# Fast Approximate Anti Aliasing (FXAA)

## Description

FXAA algorithm is novel antialiasing algorithm that runs in single post process pass, and requires only scene color buffer input. It is characterized by fast execution time, easy integration process and good antialiasing quality despite the fact that it produce slight picture blurring. Algorithm doesn't require alpha blending stage to be turned on, and have a broad range of configuration parameters for better tuning performance/quality relation. It is developed by Timothy Lottes who works in Nvidia, and detailed description of algorithm can be found in official FXAA paper (http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf) and in author personal blog (http://timothylottes.blogspot.com/search?q=fxaa)

Pixel aliasing is detected using several methods:

1. **Local contrast check.** This check uses the pixel and its North, South, East, and West neighbors. If the difference in local maximum and minimum luminance (contrast) is lower than a threshold proportional to the maximum local luminance, then the shader early exits (no visible aliasing).
2. **Sub-pixel aliasing.** Pixel contrast is estimated as the absolute difference in pixel luminance from a lowpass luminance (computed as the average of the North, South, East and West neighbors). The ratio of pixel contrast to local contrast is used to detect sub-pixel aliasing.
3. **Vertical/Horisontal edge test.** This test is performed using Sobel-like filter on single pixel lines that pass through center pixel.
4. **End-of-edge search.** To define potential edge direction, algorithm pair center pixel with highest contrast neighbor. Than limited number of search steps are performed in positive and negative directions. Search stops when maximum number of samples is tested, or average luminance is changed significantly to signify end-of-edge.

Every listed step is highly configurable via preprocessor definitions. Entire effect reside in Data\Shaders\DX9_P1\Fxaa.h file. FXAA algorithm can be turned of and on using **r_fxaa** command variable.

## Associated classes and structures

## class PFX_FXAA

FXAA algorithm runs in single full screen pass, but require luminance data in alpha channel of input image. This is not always convenient to fill alpha channel with such data, so there is an option to use green channel of input color image as luminance value. Described behavior can be turned on and off using FXAA_GREEN_AS_LUMA preprocessor definition. Using dedicated precomputed luminance will lead to slightly better result, but in additional cost.

## class PFX_FXAA_LumPass

To produce best possible result, additional pass that computes image luminance value is implemented. This post process effect should run before main FXAA pass, and PFX_FXAA need to get output of PFX_FXAA_LumPass as input. This effect is very simple, lightweight and requires no setup.

**Usage example**

To add FXAA to post process pipeline, several simple steps need to be performed. Firstly create instances of PFX_FXAA and PFX_FXAA_LumPass classes:

```
PFX_FXAA_LumPass gPFX_FXAA_LumPass;
PFX_FXAA gPFX_FXAA;
```

Add this two effects in sequential order into post process pipeline (consult post process system documentation for detailed description):

```
g_pPostFXChief->AddFX(gPFX_FXAA_LumPass, PostFXChief::RTT_TEMP0,
PostFXChief::RTT_PINGPONG_LAST);
g_pPostFXChief->AddFX(gPFX_FXAA, PostFXChief::RTT_PINGPONG_NEXT,
PostFXChief::RTT_TEMP0);
g_pPostFXChief->AddSwapBuffers();
```

# FXAA Shadows

## Description

FXAA algorithm can be adapted to smooth aliasing in any space if smoothed entity can provide analog of luminance information. This idea was used to antialias shadow edges using core of FXAA algorithm. Because entire FXAA method is rather heavy, we use only simplified local contrast check plus very limited end-of-edge search to smooth aliased edges. One important thing must be noted. Shadow FXAA algorithm requires percentage closer filtering to be enabled for shadow samples. This leads to big count of shadow map samples (and compares) if shader PCF implementation is used. Shadow FXAA requires 6 PCF samples, which leads to 24 shadow map samples (4 fetches for each PCF sample). If hardware PCF (supported by most hardware vendors) is enabled, algorithm receives big speedup through reduced sample count.

Shadow FXAA algorithm can be enabled by using DS_AccumShadows_ps.hls compiled with

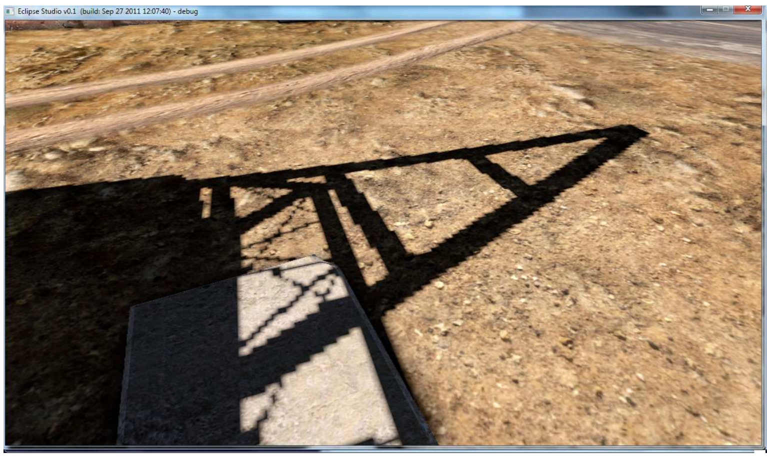
```
#define FXAA_SHADOW_BLUR 1
```

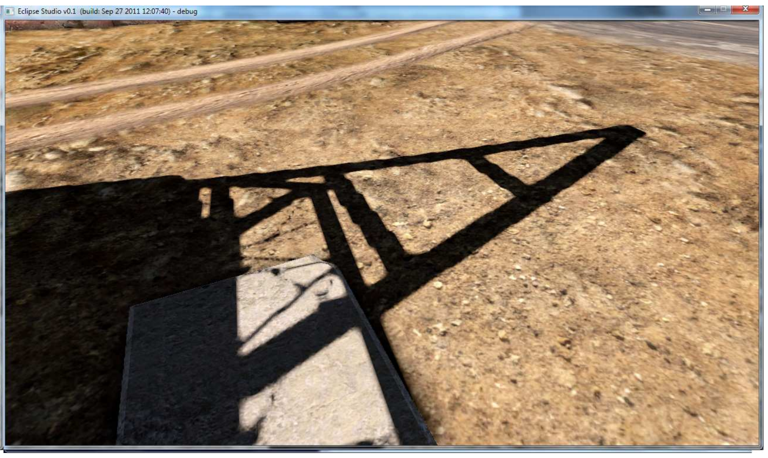
preprocessor definition. Hardware PCF is enabled if same shader was compiled with

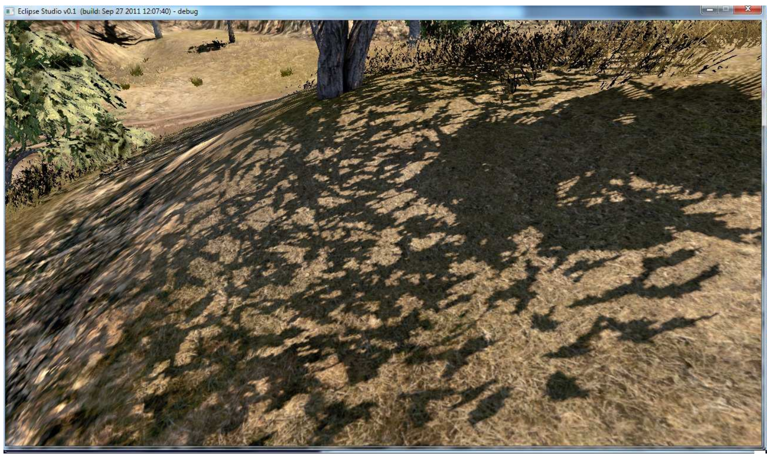
```
#define HW_SHADOW_MAPS 1
```

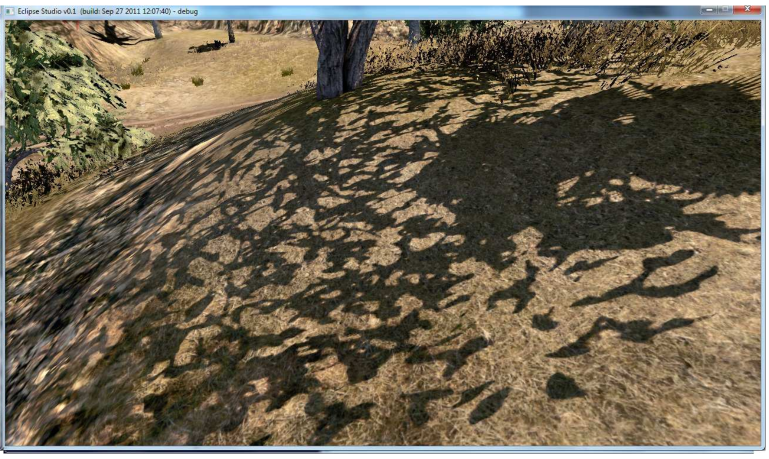
## Examples

This section presents comparison images, to show FXAA shadow algorithm results. Left figure is ordinary one tap PCF algorithm, right figure is shadow FXAA.


Eclipse Studio v0.1 (build: Sep 27 2011 12:07:40) - debug


Eclipse Studio v0.1 (build: Sep 27 2011 12:07:40) - debug


Eclipse Studio v0.1 (build: Sep 27 2011 12:07:40) - debug


Eclipse Studio v0.1 (build: Sep 27 2011 12:07:40) - debug