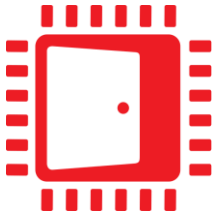


AMD GPUOpen

RADEON

AMD

AMD FidelityFX Super Resolution

Last revision: 14 July 2021

AMD

FidelityFX Super Resolution

OVERVIEW

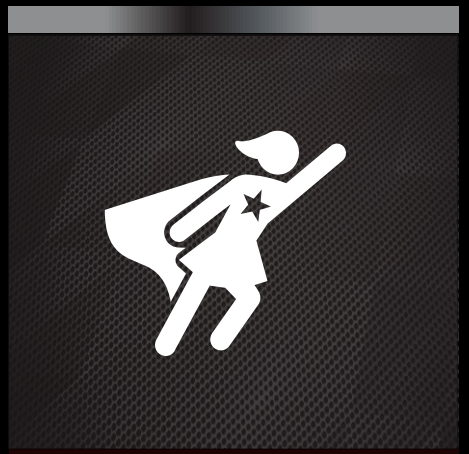
WHAT IS FIDELITYFX SUPER RESOLUTION?

AMD FidelityFX Super Resolution (FSR) is our open source high-quality solution designed to produce high resolution frames from lower resolution inputs.

It uses a collection of cutting-edge methods with a particular emphasis on creating high-quality edges, giving large performance improvements compared to rendering at native resolution directly.

FSR can enable “practical performance” for costly render operations, such as hardware ray tracing.

WHY USE AMD FIDELITYFX SUPER RESOLUTION 1.0?

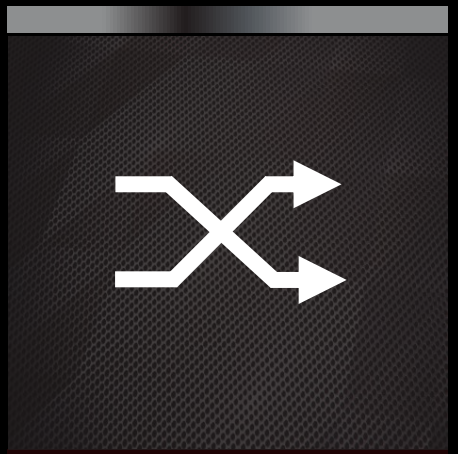


Super Resolution

Generates “super resolution” image each input frame.

Support for any area scale factor between 1X and 4X.

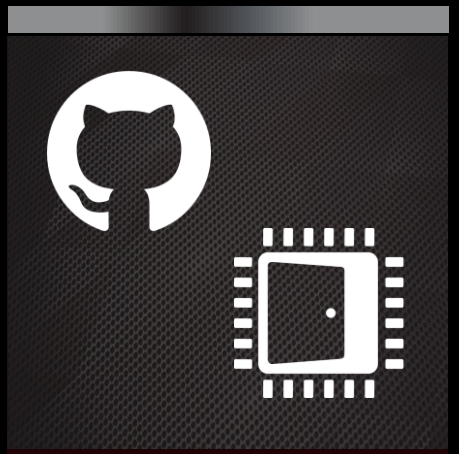
Large performance improvements over native rendering.



Cross-platform

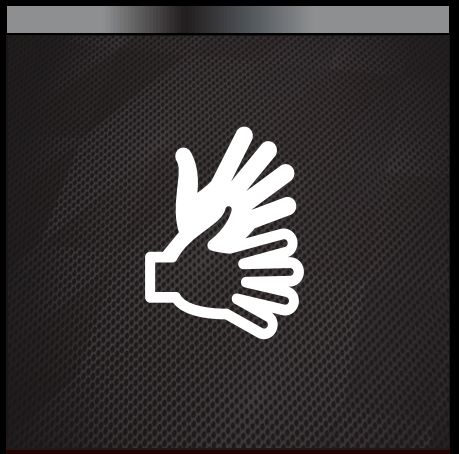
Run on a wide variety of GPUs including older architectures.

Can be ported onto multiple platforms without restriction.



Open source

Provided on GPUOpen under an MIT license.



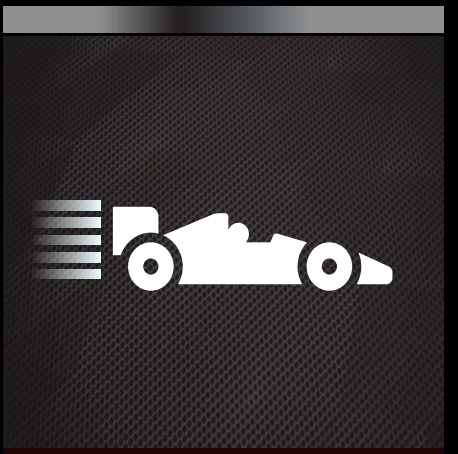
Easy to integrate

Full source provided.

Native samples and UE4 patch provided.

Also available in Xbox GDKX.

Requires no historical context.



Highly optimized

Hand-optimized for great performance across a wide variety of GPUs.

CROSS-PLATFORM

FidelityFX Super Resolution runs on a wide range of GPUs.

- No vendor-specific code path.
- Previous architectures supported – current GPUs will benefit!
- Samples provided for DirectX® 12 and Vulkan® - but FSR 1.0 shaders compile and run on DirectX® 11 too.

Port without restriction

DirectX 12 ULTIMATE

Vulkan®

OPEN SOURCE

Full source of FidelityFX Super Resolution 1.0 is hosted on GPUOpen

- Licensed under permissive MIT open source license.
- https://gpuopen.com/fsr

Code can be freely modified and redistributed subject to license.

AMD GPUOpen

EASY TO INTEGRATE

The lowest barrier to entry

- Same great experience can be expected from FidelityFX.

Integration effort estimates

Less than a day for engines supporting display resolution decoupled from render resolution

A few more days required to add support for such decoupling if not already supported

Simple compute shaders

- No additional texture/buffer input.
- No separate rendering of alpha geometry.
- No history buffer.
- No motion vectors.
- Full source provided.

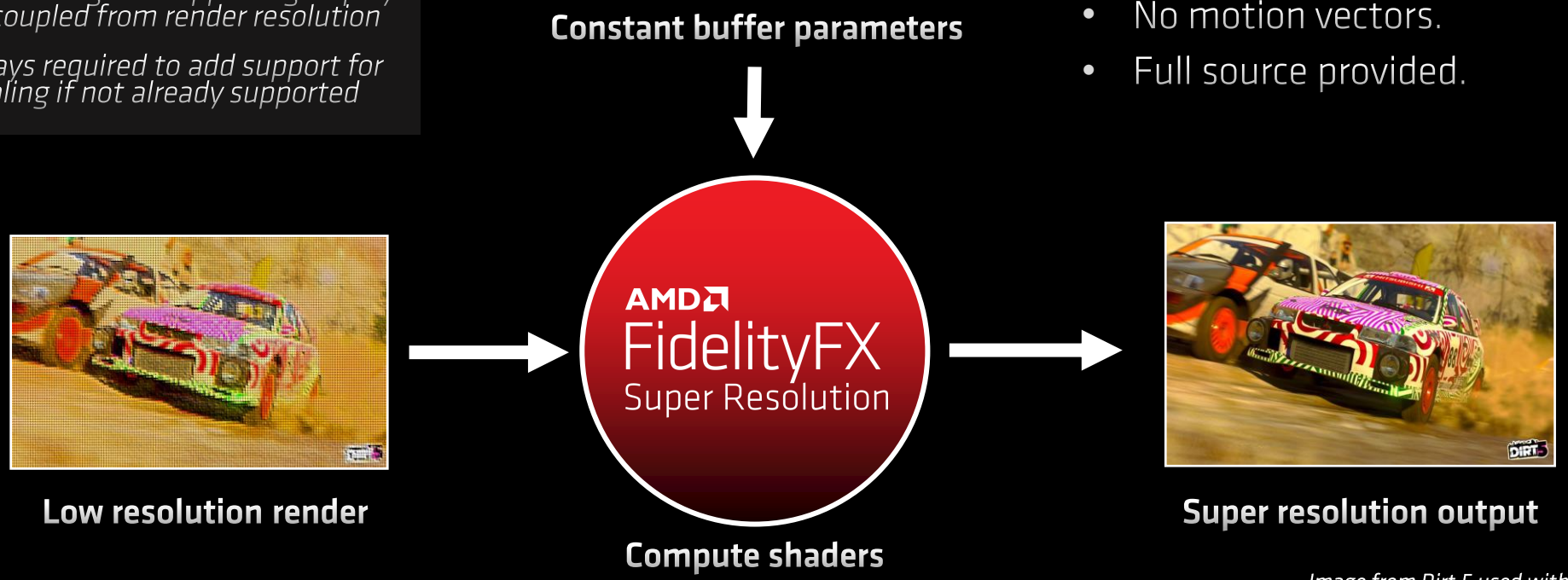


Image from Dirt 5 used with permission from Codemasters®

HIGHLY OPTIMIZED

FidelityFX Super Resolution is hand-optimized for great performance across a wide variety of GPUs.

- Does not require enthusiast-class GPUs to run!
- VALU-bound shader on most devices.

Runs in FP16 mode by default for faster performance with no visible quality impact.

- *Note:* Pre-Vega architectures that don't support FP16 packed maths may run the FP32 version of FSR 1 at better performance.
  - Please refer to detection code in the sample.

FSR PERFORMANCE OVERHEAD

FSR Target resolution	Enthusiast GPUs (AMD RADEON™ RX 6800XT, NVIDIA RTX 3080)	Performance GPUs (AMD RADEON™ RX 6700XT, NVIDIA RTX 3060 Ti)	Mainstream GPUs (AMD RADEON™ RX 5700XT, NVIDIA RTX 2060 SUPER)
4K Ultra Quality Quality Balanced Performance	0.40 ms or less	0.60 ms or less	1.0 ms or less
1440p Ultra Quality Quality Balanced Performance	0.20 ms or less	0.30 ms or less	0.50 ms or less

All numbers measured on FSR 1.0 public version
See last page for configuration details

FSR QUALITY MODES

FSR 1.0 quality mode	Description	Scale factor	Input resolution	Output resolution
Ultra Quality	**Ultra Quality** mode produces an image with quality virtually indistinguishable from native rendering. It should be selected when the highest quality is desired.	1.3x per dimension (1.69x area scale) (77% screen resolution)	1477 x 831 1970 x 1108 2646 x 1108 2954 x 1662	1920 x 1080 2560 x 1440 3440 x 1440 3840 x 2160
Quality	**Quality** mode produces a super resolution image with quality representative of native rendering, with a sizeable performance gain.	1.5x per dimension (2.25x area scale) (67% screen resolution)	1280 x 720 1706 x 960 2293 x 960 2560 x 1440	1920 x 1080 2560 x 1440 3440 x 1440 3840 x 2160
Balanced	**Balanced** mode produces a super resolution image approximating native rendering quality, with a major performance gain compared to native.	1.7x per dimension (2.89x area scale) (59% screen resolution)	1129 x 635 1506 x 847 2024 x 847 2259 x 1270	1920 x 1080 2560 x 1440 3440 x 1440 3840 x 2160
Performance	**Performance** mode visibly impacts image quality and should only be selected in situations where needing additional performance is critical.	2.0x per dimension (4x area scale) (50% screen resolution)	960 x 540 1280 x 720 1720 x 720 1920 x 1080	1920 x 1080 2560 x 1440 3440 x 1440 3840 x 2160

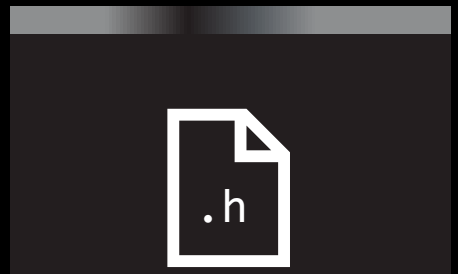
For consistency AMD recommends using these quality presets when exposing FSR in your game!

AMD

FidelityFX Super Resolution

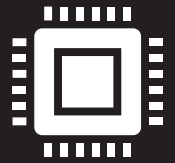
INTEGRATION GUIDE

INTEGRATION STEPS

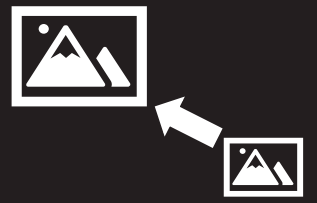


Two header files: FSR and FidelityFX portability.

Call EASU and RCAS functions.

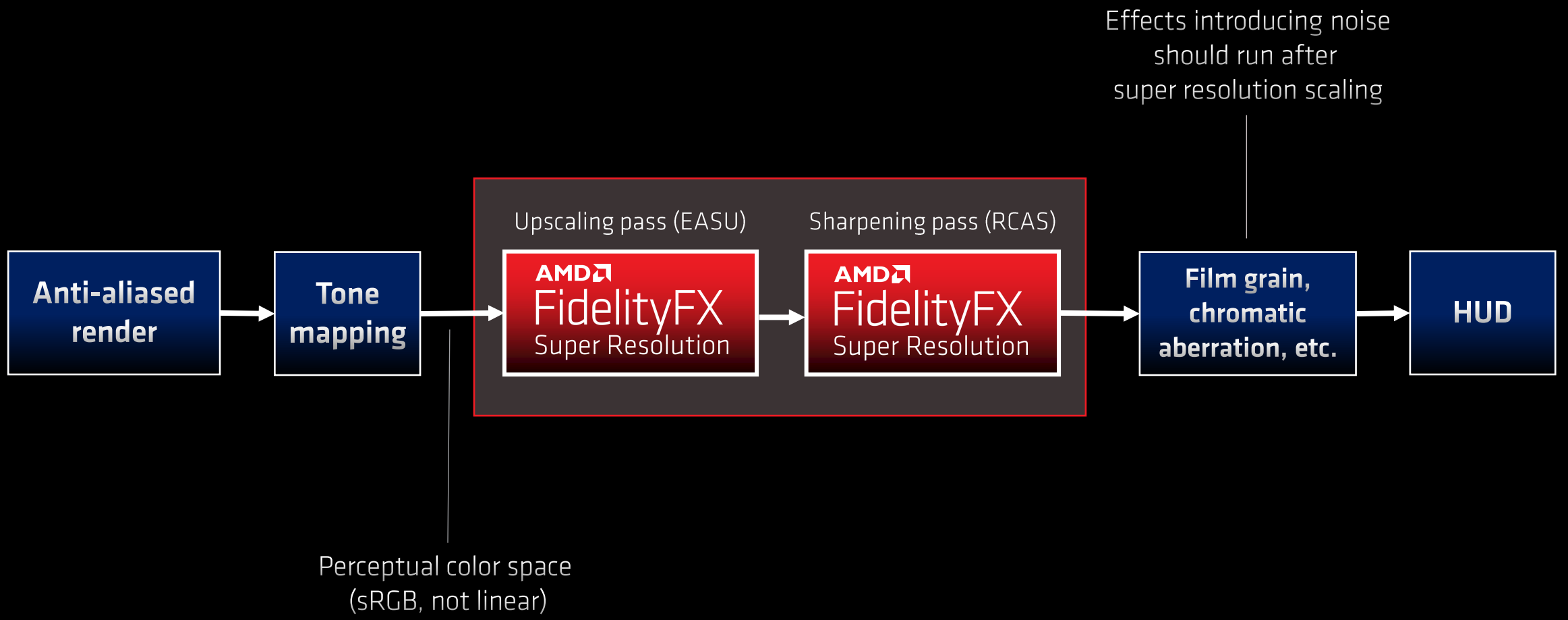


Compute shader dispatches.



Apply negative LoD bias for best results.

WHERE IN MY FRAME?



WHAT FSR NEEDS

Image should be well anti-aliased. For example: TAA, MSAA, etc.

Input image must be normalized to `[0–1]` and be in perceptual space.

- Negative input result in RCAS will output `NaN`!

Image should be generated using negative MIP bias to increase texture detail.

Image should be noise-free.

EASU SETUP FUNCTION

Include:

- `ffx_a.h`
- `ffx_fsr1.h`

Call `FsrEasuCon()` to setup the EASU constants.

- Packs the input and output resolutions into a convenient block to copy to the GPU.
- Call `FsrEasuConOffset()` if you need an input offset.

```
//copy these 4 constants into a CB block, or map directly
AU1 const0[4];
AU1 const1[4];
AU1 const2[4];
AU1 const3[4]; // Configure FSR

FsrEasuCon(const0, const1, const2, const3,
    static_cast<AF1>(RenderViewportWidth)
    , static_cast<AF1>(RenderViewportHeight) // current frame render resolution
    , static_cast<AF1>(ContainerTextureWidth)
    , static_cast<AF1>(ContainerTextureHeight) // input container resolution (for DRS)
    , static_cast<AF1>(UpscaledViewportWidth)
    , static_cast<AF1>(UpscaledViewportHeight) // upscaled-to resolution
);
```

RCAS SETUP FUNCTION

Include `ffx_a.h` and `ffx_fsr1.h`.

Call `FsrRcasCon()` to setup the RCAS constants.

- As RCAS works on full-screen images, the only input is sharpness.

```
AU1 const0[4];

FsrRcasCon(const0, Sharpness); // Sharpness in range [0-2], 0 is sharpest
```

EASU SHADER PREPARATION

EASU should be run as a compute shader.

Prepare a compute shader with the `ffx_a.h` and `ffx_fsr1.h` headers.

EASU needs 6 functions (3 for FP16(H) mode and 3 for FP32(F) mode) specified to handle texture loading:

```
AH4 FsrEasuRH(AF2 p) { AH4 res = InputTexture.GatherRed   (samLinearClamp, p, ASU2(0, 0)); return res; }
AH4 FsrEasuGH(AF2 p) { AH4 res = InputTexture.GatherGreen(samLinearClamp, p, ASU2(0, 0)); return res; }
AH4 FsrEasuBH(AF2 p) { AH4 res = InputTexture.GatherBlue  (samLinearClamp, p, ASU2(0, 0)); return res; }
```

```
AF4 FsrEasuRF(AF2 p) { AF4 res = InputTexture.GatherRed   (samLinearClamp, p, ASU2(0, 0)); return res; }
AF4 FsrEasuGF(AF2 p) { AF4 res = InputTexture.GatherGreen(samLinearClamp, p, ASU2(0, 0)); return res; }
AF4 FsrEasuBF(AF2 p) { AF4 res = InputTexture.GatherBlue  (samLinearClamp, p, ASU2(0, 0)); return res; }
```

EASU SHADER PREPARATION

To improve cache utilization, swizzle the dispatch threads using `ARmp8x8` included in `ffx_a.h`. The function assumes a linear 64 threadgroup:

```
[numthreads(64, 1, 1)]
void MainCS(uint3 LocalThreadId : SV_GroupThreadID, uint3 WorkGroupId : SV_GroupID, uint3 Dtid : SV_DispatchThreadID)
{
    // Do remapping of local xy in workgroup for a more PS-like swizzle pattern.
    AU2 gxy = ARmp8x8(LocalThreadId.x) + AU2(WorkGroupId.x << 4u, WorkGroupId.y << 4u);
```

Then call `FsrEasuH()`/`FsrEasuF()` with the swizzled coordinates, and the constants we set up before:

```
AH3 Gamma2Color = AH3(0,0,0);
FsrEasuH(Gamma2Color, gxy, Const0, Const1, Const2, Const3);
```

EASU SHADER PREPARATION

To further improve cache utilization, call the function four times in each thread.

Finally, store the results, including a viewport offset if needed:

```
FsrEasuH(Gamma2Color, gxy, Const0, Const1, Const2, Const3);
OutputTexture[gxy + ViewportOffset] = Gamma2Color;

gxy.x += 8u;
FsrEasuH(Gamma2Color, gxy, Const0, Const1, Const2, Const3);
OutputTexture[gxy + ViewportOffset] = Gamma2Color;

gxy.y += 8u;
FsrEasuH(Gamma2Color, gxy, Const0, Const1, Const2, Const3);
OutputTexture[gxy + ViewportOffset] = Gamma2Color;

gxy.x -= 8u;
FsrEasuH(Gamma2Color, gxy, Const0, Const1, Const2, Const3);
OutputTexture[gxy + ViewportOffset] = Gamma2Color;
```

RCAS SHADER PREPARATION

RCAS should be run as a compute shader.

- Prepare a compute shader with the `ffx_a.h` and `ffx_fsr1.h` headers.

RCAS needs six functions (two for each mode) specified to handle texture loading and colour-space conversion if needed:

```
AH4 FsrRcasLoadH(ASW2 p) {return InputTexture.Load(int3(p, 0));}
void FsrRcasInputH(inout AH1 r,inout AH1 g,inout AH1 b){}
```

```
AH4 FsrRcasLoadHx2(ASW2 p){return InputTexture.Load(int3(p, 0));}
void FsrRcasInputHx2(inout AH2 r,inout AH2 g,inout AH2 b){}
```

```
AF4 FsrRcasLoadF(ASU2 p){return InputTexture.Load(int3(p, 0));}
void FsrRcasInputF(inout AF1 r, inout AF1 g, inout AF1 b){}
```

RCAS SHADER PREPARATION

To improve cache utlization, swizzle the dispatch threads using `ARmp8x8` is included in `ffx_a.h`. The function assumes a linear 64 threadgroup:

```
[numthreads(64, 1, 1)]
void MainCS(uint3 LocalThreadId : SV_GroupThreadID, uint3 WorkGroupId : SV_GroupID, uint3 Dtid : SV_DispatchThreadID)
{
    // Do remapping of local xy in workgroup for a more PS-like swizzle pattern.
    AU2 gxy = ARmp8x8(LocalThreadId.x) + AU2(WorkGroupId.x << 4u, WorkGroupId.y << 4u);
```

Then call `FsrRcasH/FsrRcasH2/FsrEasuF` with the swizzled coordinates, and the constants we set up before:

```
AH3 Gamma2Color = AH3(0, 0, 0);
FsrRcasH(Gamma2Color.r, Gamma2Color.g, Gamma2Color.b, gxy, Const0);
```

RCAS SHADER PREPARATION

To further improve cache utilization, call the function four times in each thread.

Finally, store the results, including a viewport offset if needed:

```
FsrRcasH(Gamma2Color.r, Gamma2Color.g, Gamma2Color.b, gxy, Const0);
OutputTexture[gxy + ViewportOffset] = Gamma2Color;

gxy.x += 8u;
FsrRcasH(Gamma2Color.r, Gamma2Color.g, Gamma2Color.b, gxy, Const0);
OutputTexture[gxy + ViewportOffset] = Gamma2Color;

gxy.y += 8u;
FsrRcasH(Gamma2Color.r, Gamma2Color.g, Gamma2Color.b, gxy, Const0);
OutputTexture[gxy + ViewportOffset] = Gamma2Color;

gxy.x -= 8u;
FsrRcasH(Gamma2Color.r, Gamma2Color.g, Gamma2Color.b, gxy, Const0);
OutputTexture[gxy + ViewportOffset] = Gamma2Color;
```

DISPATCH THE SHADER

Since each invocation of the shader runs on four pixels, and the swizzle function assume 64 threads per thread group, divide the OUTPUT image resolution by 16 for the dispatch:

```
static const int threadGroupWorkRegionDim = 16;
int dispatchX = (UpscaledViewportWidth + (threadGroupWorkRegionDim - 1)) / threadGroupWorkRegionDim;
int dispatchY = (UpscaledViewportHeight + (threadGroupWorkRegionDim - 1)) / threadGroupWorkRegionDim;
```

That's it! You are now ready for AMD FidelityFX Super Resolution

NEGATIVE MIP BIAS

Applying a negative MIP bias will typically generate an upscaled image with better texture detail.

- Experiment for best results.

Bias to apply (adjust as desired):

- MIP bias = -log2(DisplayResolution/SourceResolution)

Provided FSR 1.0 samples set negative MIP bias slider to suitable defaults based on quality mode selected.

FSR 1.0 quality mode	Scale factor	MIP bias
Ultra Quality	1.3x per dimension	**-0.38**
Quality	1.5x per dimension	**-0.58**
Balanced	1.7x per dimension	**-0.79**
Performance	2.0x per dimension	**-1.0**

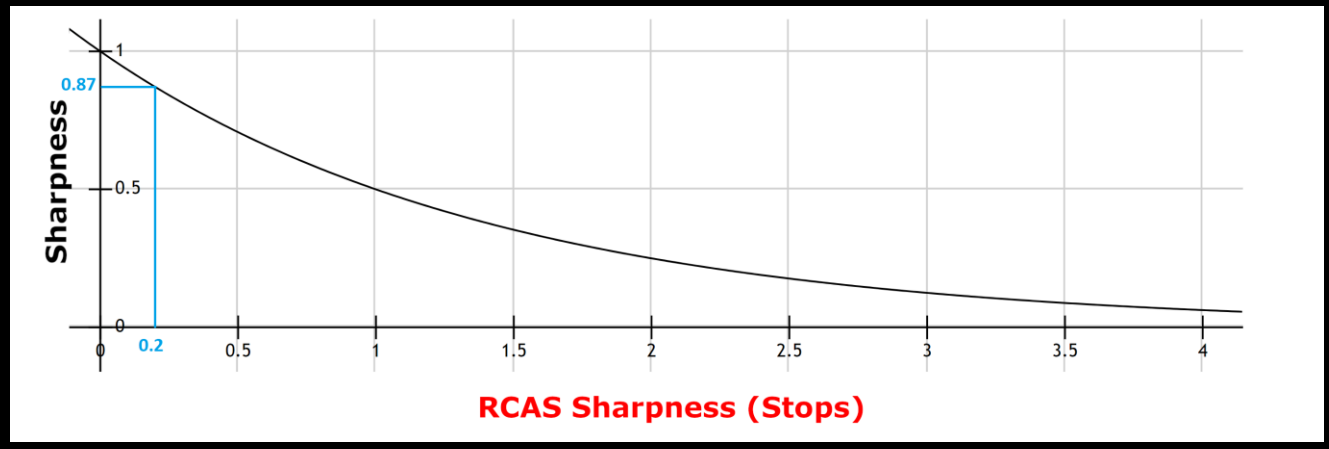
RCAS SHARPENING AMOUNT

RCAS sharpness parameter is specified in ‘stops’.

- Sharpness halves with every whole number.
- Goes from 0.0 (sharpest) to about 2.0.
- Values above 2.0 won’t make a visible difference.

A good recommended default value for RCAS Include is 0.2.

- Adjust as needed based on preferences.



HDR DISPLAY SUPPORT

Input needs to be encoded as sRGB or Gamma 2.0 UNORM.

- PQ or HLG are not suitable encodings for input.

Avoid banding in input as FSR will highlight it further.

- Optimally use dithering to keep input as `R10G10B10A2_UNORM`.

Multiple utilities included in `ffx_fsr1.h`.

- Temporal Energy Preserving Dither – helps avoid banding in input.
- Simple Reversible Tone-Mapper – normalizes linear HDR.
- Quick approximate reversible PQ to Gamma 2.0 conversion.

FULL PRECISION FALLBACK

FSR 1.0 was designed to take advantage of half precision (FP16) for best performance.

- A slower full precision (FP32) fallback is included to run on older hardware.

Direct3D apps:

- Query `D3D[11/12]_FEATURE_DATA_SHADER_MIN_PRECISION_SUPPORT`.
- If `D3D[11/12]_SHADER_MIN_PRECISION_16_BIT` is not set, fallback to FP32.

Vulkan apps:

- If `VkPhysicalDeviceFloat16Int8FeaturesKHR f.shaderFloat16` is not set, fallback to FP32.
- If `VkPhysicalDevice16BitStorageFeatures f.storageBuffer16BitAccess` is not set, fallback to FP32.

WHEN TO RUN THE FP32 FALLBACK

GPU	FSR 1.0 Precision
AMD Radeon™ RX 6000 Series	FP16
AMD Radeon™ RX 5000 Series	FP16
AMD Radeon™ RX Vega Series	FP16
AMD Radeon™ RX 400 Series	Fallback to FP32
AMD Radeon™ RX 500 Series	Fallback to FP32

GPU	FSR 1.0 Precision
NVIDIA GeForce 30 Series	FP16 *
NVIDIA GeForce 20 Series	FP16
NVIDIA GeForce 10 Series	FP16
NVIDIA GeForce 900 Series	Fallback to FP32
Intel UHD Graphics	FP16

** You may encounter image corruption issues on NVIDIA RTX 3000 series when FP16 FSR shaders are used with DirectX 11. If this issue manifests itself then the FP32 version of FSR should be used on these GPUs until NVIDIA issues a driver fix.* ***This issue does not affect DirectX 12 or Vulkan.***

AMD

FidelityFX Super Resolution

USER INTERFACE GUIDE

FSR UI GUIDELINES - NAMING

Recommended UI names for FidelityFX Super Resolution 1.0:

- “AMD FidelityFX Super Resolution 1.0”
- “FidelityFX Super Resolution 1.0”
- “AMD FSR 1.0”

Recommended UI description for FidelityFX Super Resolution 1.0:

- “AMD FidelityFX Super Resolution 1.0 is a cutting edge super-optimized spatial upscaling technology that produces impressive image quality at fast framerates.”

Recommended UI descriptions for FSR 1.0 quality presets:

- Please use quality preset descriptions from the table on slide 10.

FSR UI GUIDELINES - UI EXAMPLE

Please list FSR quality presets from highest to lowest quality.

- An example is shown below:

FSR UI GUIDELINES – ANTI-ALIASING

FSR 1.0 applied on a non anti-aliased input image will likely return poor results.

- Hard edges will be detected as such and exacerbated in the upscaled image!

To avoid this situation please implement either of the following options:

- Enabling FSR automatically enables the highest-quality AA option such as TAA, MSAA 8x etc.
- The FSR 1.0 option becomes grayed out if the game's AA option is set to "Off" or "None".

FSR UI GUIDELINES – SHARPENING

FSR comes with its own sharpening pass.

If your game already supports a sharpening option in the UI please gray out sharpening UI when FSR is enabled.

- This is to avoid a clash with the RCAS sharpening feature of FSR.

Please do not explicitly mention "RCAS" in UI – it is part of the FSR 1.0 solution!

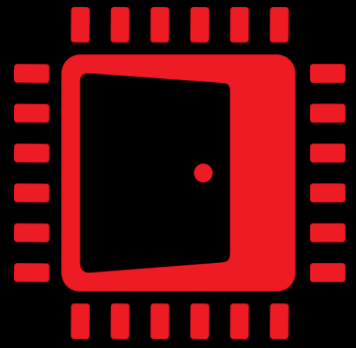
Please do not use FSR RCAS without EASU upscaling!

- If sharpening is desired without FSR upscaling, then FidelityFX CAS is recommended instead.

TALK TO US!

Please contact your AMD representative if you integrate FSR in your game!

We may be able to help promote your game on social channels



AMD GPUOpen

EPYC

RADEON

RYZEN

AMD

CONFIGURATION FOR PERFORMANCE TESTING

Motherboard: MSI X570-A Pro

CPU: AMD Ryzen 9 5900X @ 3.70 Ghz

Chipset: Ryzen SOC

System RAM: 16GB G.Skill DDR4-3600 CL16-16-16-36

OS: Windows 10 Pro 64-bit (OS Build 19042.928)

AMD driver version: 21.20-210624n

NVIDIA driver version: 471.11

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. GD-18.

AMD FidelityFX Super Resolution is supported on the following AMD products: AMD Radeon™ RX 6000, RX 5000, RX 500, RX Vega series graphics cards, RX 480, RX 470, RX 460, and all AMD Ryzen™ Processors with Radeon™ Graphics if the minimum requirements of the game are met. AMD does not provide technical or warranty support for AMD FidelityFX Super Resolution enablement on other vendor's graphics cards. GD-187

Vulkan and the Vulkan logo are registered trademarks of the Khronos Group Inc. PCIe and PCI Express are registered trademarks of the PCI-SIG Corporation. DirectX is a registered trademark of Microsoft Corporation in the US and other jurisdictions.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.