INTRODUCING FIDELITYFX VARIABLE SHADING

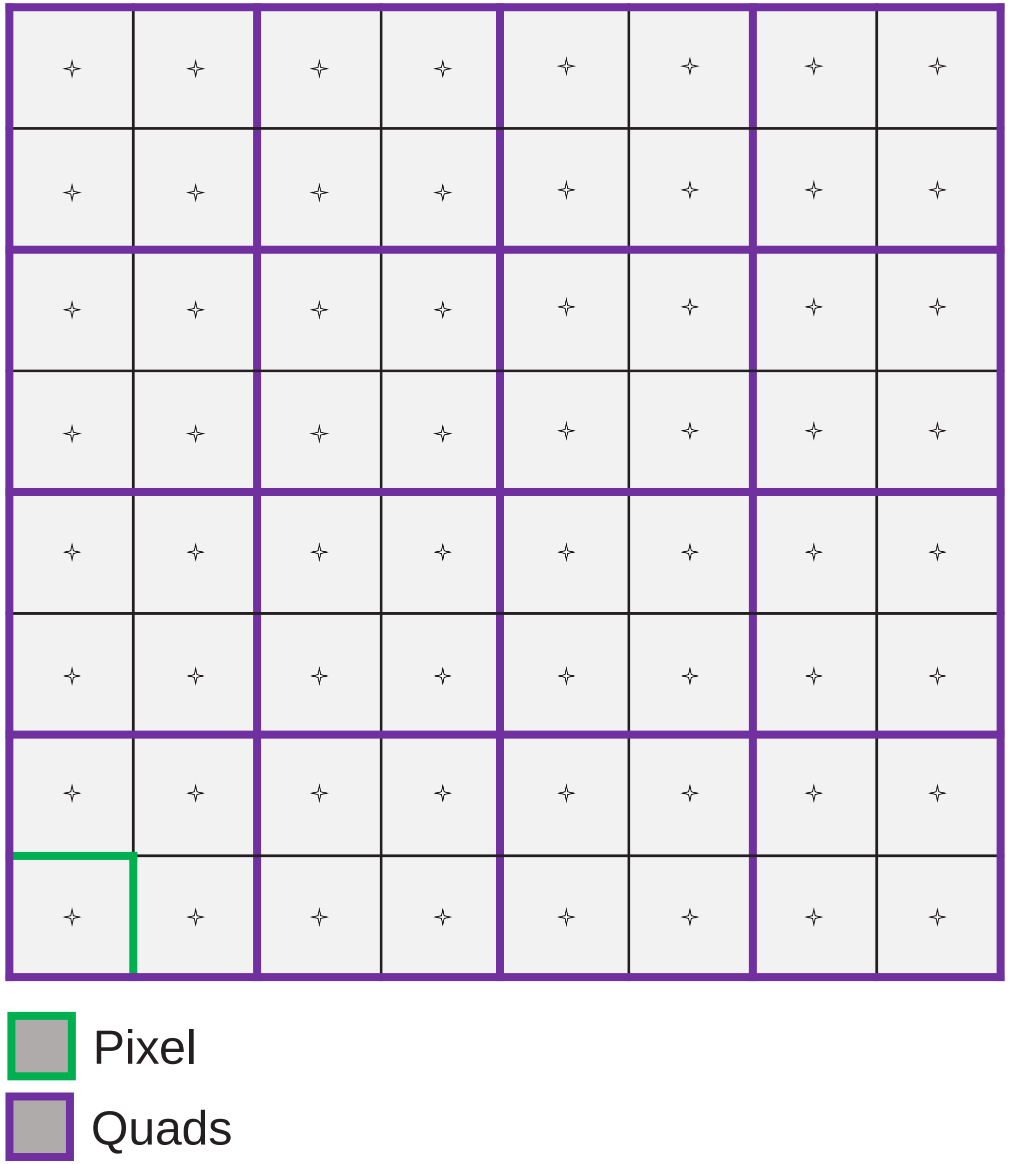
STEPHAN HODES

DEVELOPER TECHNOLOGY ENGINEER, AMD

INTRODUCTION TO VARIABLE RATE SHADING

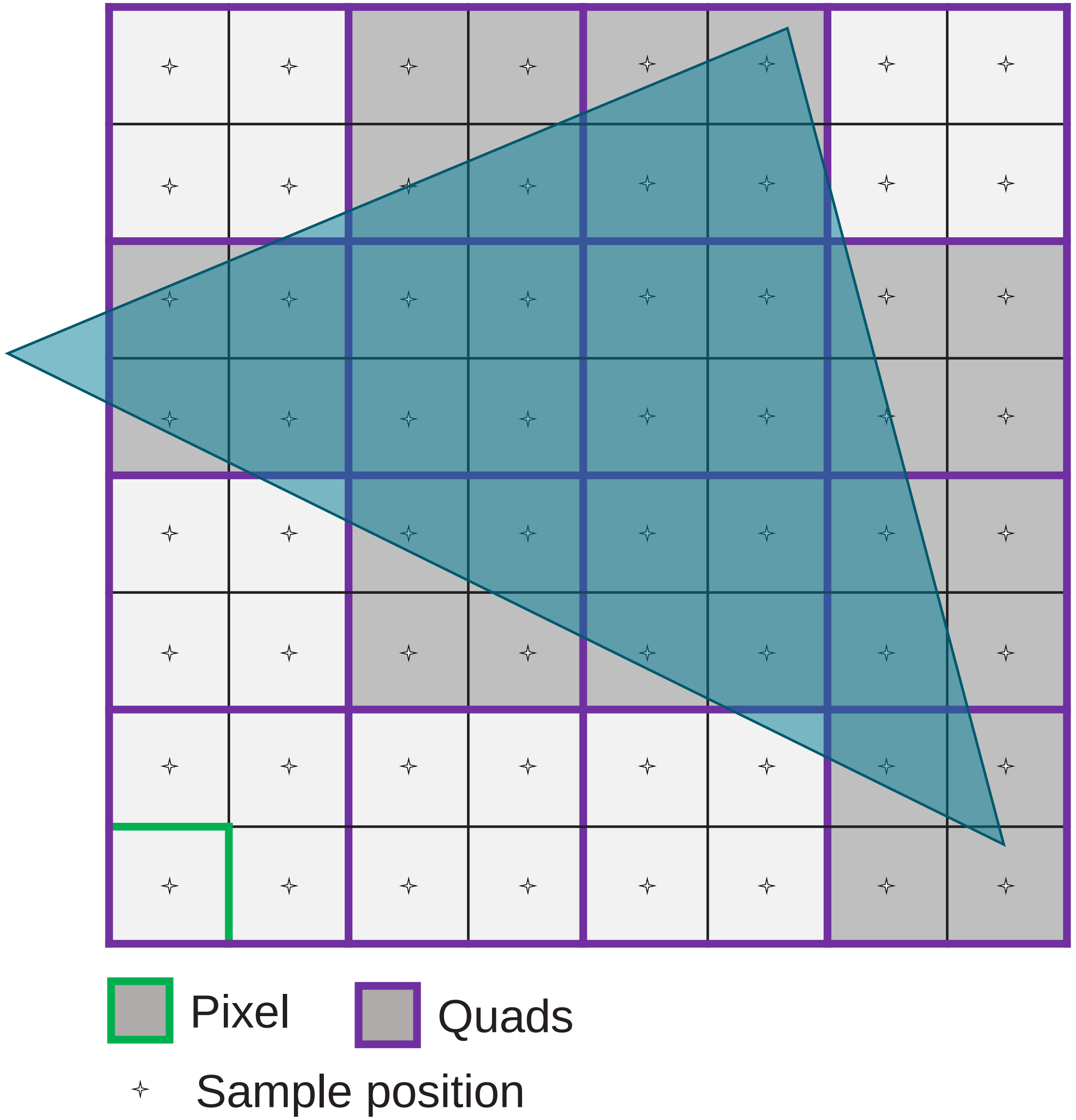
- Variable Rate Shading (VRS) is a feature of DirectX®12 Ultimate
- Goal of VRS is to save GPU work
  (where it does not significantly contribute to the final frame)
- Games today are usually played at very high resolution
  - Pixels are very small on screen
  - Adjacent pixels often have similar color
    (if they belong to the same primitive)
- Post-processing effects like Antialiasing, Depth of Field, or Motion Blur further reduce the difference between adjacent pixels

THE CONCEPT OF VRS



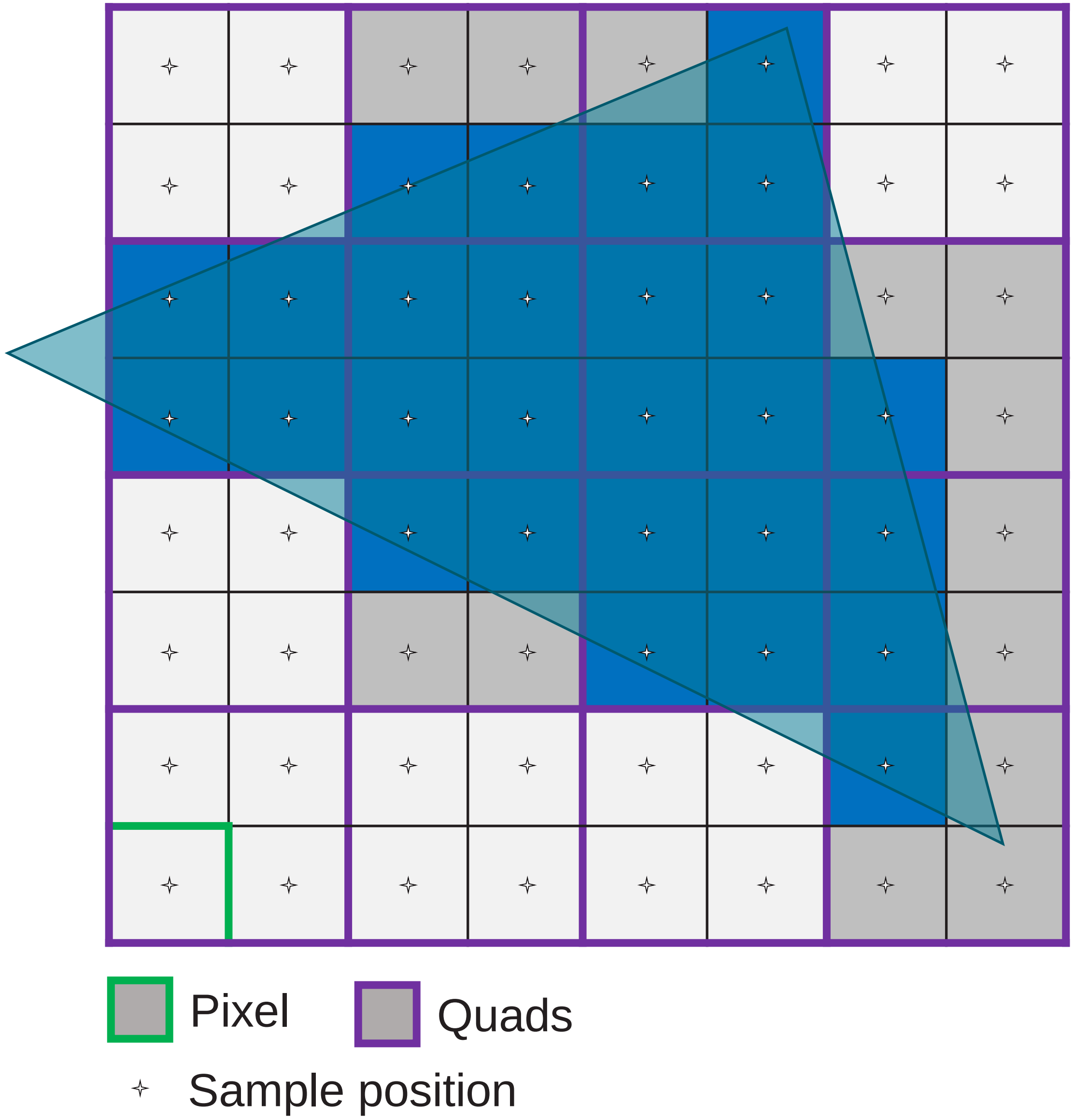
THE CONCEPT OF VRS

- **Without VRS**, 4 pixel shader (PS) threads are getting generated for every quad of which at least one pixel is covered by a primitive



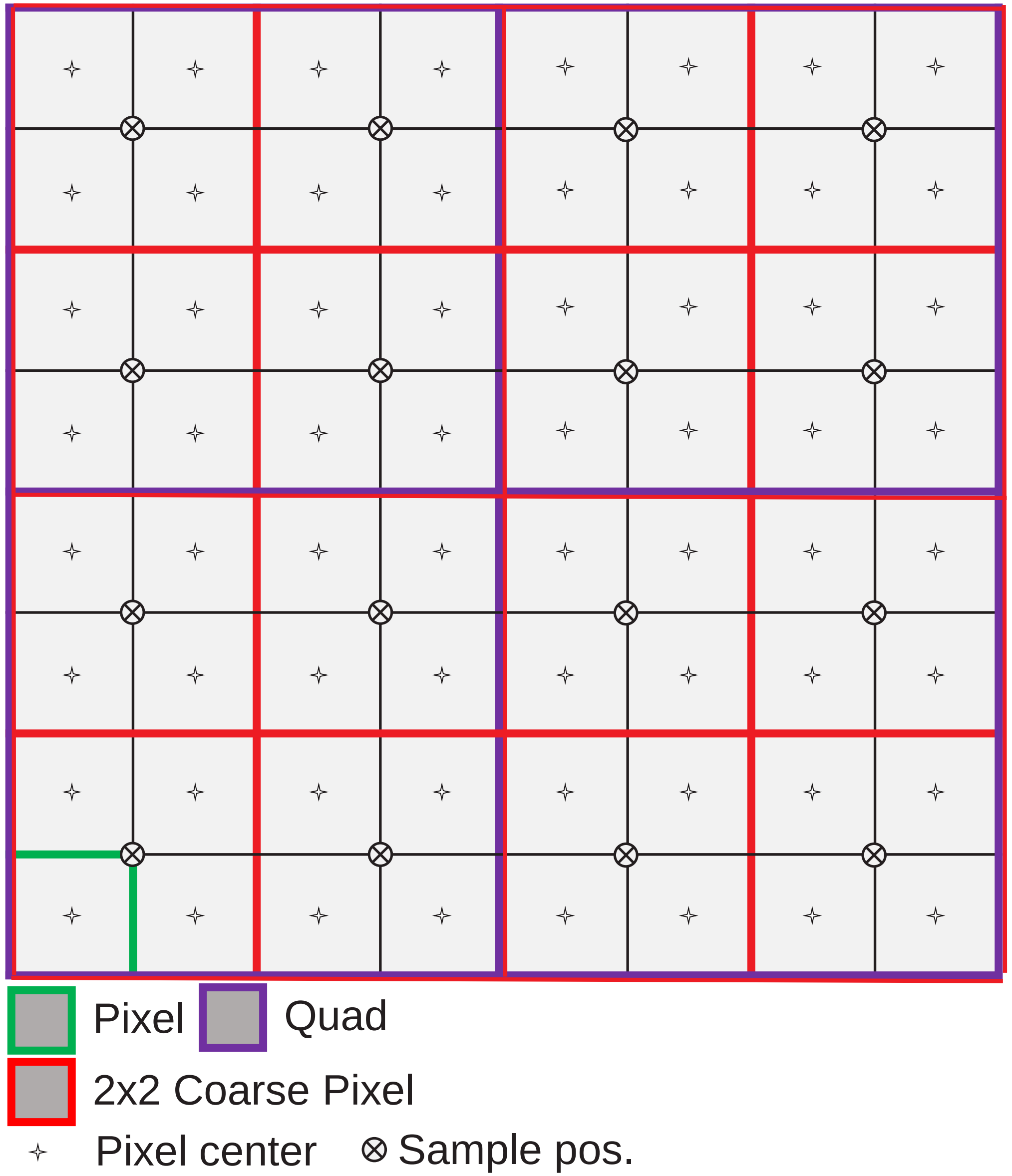
THE CONCEPT OF VRS

- **Without VRS**, 4 pixel shader (PS) threads are getting generated for every quad of which at least one pixel is covered by a primitive
  - For every pixel where the sample-position is covered by the primitive, the result of the PS gets written to the render target
  - Example:
    10 Quads/40 PS threads (27 active)



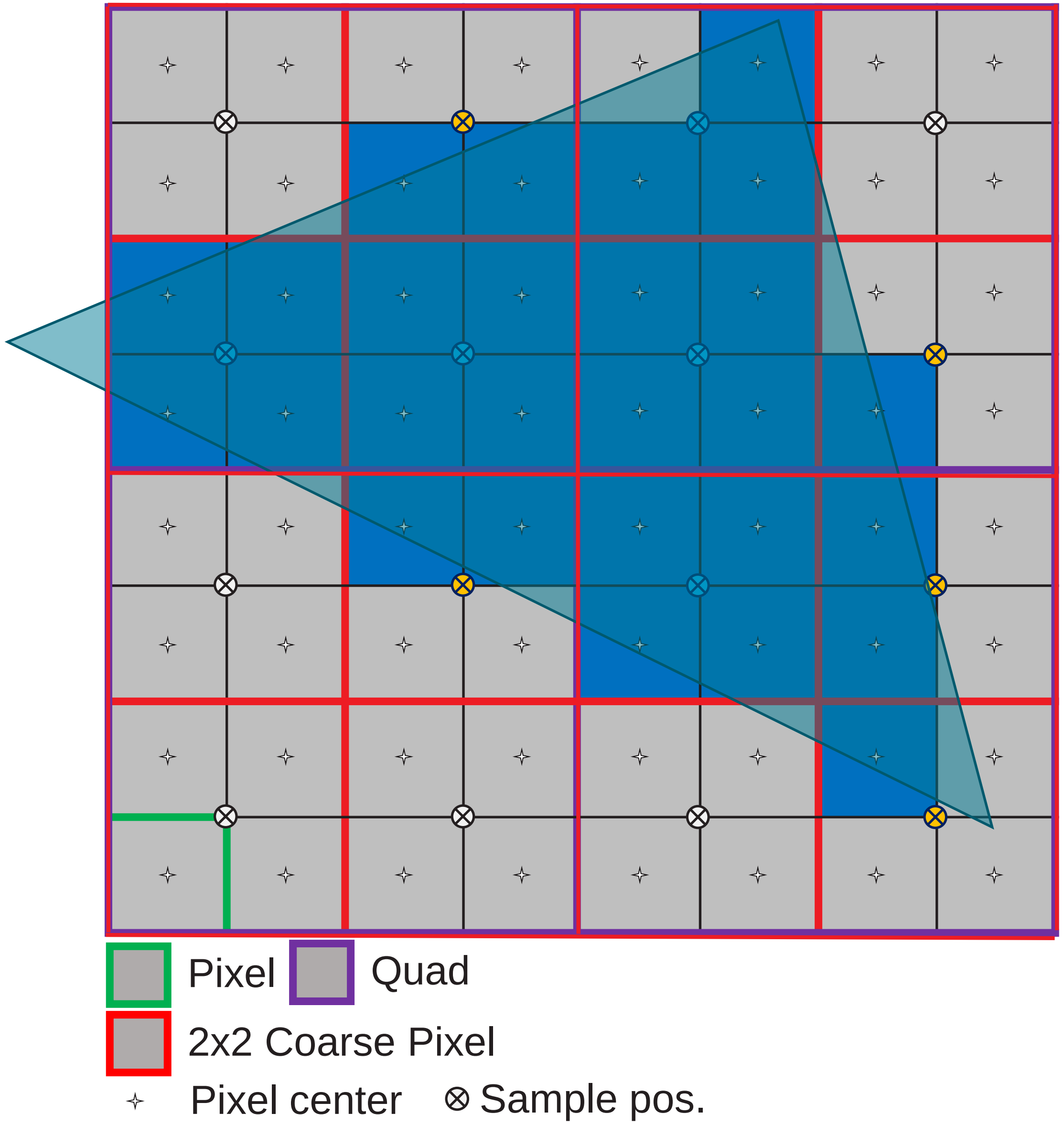
THE CONCEPT OF VRS

- **Without VRS**, 4 pixel shader (PS) threads are getting generated for every quad of which at least one pixel is covered by a primitive
  - For every pixel where the sample-position is covered by the primitive, the result of the PS gets written to the render target
  - Example:
    10 Quads/40 PS threads (27 active)
- **With VRS** one or multiple pixels form a coarse pixel (2x2 in this example)



THE CONCEPT OF VRS

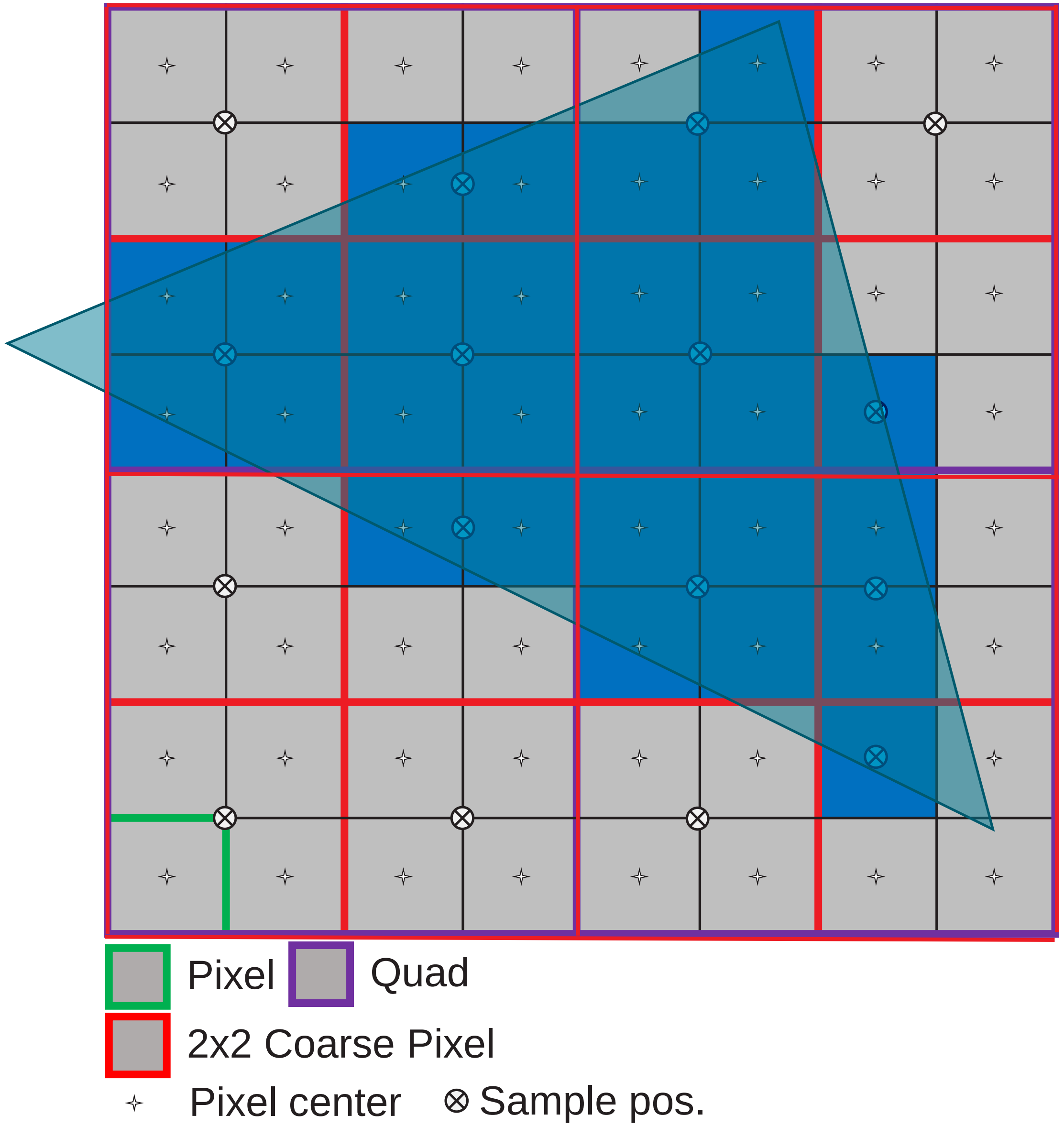
- **Without VRS**, 4 pixel shader (PS) threads are getting generated for every quad of which at least one pixel is covered by a primitive
  - For every pixel where the sample-position is covered by the primitive, the result of the PS gets written to the render target
  - Example:
    10 Quads/40 PS threads (27 active)
- **With VRS** one or multiple pixels form a coarse pixel (2x2 in this example)
  - Example:
    4 Quads/16 PS threads (10 active)
  - VRS only reduces shading quality within a triangle, the geometry edges are preserved



THE CONCEPT OF VRS

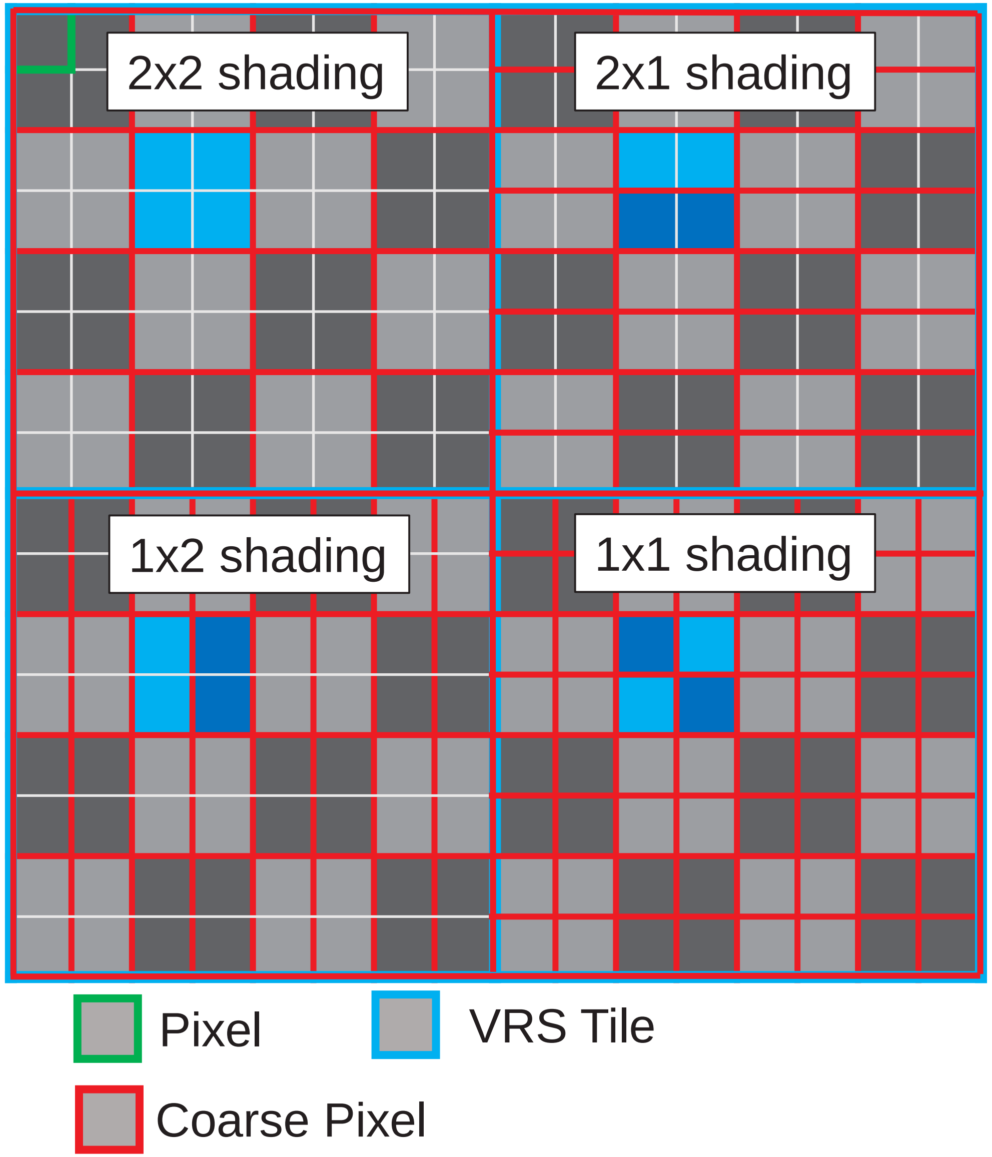
- **Without VRS**, 4 pixel shader (PS) threads are getting generated for every quad of which at least one pixel is covered by a primitive
  - For every pixel where the sample-position is covered by the primitive, the result of the PS gets written to the render target
  - Example:
    10 Quads/40 PS threads (27 active)
- **With VRS** one or multiple pixels form a coarse pixel (2x2 in this example)
  - Example:
    4 Quads/16 PS threads (10 active)
  - VRS only reduces shading quality within a triangle, the geometry edges are preserved

Make sure to use centroid interpolation!



VRS ON RDNA2

- VRS has multiple ways to control shading rate
  - Per drawcall (VRS tier 1)
  - Per primitive (VRS tier 2, VS/GS output)
  - Per screen tile (VRS tier 2, Image Based)
    - **8x8 pixel tile size**
    - Small tile size provides fine grained control
- **Additional shading rates not supported**
  - At common resolutions 4x can hardly be used without generating visual artifacts
  - Additional shading rates make image generation more complex
- VRS image gets copied into H-tile on bind
  - Small (but not negligible) overhead when binding the VRS image
  - No overhead during rendering!



IMPLEMENTING VRS (INITIALIZATION)

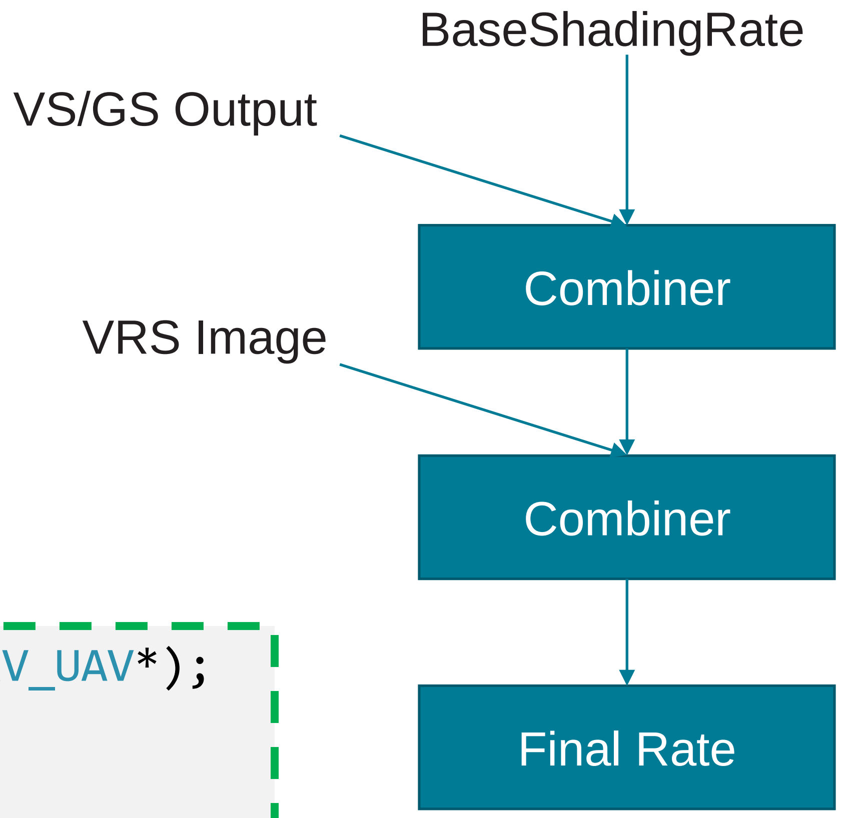
- Query Hardware details:
  - Is VRS supported / which shading rates?
    - Supporting 4x4 shading rate makes the image generation shader more complex
    - 4x4 is likely to cause visible quality degradation at common resolutions
  - Is tier 2 (Shader or Image Based VRS) supported?
    - For image based: What is the tile size?
- Create VRS Image & generation shader

```
void OnCreate(Device *pDevice,
       ResourceViewHeaps *pResourceViewHeaps,
       DynamicBufferRing *pConstantBufferRing,
       StaticBufferPool *pStaticBufferPool,
       DXGI_FORMAT overlayOutputFormat);
void OnDestroy();
void OnCreateWindowSizeDependentResources(uint32_t w, uint32_t h);
void OnDestroyWindowSizeDependentResources();
```

IMPLEMENTING VRS (RENDERING)

- Compute VRS image
- Bind VRS image
- Set base shading rate and combiners
- Unbind VRS image when done
- [Render VRS image as overlay for debugging]

```
void ComputeVrsMap(ID3D12GraphicsCommandList*, CBV_SRV_UAV*);
void SetShadingRate(D3D12_SHADING_RATE,
       const D3D12_SHADING_RATE_COMBINER*,
       ID3D12GraphicsCommandList*);
void StartVrsRendering(ID3D12GraphicsCommandList*);
void EndVrsRendering(ID3D12GraphicsCommandList*);
void DrawOverlay(ID3D12GraphicsCommandList*);
```



FIDELITYFX VARIABLE SHADING (CPP)

```cpp
struct FFX_VariableShading_CB
{
        uint32_t width, height;
        uint32_t tileSize;
        float varianceCutoff;
        float motionFactor;
};

static void FFX_VariableShading_GetVrsImageResourceDesc(
        const uint32_t rtWidth, const uint32_t rtHeight,
        const uint32_t tileSize,
        CD3DX12_RESOURCE_DESC& VRSImageDesc);
static void FFX_VariableShading_GetDispatchInfo(
        const FFX_Variable_Shading_CB* cb,
        const bool useAditionalShadingRates,
        uint32_t& numThreadGroupsX, uint32_t& numThreadGroupsY)
```

FIDELITYFX VARIABLE SHADING (HLSL)

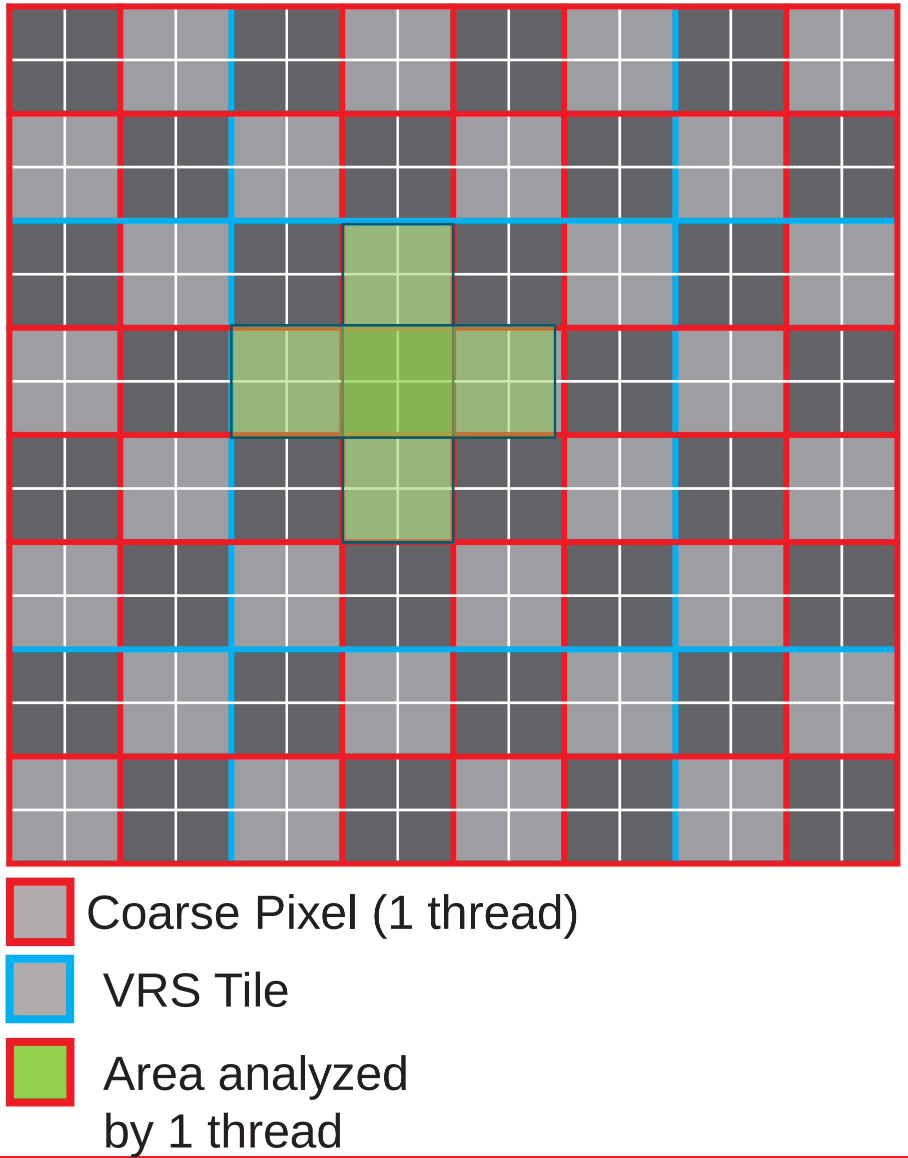
```
// Define: FFX_VARIABLESHADING_TILESIZE
// Optional: FFX_VARIABLESHADING_ADDITIONALSHADINGRATES

// Constant Buffer
cbuffer FFX_VariableShading_CB0 {
       int2    g_Resolution;
       uint    g_TileSize;
       float   g_VarianceCutoff;
       float   g_MotionFactor;
}

// Forward declaration of functions that need to be implemented
// by shader code using this technique
float  FFX_VariableShading_ReadLuminance(int2 pos);
float2 FFX_VariableShading_ReadMotionVec2D(int2 pos);
void   FFX_VariableShading_WriteVrsImage (int2 pos, uint value);
```

FIDELITYFX VARIABLE SHADING (HLSL USAGE)

```
// define FFX_VARIABLESHADING_TILESIZE on compile!
// may define FFX_VARIABLESHADING_ADDITIONALSHADINGRATES
RWTexture2D<uint>     imgDestination: register(u0);
Texture2D             texColor      : register(t0);
Texture2D             texVelocity   : register(t1);

#define FFX_HLSL 1
#include "ffx_variable_shading.h"

float FFX_VariableShading_ReadLuminance(int2 pos) {
       float3 color = texColor[pos].xyz;
       return dot(color, float3(0.30, 0.59, 0.11));
}

float2 FFX_VariableShading_ReadMotionVec2D(int2 pos) {
       return texVelocity[pos].xy * float2(0.5f, -0.5f) * g_Resolution;
}
```

HOW THE SHADER WORKS

1. One threadgroup computes between 1 and 4 tiles
   - Without additional shading rates, and 8x8 tile size, each group of 8x8 threads computes the shading rate for 4 tiles.
2. Analyze pairs of pixels within 2x2 region
   - Using LDS
   - Each thread also takes pixels outside the 2x2 box into account. This avoids burn-in (i.e. Low VRS rate because it was low in last frame)
   - Reduce luminance delta by motion influence
3. Compute largest luminance delta within tile
   1. Using wave intrinsics
4. Compare against threshold
5. Write out VRSImage

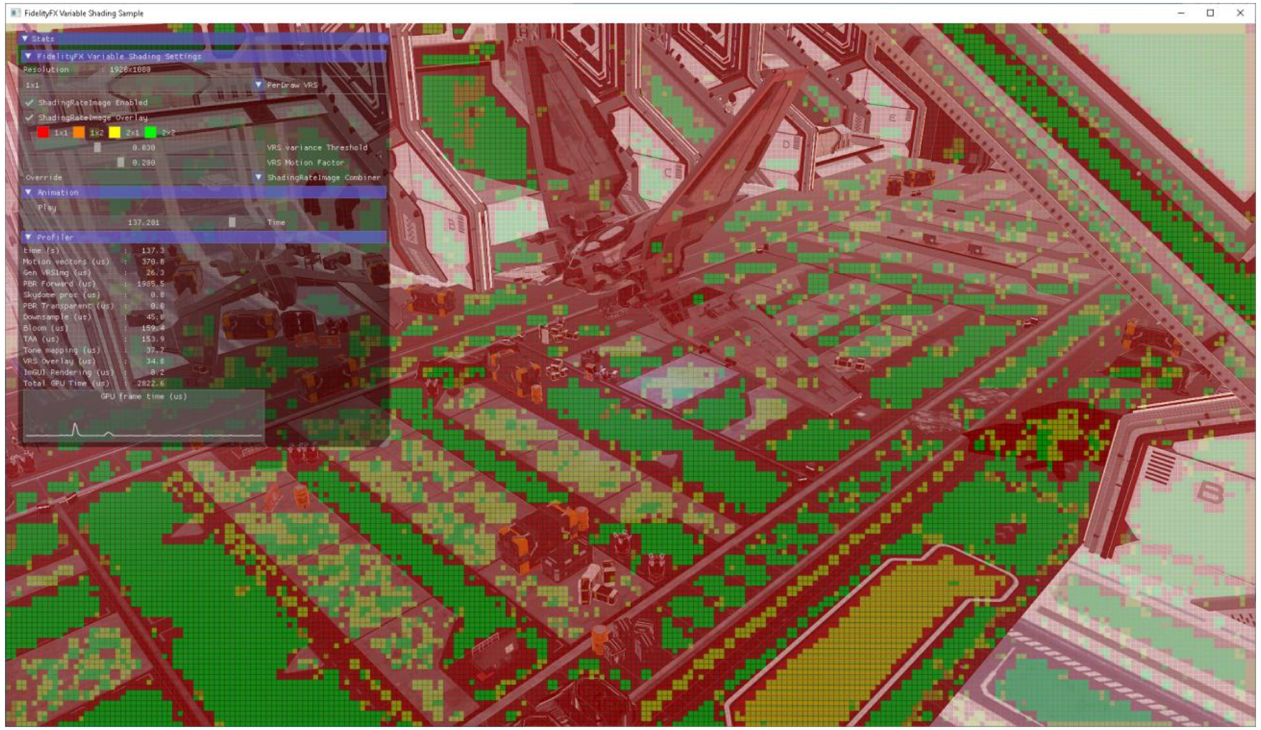


VRS OVERLAY

- Display VRS image as overlay for debugging tweaking
- Ready to use code in VrsOverlay.hlsl
- Easy to integrate:
  1. Build VS/PS pipeline and bind it
  2. Provide constant buffer containing resolution and tile size

     (same as for VRS image generation)
  3. Draw a single triangle

     (No vertex or index buffers required)


FidelityFX Variable Shading Sample
▼ Stats
▼ FidelityFX Variable Shading Settings
🟥 1x1 🟧 1x2 🟨 2x1 🟩 2x2
▼ Animation
▼ Profiler
GPU frame time (us)

```
Resolution          : 1920x1080
1x1                                  ▼ PerDraw VRS
✓ ShadingRateImage Enabled
✓ ShadingRateImage Overlay
                0.030                VRS variance Threshold
                0.200                VRS Motion Factor
Override                             ▼ ShadingRateImage Combiner
Play
              137.201                Time
```

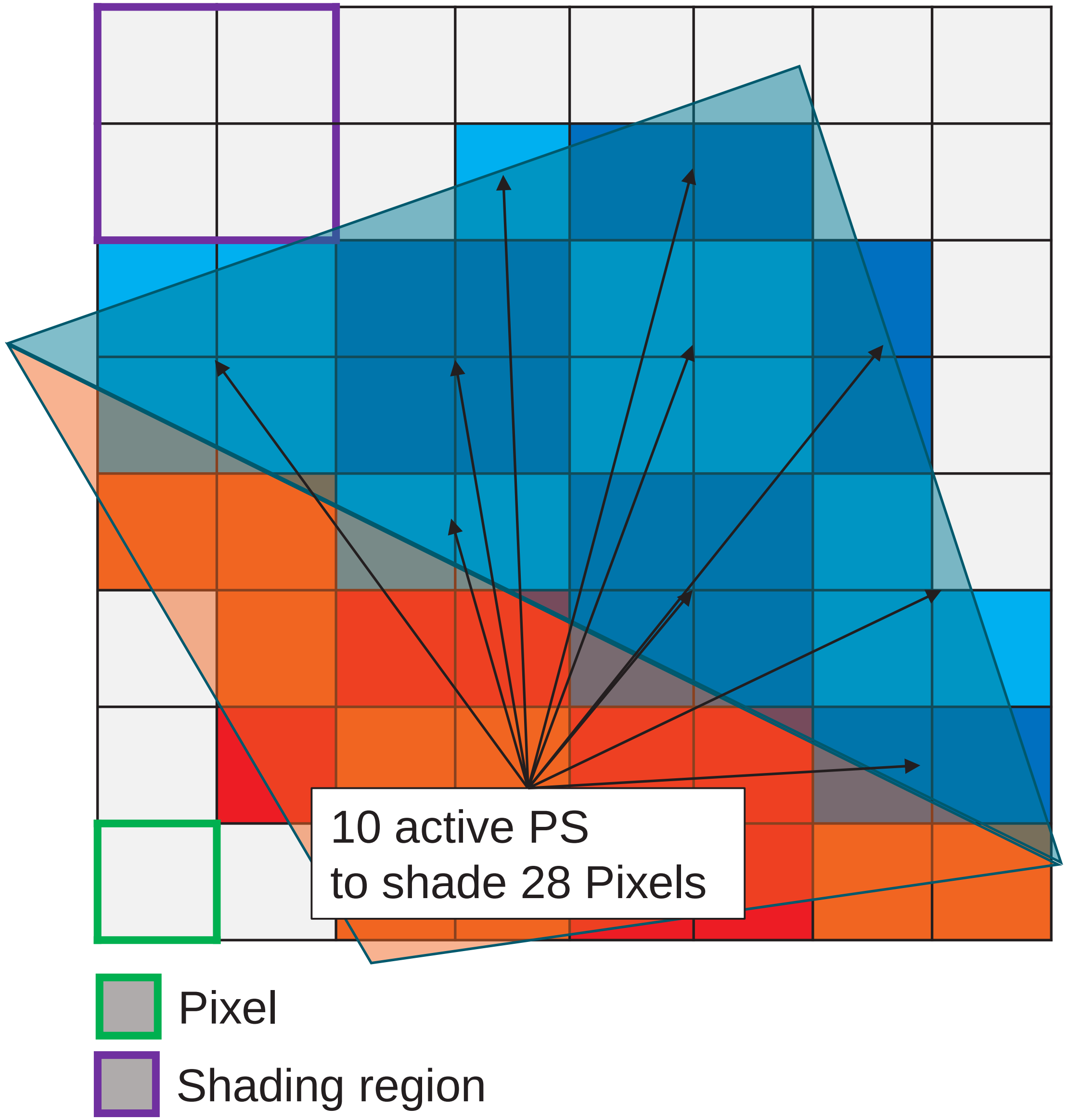
```
time (s)               :   137.3
Motion vectors (us)    :   370.8
Gen VRSImg (us)        :    26.3
PBR Forward (us)       :  1985.5
Skydome proc (us)      :     0.0
PBR Transparent (us)   :     0.0
Downsample (us)        :    45.6
Bloom (us)             :   159.4
TAA (us)               :   153.9
Tone mapping (us)      :    37.7
VRS Overlay (us)       :    34.0
ImGUI Rendering (us)   :     8.2
Total GPU Time (us)    :  2822.6
```

⚠ CAVEATS ⚠

Since VRS works by reducing number of PS executions:

- No benefit in depth/stencil only passes
- No benefit in fill rate bound scenarios
- No benefit in compute passes
- Very little benefit if average triangle size is very small (think of quad utilization)

Example: 2x2 shading rate



⚠ CAVEATS ⚠

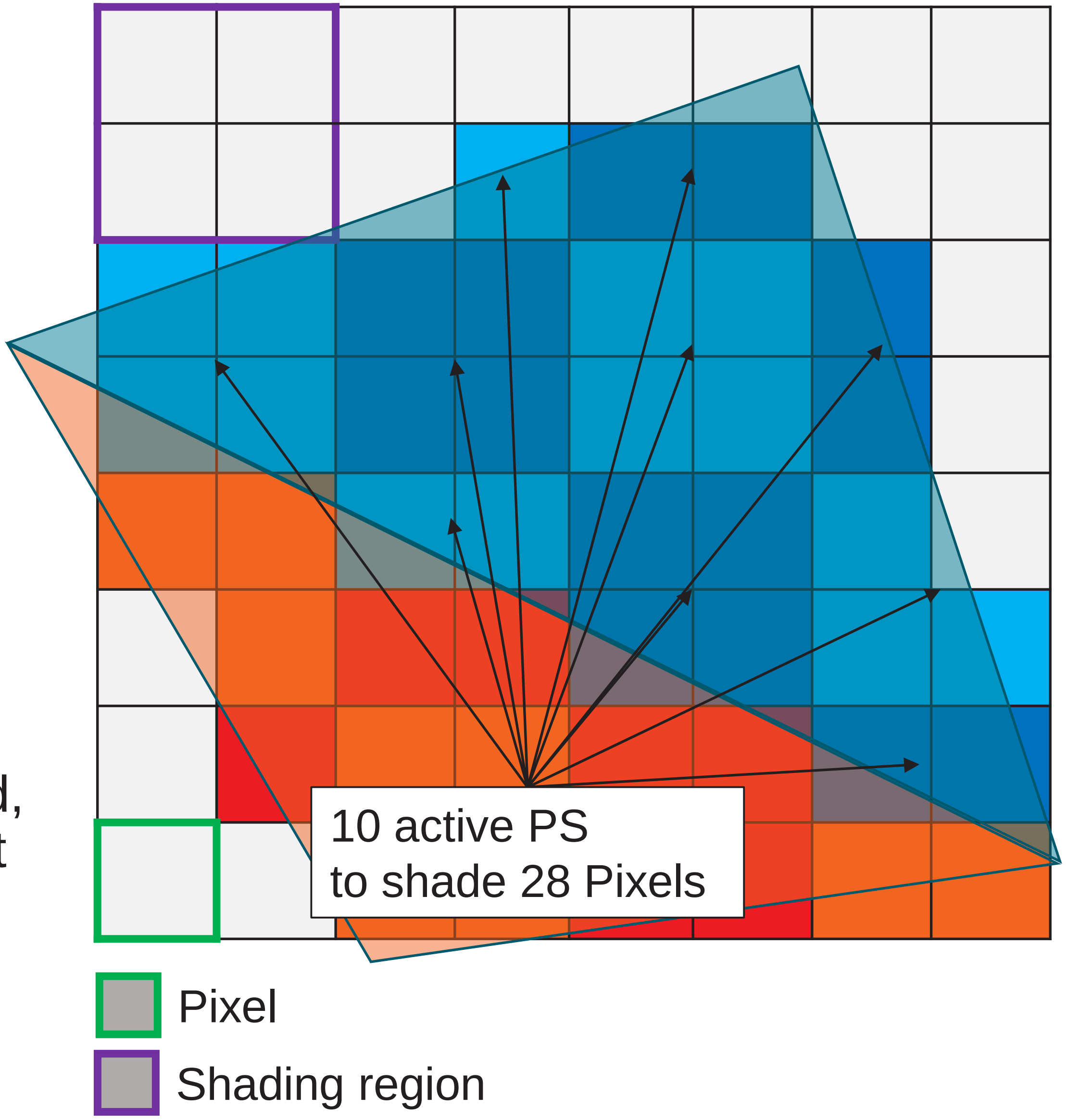
Features that cause shading rate to drop to 1x1

- Depth export
- Post-depth coverage
- Raster Order Views
- 16xMSAA

Minimize the number of times per frame the VRS image gets bound or unbound!

  - If VRS needs to get disabled for a few draw calls while the same depth buffer is being used, (e.g. to render alpha-tested geometry) the best practice is to leave the VRS image bound and disable VRS by modifying the combiners.

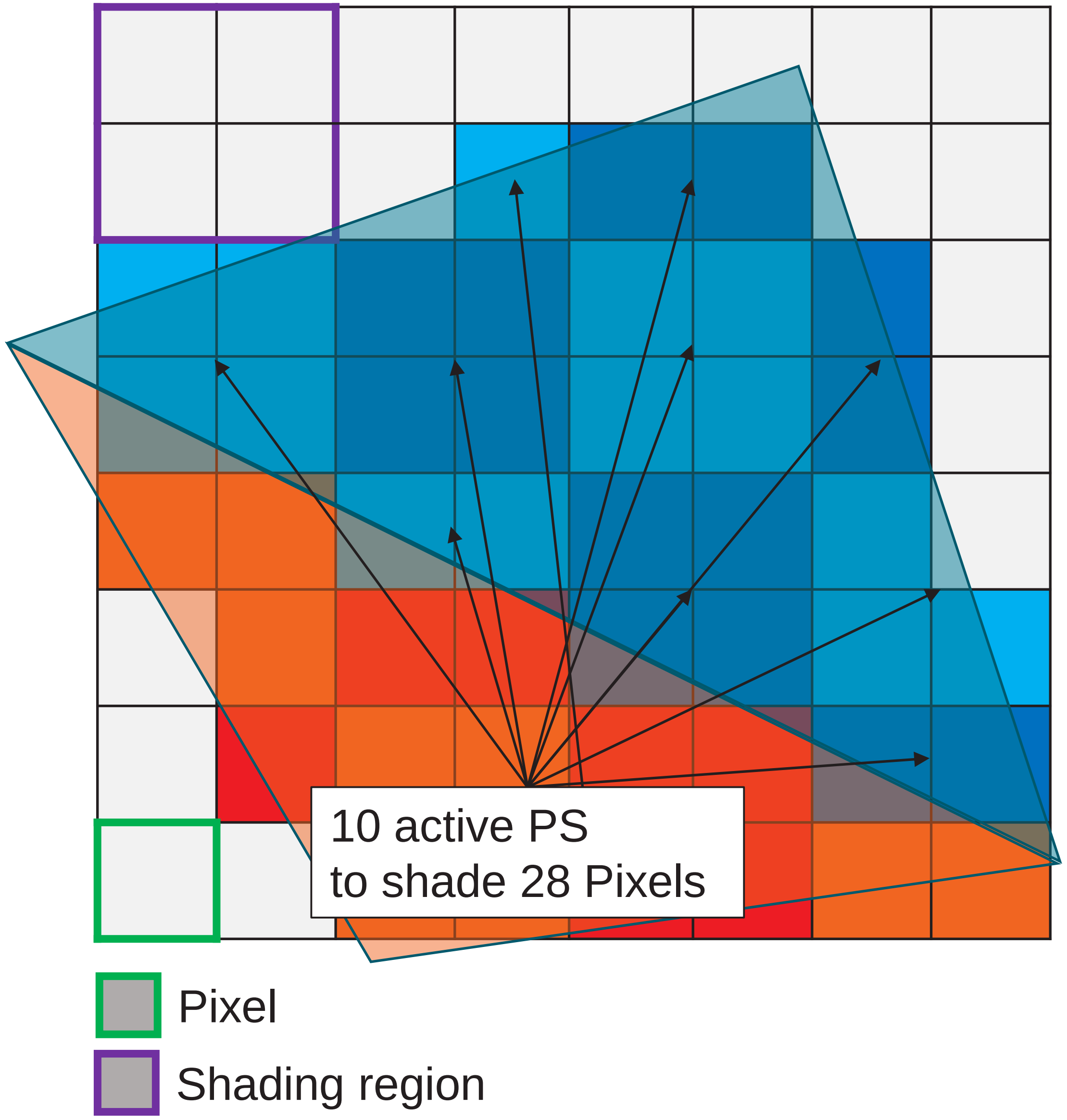
Example: 2x2 shading rate



TAKEAWAY

- Easy to integrate
  - Free performance
- VRS preserves triangle edges
  - Also depth/stencil information
- Experiments show that 2x2 shading rate is ideal for commonly used resolutions

Example: 2x2 shading rate



10 active PS
to shade 28 Pixels

LEARN MORE AT GPUOPEN.COM

AMD GPUOpen

EPYC RADEON RYZEN AMD

Contact Information:

stephan.hodes@amd.com

Twitter: @GPUOpen

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.