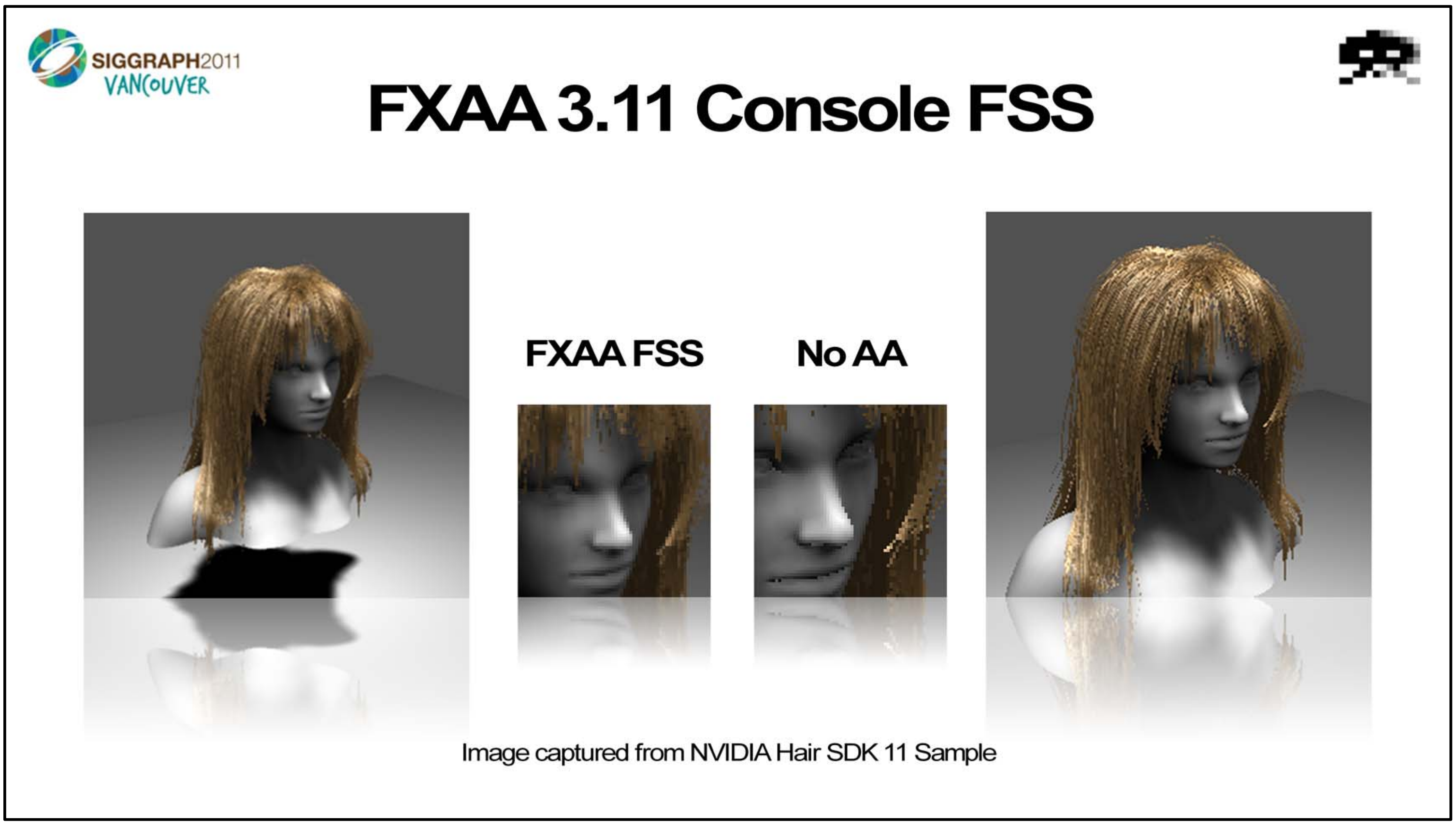
```
Pass   SCT  TEX  SCB
  1:   0% 100%  25%
  2:   0% 100%  25%
  3:   0% 100%  50%
  4:   0% 100%  50%
  5:  50%   0%  25%
  6:   0%   0%  25%
  7: 100%   0%  25%
  8:   0% 100%  50%
  9:   0% 100% 100%
 10:   0% 100%  50%
 11:   0% 100%  75%
 12:   0% 100% 100%
 13: 100%   0% 100%
 14:  50%   0%  50%
 15: 100%   0% 100%

MEAN:  26%  60%  56%

Pass   SCT0  SCT1   TEX  SCB0  SCB1
  1:    0%    0%  100%  100%    0%
  2:    0%    0%  100%  100%    0%
  3:    0%    0%  100%  100%  100%
  4:    0%    0%  100%  100%  100%
  5:  100%  100%    0%  100%    0%
  6:    0%    0%    0%    0%  100%
  7:  100%  100%    0%    0%  100%
  8:    0%    0%  100%  100%  100%
  9:    0%    0%  100%  100%  100%
 10:    0%    0%  100%  100%  100%
 11:    0%    0%  100%  100%  100%
 12:    0%    0%  100%  100%  100%
 13:  100%  100%    0%  100%  100%
 14:  100%  100%    0%  100%  100%
 15:  100%  100%    0%  100%  100%

MEAN:   33%   33%   60%   86%   80%
Fragment Performance Setup: Driver RSX Compiler, GPU RSX, Flags 0x5
Results 15 cycles, 3 r regs, 800,000,000 pixels/s
============================================================================*/
#if (FXAA_PS3 == 1) && (FXAA_EARLY_EXIT == 1)
/*--------------------------------------------------------------------------*/
#pragma regcount 7
#pragma disablepc all
#pragma option O2
#pragma option OutColorPrec=fp16
#pragma texformat default RGBA8
/*==========================================================================*/
half4 FxaaPixelShader(
    // See FXAA Quality FxaaPixelShader() source for docs on Inputs!
    FxaaFloat2 pos,
    FxaaFloat4 fxaaConsolePosPos,
    FxaaTex tex,
    FxaaTex fxaaConsole360TexExpBiasNegOne,
    FxaaTex fxaaConsole360TexExpBiasNegTwo,
    FxaaFloat2 fxaaQualityRcpFrame,
    FxaaFloat4 fxaaConsoleRcpFrameOpt,
    FxaaFloat4 fxaaConsoleRcpFrameOpt2,
    FxaaFloat4 fxaaConsole360RcpFrameOpt2,
    FxaaFloat fxaaQualitySubpix,
    FxaaFloat fxaaQualityEdgeThreshold,
    FxaaFloat fxaaQualityEdgeThresholdMin,
    FxaaFloat fxaaConsoleEdgeSharpness,
    FxaaFloat fxaaConsoleEdgeThreshold,
    FxaaFloat fxaaConsoleEdgeThresholdMin,
    FxaaFloat4 fxaaConsole360ConstDir
) {
/*--------------------------------------------------------------------------*/
// (1)
    half4 rgbyNe = h4tex2Dlod(tex, half4(fxaaConsolePosPos.zy, 0, 0));
    #if (FXAA_GREEN_AS_LUMA == 0)
        half lumaNe = rgbyNe.w + half(1.0/512.0);
    #else
        half lumaNe = rgbyNe.y + half(1.0/512.0);
    #endif
/*--------------------------------------------------------------------------*/
// (2)
    half4 lumaSw = h4tex2Dlod(tex, half4(fxaaConsolePosPos.xw, 0, 0));
    #if (FXAA_GREEN_AS_LUMA == 0)
        half lumaSwNegNe = lumaSw.w - lumaNe;
    #else
        half lumaSwNegNe = lumaSw.y - lumaNe;
    #endif
/*--------------------------------------------------------------------------*/
// (3)
    half4 lumaNw = h4tex2Dlod(tex, half4(fxaaConsolePosPos.xy, 0, 0));
    #if (FXAA_GREEN_AS_LUMA == 0)
        half lumaMaxNwSw = max(lumaNw.w, lumaSw.w);
        half lumaMinNwSw = min(lumaNw.w, lumaSw.w);
    #else
        half lumaMaxNwSw = max(lumaNw.y, lumaSw.y);
        half lumaMinNwSw = min(lumaNw.y, lumaSw.y);
    #endif
/*--------------------------------------------------------------------------*/
// (4)
    half4 lumaSe = h4tex2Dlod(tex, half4(fxaaConsolePosPos.zw, 0, 0));
    #if (FXAA_GREEN_AS_LUMA == 0)
        half dirZ =  lumaNw.w + lumaSwNegNe;
        half dirX = -lumaNw.w + lumaSwNegNe;
    #else
        half dirZ =  lumaNw.y + lumaSwNegNe;
        half dirX = -lumaNw.y + lumaSwNegNe;
    #endif
/*--------------------------------------------------------------------------*/
// (5)
    half3 dir;
    dir.y = 0.0;
    #if (FXAA_GREEN_AS_LUMA == 0)
        dir.x =  lumaSe.w + dirX;
        dir.z = -lumaSe.w + dirZ;
        half lumaMinNeSe = min(lumaNe, lumaSe.w);
    #else
        dir.x =  lumaSe.y + dirX;
        dir.z = -lumaSe.y + dirZ;
        half lumaMinNeSe = min(lumaNe, lumaSe.y);
    #endif
/*--------------------------------------------------------------------------*/
// (6)
    half4 dir1_pos;
    dir1_pos.xy = normalize(dir).xz;
    half dirAbsMinTimes8 = min(abs(dir1_pos.x), abs(dir1_pos.y)) *
half(FXAA_CONSOLE__PS3_EDGE_SHARPNESS);
/*--------------------------------------------------------------------------*/
// (7)
    half4 dir2_pos;
    dir2_pos.xy = clamp(dir1_pos.xy / dirAbsMinTimes8, half(-2.0), half(2.0));
    dir1_pos.zw = pos.xy;
    dir2_pos.zw = pos.xy;
    #if (FXAA_GREEN_AS_LUMA == 0)
        half lumaMaxNeSe = max(lumaNe, lumaSe.w);
    #else
        half lumaMaxNeSe = max(lumaNe, lumaSe.y);
    #endif
/*--------------------------------------------------------------------------*/
// (8)
    half4 temp1N;
    temp1N.xy = dir1_pos.zw - dir1_pos.xy * fxaaConsoleRcpFrameOpt.zw;
    temp1N = h4tex2Dlod(tex, half4(temp1N.xy, 0.0, 0.0));
    half lumaMax = max(lumaMaxNwSw, lumaMaxNeSe);
    half lumaMin = min(lumaMinNwSw, lumaMinNeSe);
/*--------------------------------------------------------------------------*/
// (9)
    half4 rgby1;
    rgby1.xy = dir1_pos.zw + dir1_pos.xy * fxaaConsoleRcpFrameOpt.zw;
    rgby1 = h4tex2Dlod(tex, half4(rgby1.xy, 0.0, 0.0));
    rgby1 = (temp1N + rgby1) * 0.5;
/*--------------------------------------------------------------------------*/
// (10)
    half4 rgbyM = h4tex2Dlod(tex, half4(pos.xy, 0.0, 0.0));
    #if (FXAA_GREEN_AS_LUMA == 0)
        half lumaMaxM = max(lumaMax, rgbyM.w);
        half lumaMinM = min(lumaMin, rgbyM.w);
    #else
        half lumaMaxM = max(lumaMax, rgbyM.y);
        half lumaMinM = min(lumaMin, rgbyM.y);
    #endif
/*--------------------------------------------------------------------------*/
// (11)
    half4 temp2N;
    temp2N.xy = dir2_pos.zw - dir2_pos.xy * fxaaConsoleRcpFrameOpt2.zw;
    temp2N = h4tex2Dlod(tex, half4(temp2N.xy, 0.0, 0.0));
    half4 rgby2;
    rgby2.xy = dir2_pos.zw + dir2_pos.xy * fxaaConsoleRcpFrameOpt2.zw;
    half lumaRangeM = (lumaMaxM - lumaMinM) / FXAA_CONSOLE__PS3_EDGE_THRESHOLD;
```

```
/*--------------------------------------------------------------------------*/
// (12)
    rgby2 = h4tex2Dlod(tex, half4(rgby2.xy, 0.0, 0.0));
    rgby2 = (temp2N + rgby2) * 0.5;
/*--------------------------------------------------------------------------*/
// (13)
    rgby2 = (rgby2 + rgby1) * 0.5;
/*--------------------------------------------------------------------------*/
// (14)
    #if (FXAA_GREEN_AS_LUMA == 0)
        bool twoTapLt = rgby2.w < lumaMin;
        bool twoTapGt = rgby2.w > lumaMax;
    #else
        bool twoTapLt = rgby2.y < lumaMin;
        bool twoTapGt = rgby2.y > lumaMax;
    #endif
    bool earlyExit = lumaRangeM < lumaMax;
    bool twoTap = twoTapLt || twoTapGt;
/*--------------------------------------------------------------------------*/
// (15)
    if(twoTap) rgby2 = rgby1;
    if(earlyExit) rgby2 = rgbyM;
/*--------------------------------------------------------------------------*/
    return rgby2; }
/*==========================================================================*/
#endif
```



FXAA 3.11 Console FSS

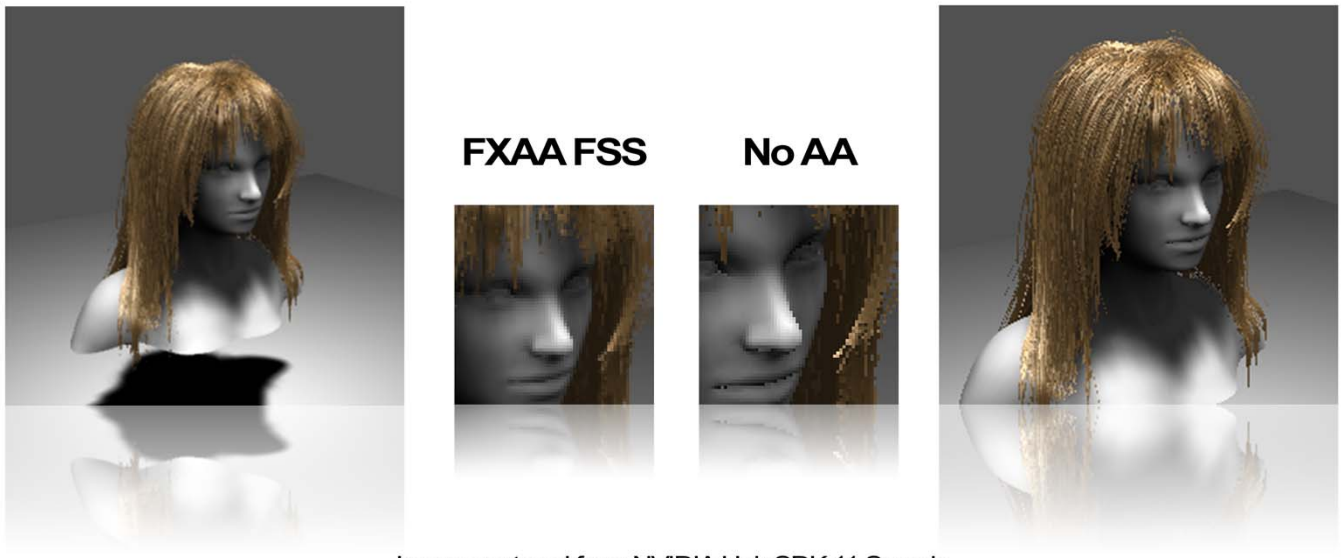


Image captured from NVIDIA Hair SDK 11 Sample

An idea for getting higher quality output, fractional Super-Sampling with FXAA Console

(1.) Render slightly larger than output resolution

(2.) Use FXAA Console instead of scaling copy (FXAA reads the larger source frame, FXAA writes directly to the smaller output frame)

(3.) Dynamically adjust output resolution to maintain a set frame rate

FXAA 3.11 Console cost is set by (the smaller) output resolution.

Technically FXAA 3.11 Console can enlarge up to 2x also,

results hide the bilinear sampling enlargement artifacts with a noisy painterly effect.

This image shows worst case aliasing example, and thus the limits of FSS with FXAA Console.

FXAA 3.11 Quality Preset 13

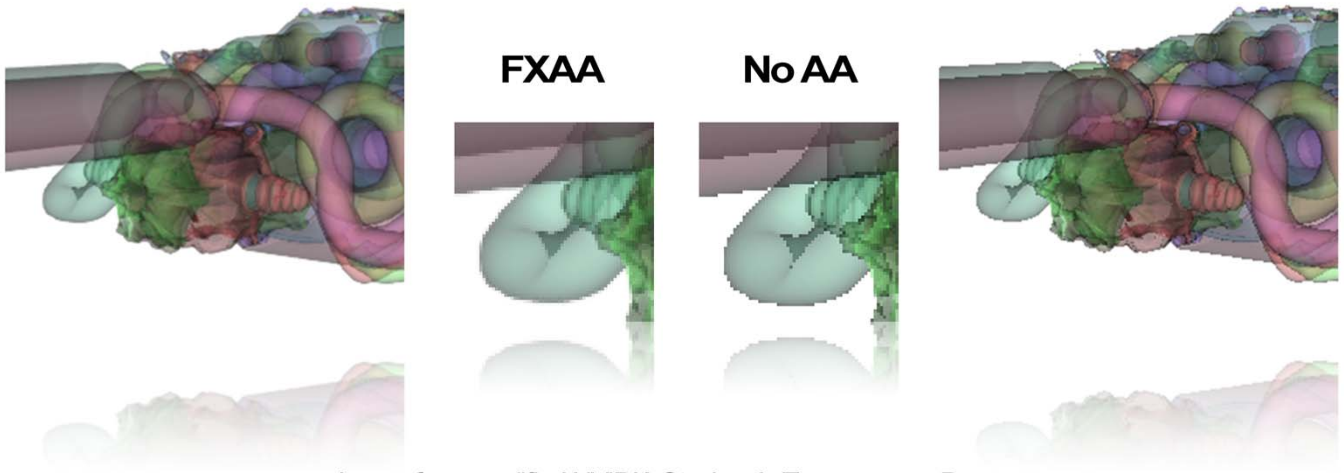


Image from modified NVIDIA Stochastic Transparency Demo

FXAA Quality chooses a Vert or Horz direction then does a Sub-pixel Shift in the Direction of Highest Contrast

(In short)
Similar early exit to FXAA Console
MLAA like, search in one direction only, extra shift to attenuate contrast on sub-pixel aliasing
Search steps defined by preset

(Advantages)
Quality, low cost
0.21 ms/Mpix on GTX480 (for default preset)

FXAA Quality Algorithm,

(1.) Early exit similar to FXAA Console but using the N,S,W,E, and middle pixel.

(2.) Then using the 3x3 pixel neighborhood, choose the dominate edge direction (either vertical or horizontal).

(3.) Also use the 3x3 pixel neighborhood to estimate sub-pixel aliasing.

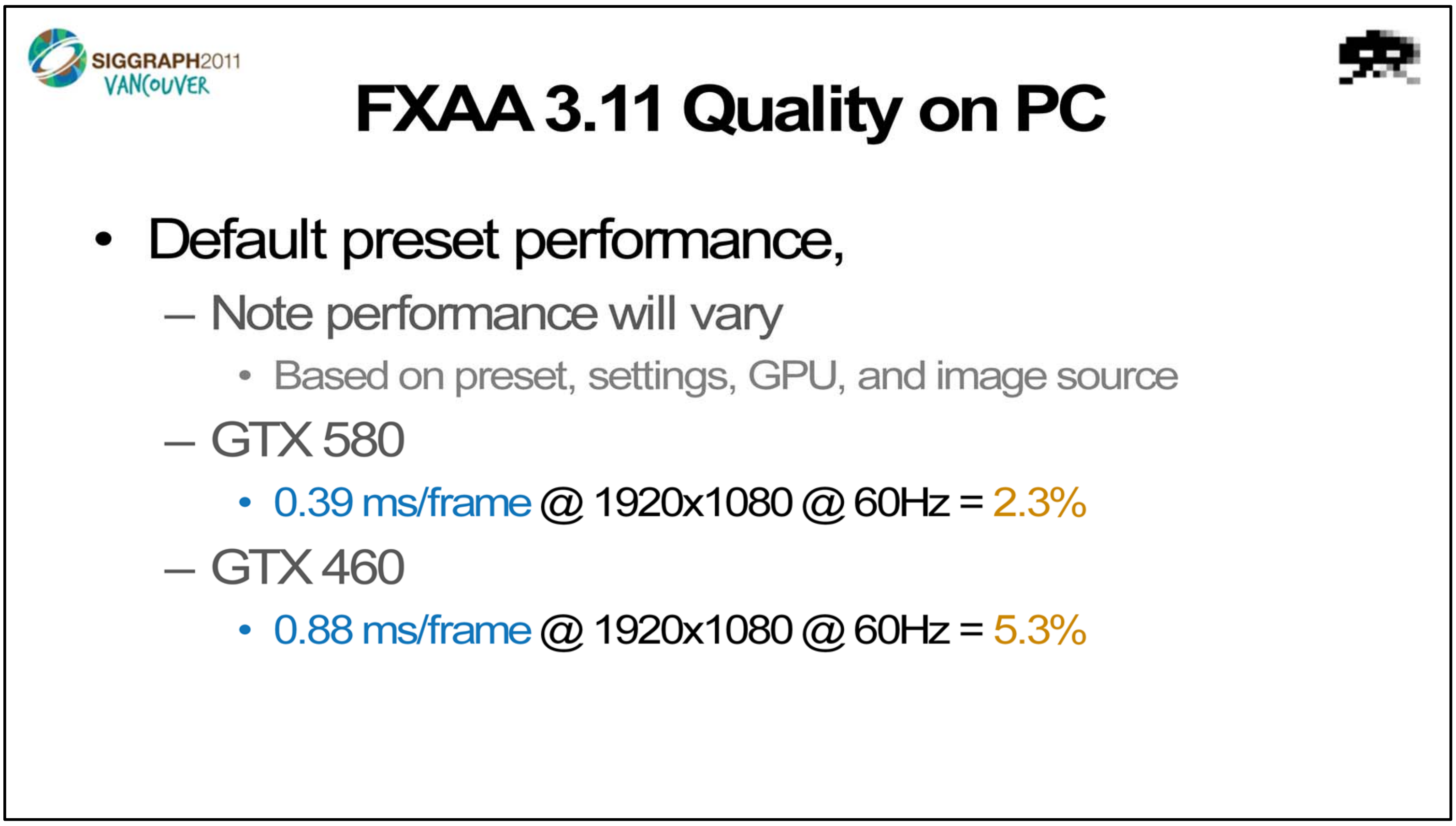
(4.) Then choose the side of that direction which has the highest contrast.

(5.) Search for end-of-edge using positions defined by the preset.
Search samples center of either a pair of pixels of a quad of pixels along span.

(6.) Convert end-of-edge to a sub-pixel shift using 3 simple cases.

(7.) Convert sub-pixel aliasing detection to a sub-pixel shift.

(8.) Re-sample the pixel given the sub-pixel shift.



FXAA 3.11 Quality on PC

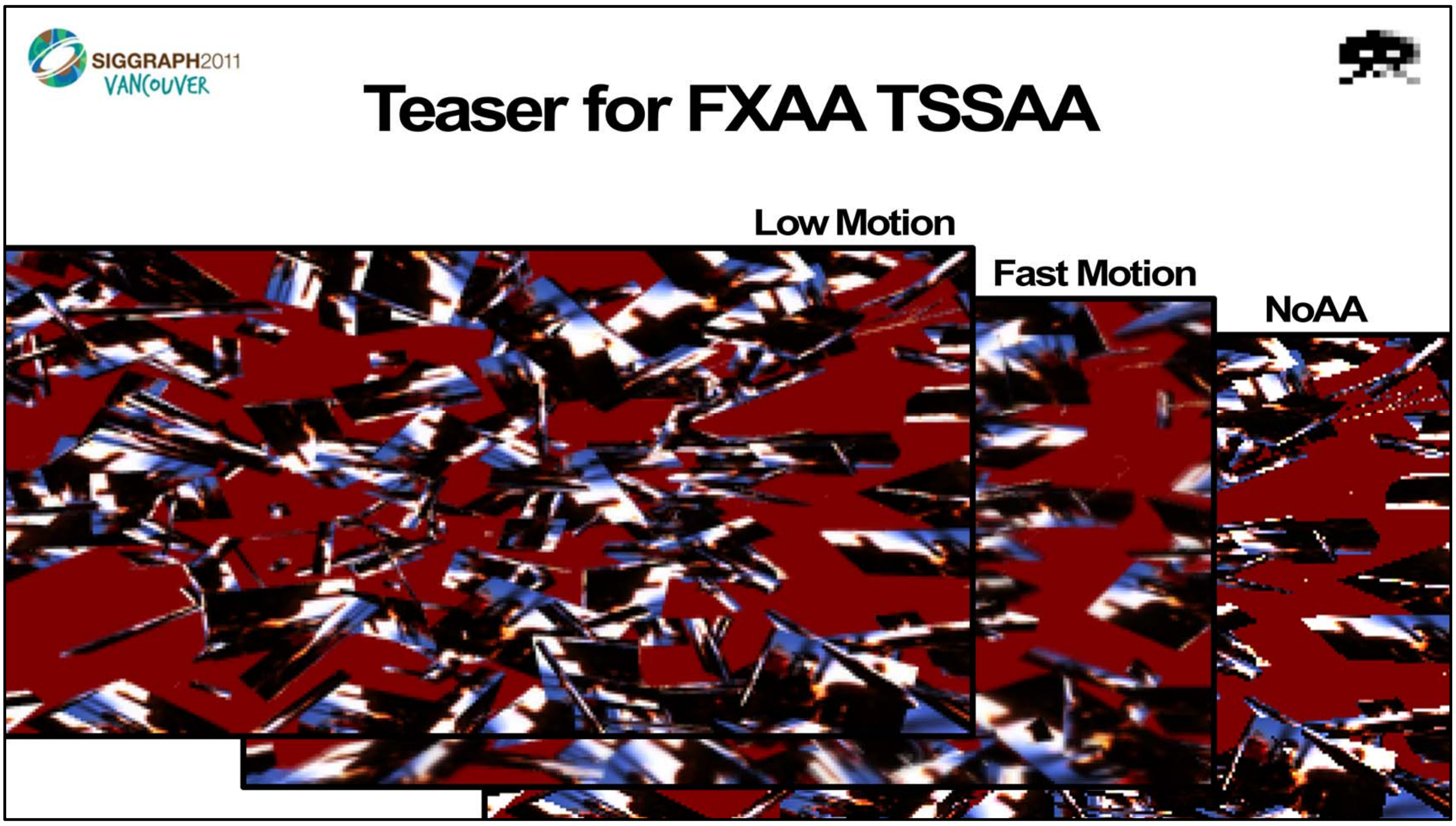
- Default preset performance,
  - Note performance will vary
    - Based on preset, settings, GPU, and image source
  - GTX 580
    - 0.39 ms/frame @ 1920x1080 @ 60Hz = 2.3%
  - GTX 460
    - 0.88 ms/frame @ 1920x1080 @ 60Hz = 5.3%

FXAA 3.11 Quality FSS

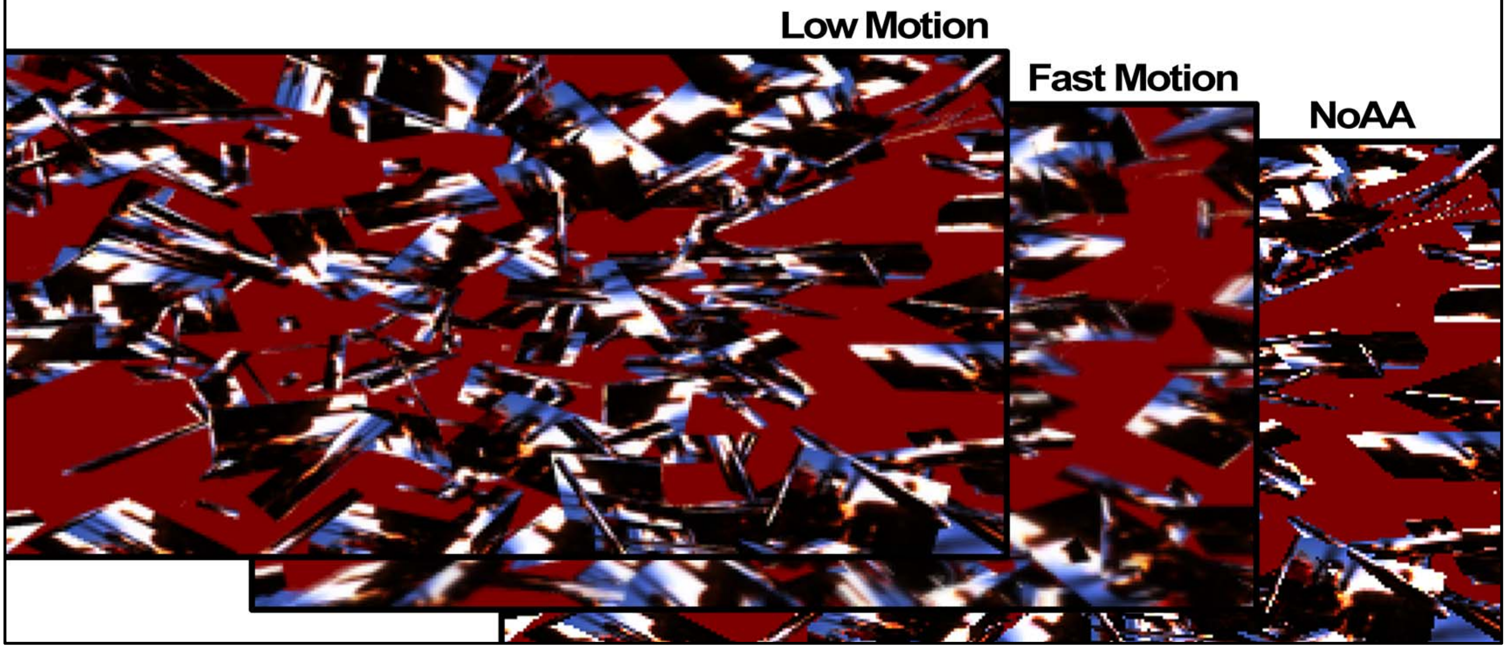
Similar but higher quality method from the FXAA Console FSS.
Runs FXAA Quality at the larger resolution, then down-samples.

Simple method to mix some super-sampling with FXAA.

(1.) Render at {width*1.3333, height*1.3333} resolution (1.77x the number of pixels).
(2.) Apply FXAA.
(3.) Down-sample to display resolution {width, height}.
(4.) Apply HUD/UI.
(5.) Scan-out to display (perhaps even using a hardware scalar to up-sample on Xbox).



Teaser for FXAA TSSAA



FXAA Temporal Super-Sampling Anti-Aliasing

Relaxes the constraint that FXAA use only input color, adds some new requirements,

(1.) Requires motion vector field as input (can be half size).

(2.) Requires full size prior FXAA TSSAA output color as input.

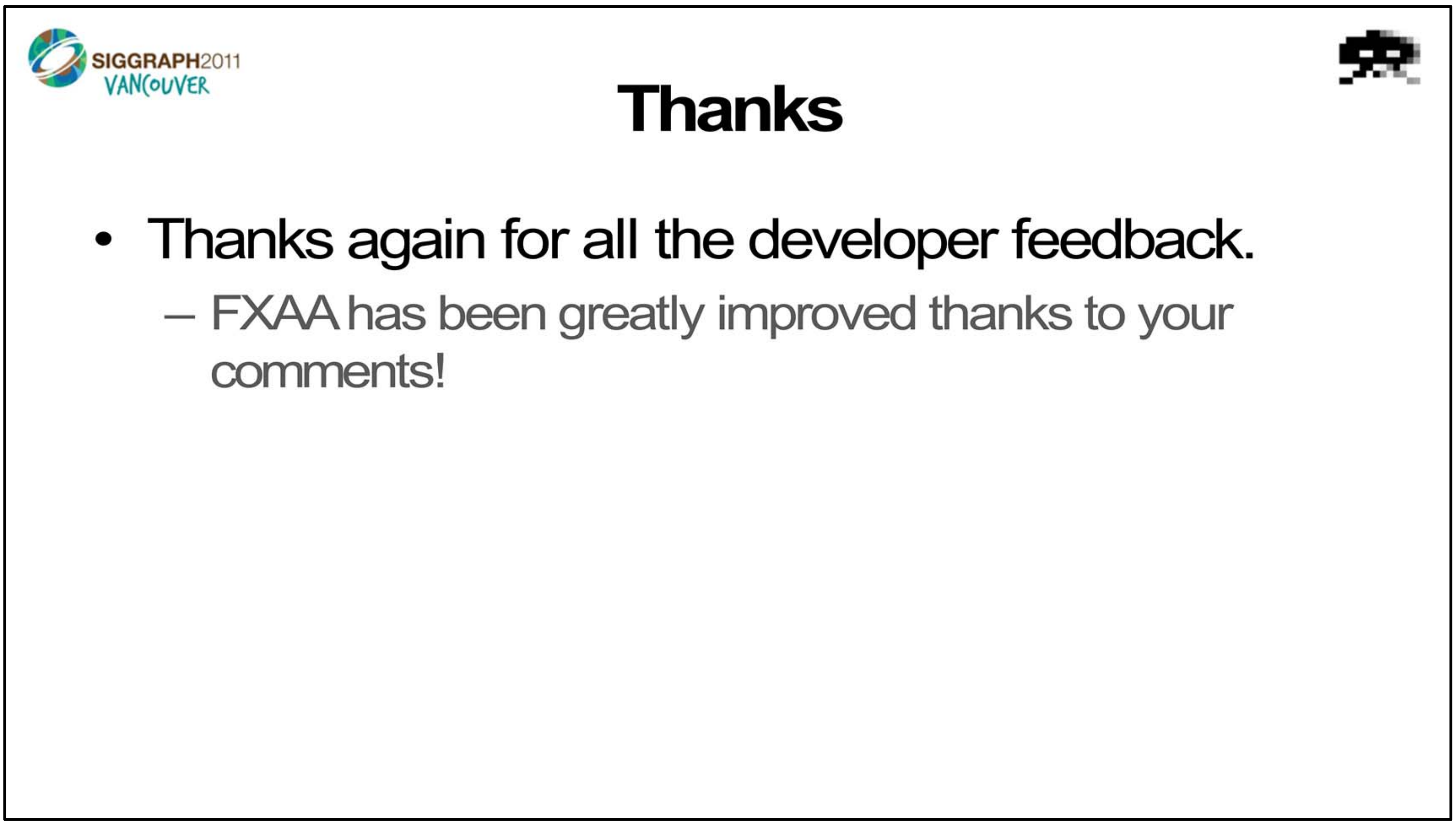
(3.) Requires each frame be rendered at a different sub-pixel offset (following an 8x or 16x MSAA pattern).

8x or 16x super-sampled output under low motion.

Gets slightly blurry under fast motion for fast lighting changes (slight loss of resolution).

No ghosting.

Doubles as a noise reduction filter.



Thanks

- Thanks again for all the developer feedback.
  - FXAA has been greatly improved thanks to your comments!