A Directionally Adaptive Edge Anti-Aliasing Filter

Konstantine Iourcha

Jason Yang

Andrew Pomianowski

High Performance Graphics | August 2, 2009

AMD

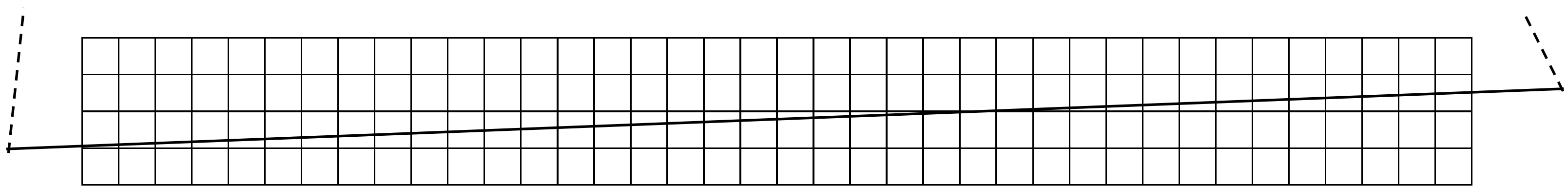
The future is fusion

Motivation

- Can we use the GPU’s shader processing power and flexibility for better edge anti-aliasing (AA)?
  - Goal
    - Improve primitive edge appearance (vs. “standard” MSAA processing) using the same number of samples and better software filtering algorithms
  - Benefits
    - Rendering time (sans post-processing) and memory footprint stay the same
    - Software filters can be easily modified if needed
  - Constraints
    - Needs to run in real-time on the same GPU as rendering
    - Needs to use existing HW features (*e.g.*, MSAA data)

Motivation

- Can we use the GPU's shader processing power and flexibility for better edge anti-aliasing (AA)?



Motivation

- Can we use the GPU's shader processing power and flexibility for better edge anti-aliasing (AA)?

No AA

Motivation

- Can we use the GPU's shader processing power and flexibility for better edge anti-aliasing (AA)?

8x MSAA

Motivation

- Can we use the GPU's shader processing power and flexibility for better edge anti-aliasing (AA)?

Our New Filter

Contribution

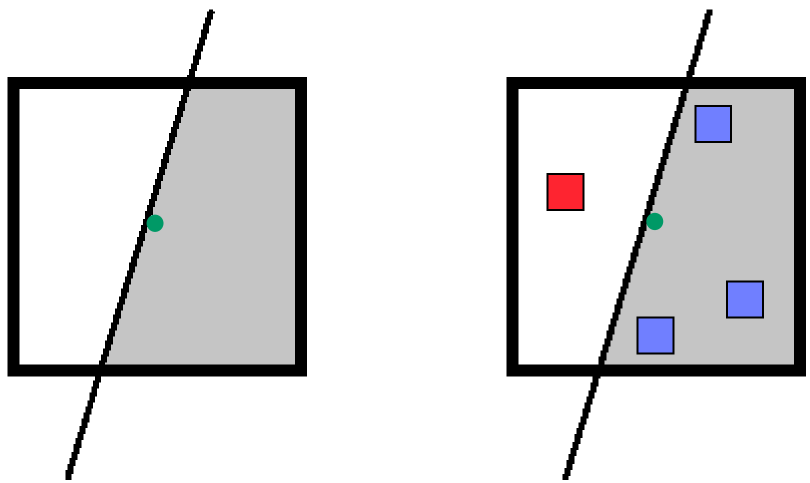
- Developed an approach to edge AA using non-linear filtering with significant quality improvements over “standard” linear filters
- Developed a filtering algorithm feasible for real-time rendering on the GPU by taking advantage of existing MSAA data
- Implemented algorithms on the GPU

Outline

- MSAA overview
- Prior and parallel work
- Algorithm overview
- Implementation and results
- Future work

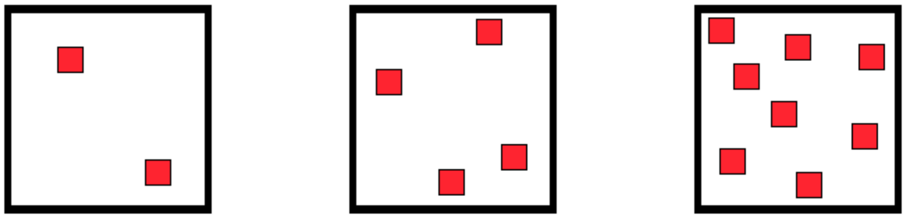
Multisample Anti-Aliasing (MSAA) [Akeley 1993]

- Estimate primitive pixel coverage by testing at sample points
- Calculate a single color value per pixel per primitive (usually at the center of the pixel or at the centroid of the covered samples) and assign it to all covered samples



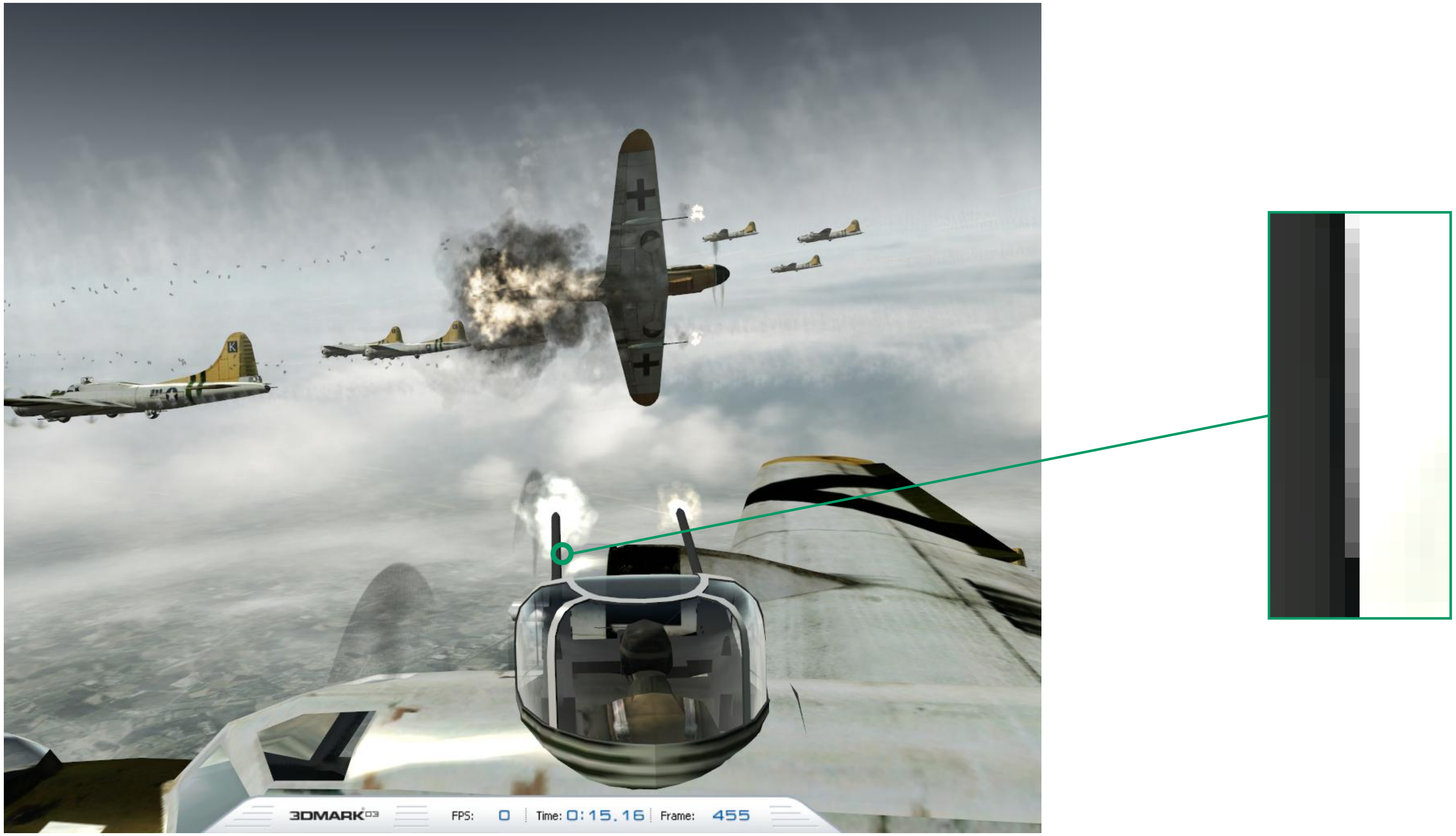
MSAA Overview

- Speeds up rendering by not calculating values for each sample separately
- Non-uniform sampling grid is used to improve pixel coverage estimate, see also [Laine and Aila 2006]



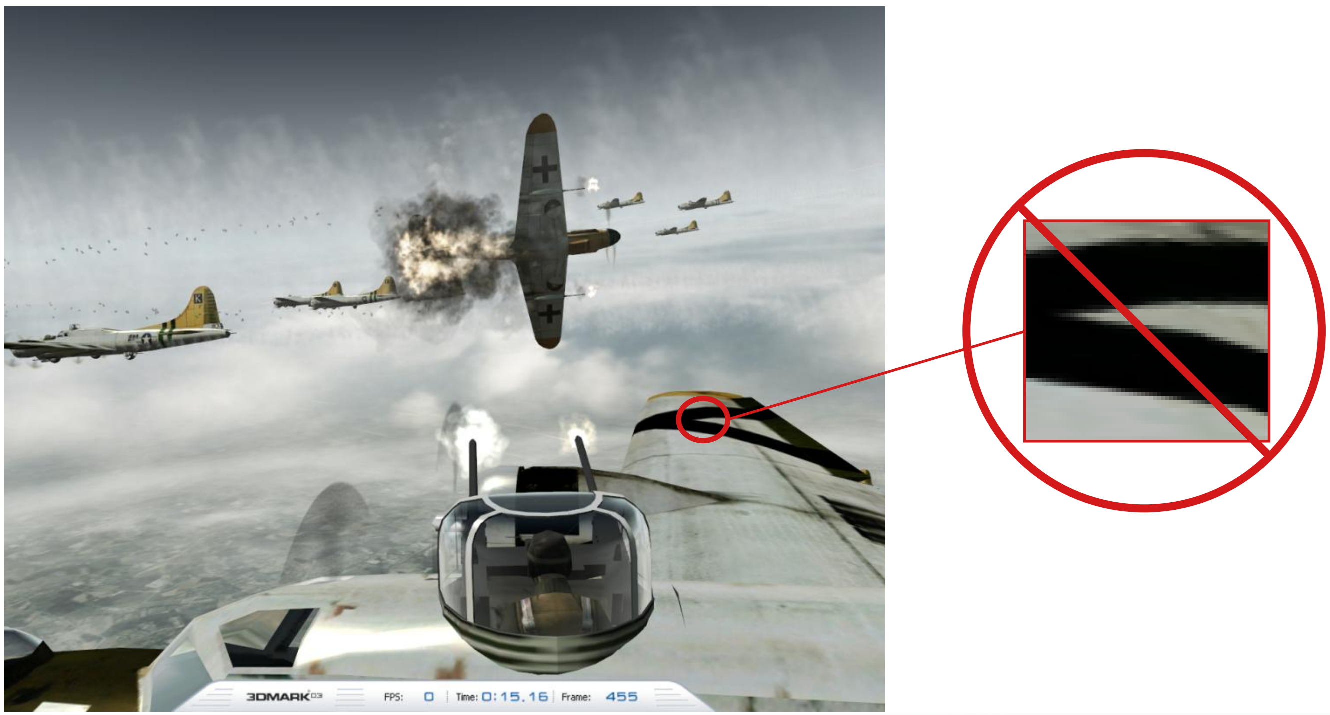
MSAA Overview

- Primary goal is anti-aliasing of primitive and Z edges



MSAA Overview

- Texture resampling performed elsewhere (mipmapping)
  - High frequency texture details are lost in the MSAA color buffer and cannot be easily post-processed further



Prior Work - AA

- Edge anti-aliasing research has more then 30 years history (see references in the paper)
  - Most algorithms use simple integration over pixel area
- More relevant are
  - [Deering and Naegle 2002]
    - Uses a wider kernel resolve filter in HW, but still linear
    - Some blurring across the edge
  - [Lau 2003]
    - Post-processing with non-linear filter, 5x5 pixel area
    - Does not use sub-pixel sample information
    - Table grows exponentially with the size of the sampling neighborhood; might be expensive on the GPU as it is

Parallel Work – Image Upscaling

- Huge amount of work, but little can be adopted
- Similar, but not the same problem
  - 2x upsampling has ¼ sample/target pixel density
  - AA has 8 sample/pixel density
- Common issues adapting upsampling algorithms
  - Good processing of near 45° edges is most important, (near) horizontal/vertical is less important
    - It is exactly the opposite in the AA situation
  - Not the best at handling high contrast edges (modulation/blurring) (see also [Su and Willis 2004])
  - Designed explicitly around Cartesian grids
  - Naïve scaling to 72+ sample area often is not feasible

Parallel Work – Image Upscaling

- More relevant to our work
  - Isolines/ isophote based [Wang and Ward 2007]
    - Too complex, difficult to implement for our purposes
  - Data dependant triangulation [Yu, Morse, and Sederberg 2001]
    - Fairly universal
    - Very difficult to implement on GPU
    - Requires triangulation structure which supports flips
    - Indirect inspiration for our work

Design Approach for the New AA Filter

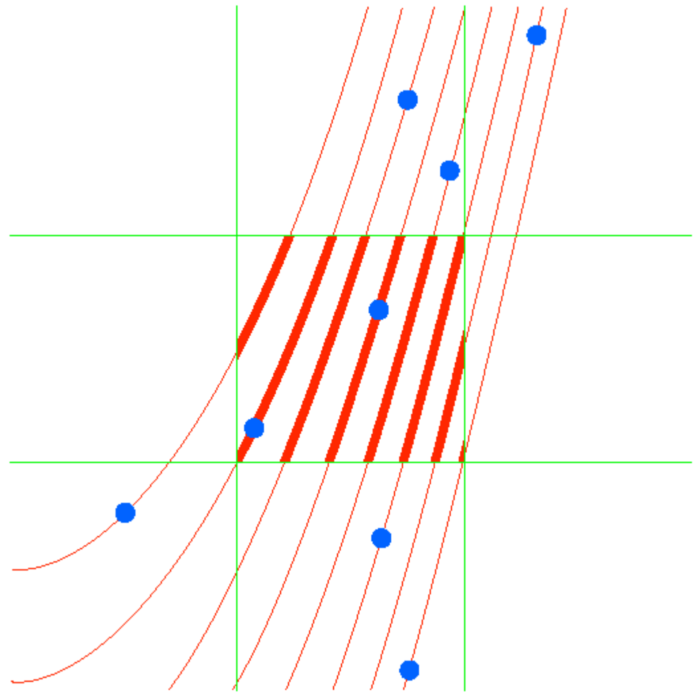
- Use intensity isolines rather than intensity itself
- Do not evaluate isolines explicitly
  - Use interpolants or approximating functions instead; simple ones can work
- Do not evaluate edge subpixel position explicitly
- Avoid complex structures
- Avoid big tables

Anti-Aliasing Model Revisited

- If we consider a single channel (gray-scale) image as an intensity function over continuous [x, y]
  - Pixel value can be computed by integrating the intensity over the pixel area [Catmull 1978]
  - Minimizes RMS error when a box filter (LCD display) is used for reconstruction
  - Some Moiré patterns can be visible due to high frequency leakage; they can be reduced by a low pass prefilter
- Standard MSAA computes an estimate of this integration over the pixel
- We want to improve this estimate

Integration Approach and Isolines

- If we know isolines (isophotes) of the image function
  - Sample values outside of the pixel can be used
  - Weight them by the length of the corresponding isoline segment in the pixel and add all of them



Integration Approach and Isolines

- If we know isolines (isophotes) of the image function
  - Sample values outside of the pixel can be used
  - Weight them by the length of the corresponding isoline segment in the pixel and add all of them
  - This works regardless of an edge presence (assuming “more or less” uniform sampled isoline distribution)
  - Will work for complex isoline topologies (unless isolines are inside a single pixel), but we do not handle this at the moment

Isoline Evaluation

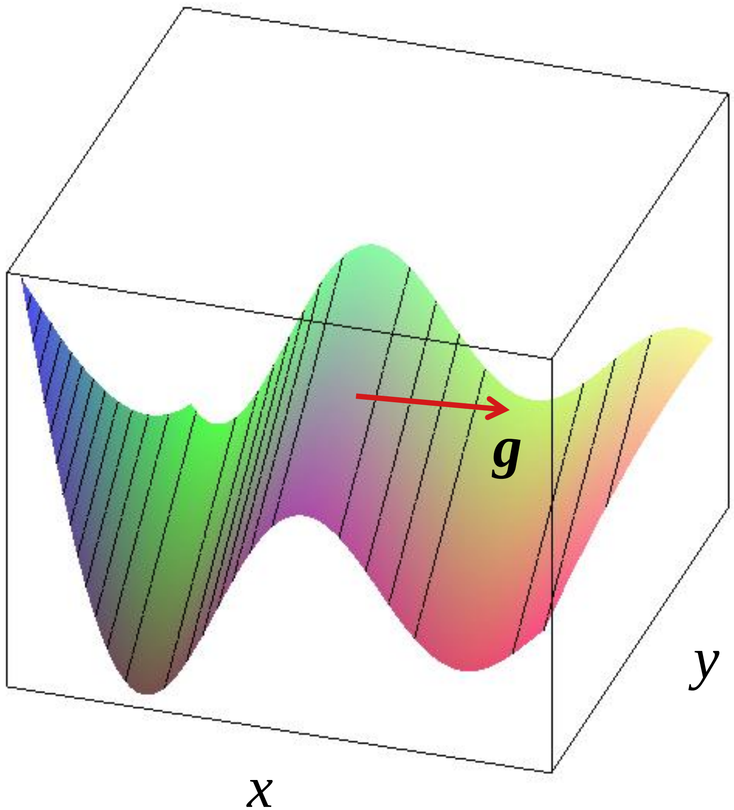
- Need to find isolines
  - But only in the case of low isoline curvature (as this is the case near the actual primitive edges)
  - Then we can model them as a straight lines
  - And try to approximate our function with an extrusion surface

$$f(v) \approx \tilde{f}(\langle g, v \rangle)$$

    $\boldsymbol{v}$ = [x, y], $\boldsymbol{g}$ is a fixed vector collinear to a local gradient

  - The biggest simplification would be to use a plane as an approximation
    - Still will estimate the local gradient

Isoline Evaluation



Linear Fitting In The Case of Three Channels ("Gradient" Estimate)

- Use linear approximation
- Solve least squares problem to minimize

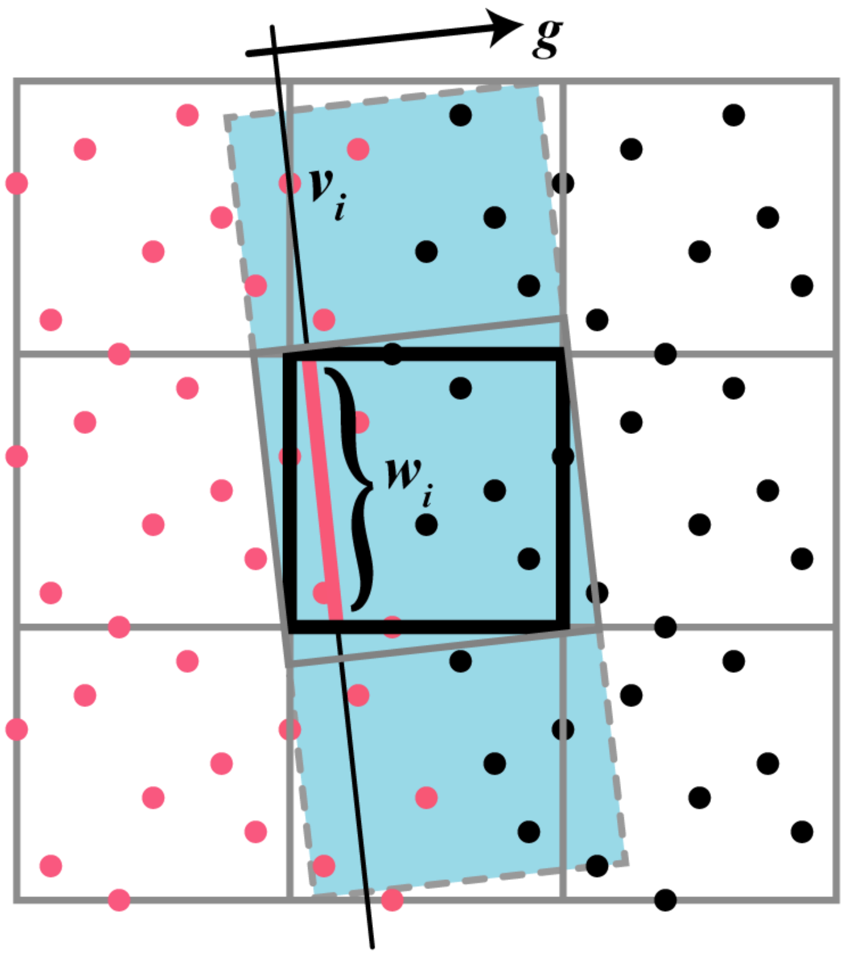
$$F = \sum_{i \in I} \left\| \left(C_1 \bullet \langle g, v_i \rangle + C_0 \right) - f(v_i) \right\|^2$$

over RGB vectors C_0, C_1, and 2D vector $\boldsymbol{g}$

- This approach works as locally R,G,B correlate well (compare with S3TC); C_1 is maximum correlation vector in RGB space and C_0 is mean color value
- Better than using just the luma channel as chromatic edges are taken into account too
- Note: $\boldsymbol{g}$ is not a really a gradient in 3 channel case

Filter Computation (Integration)

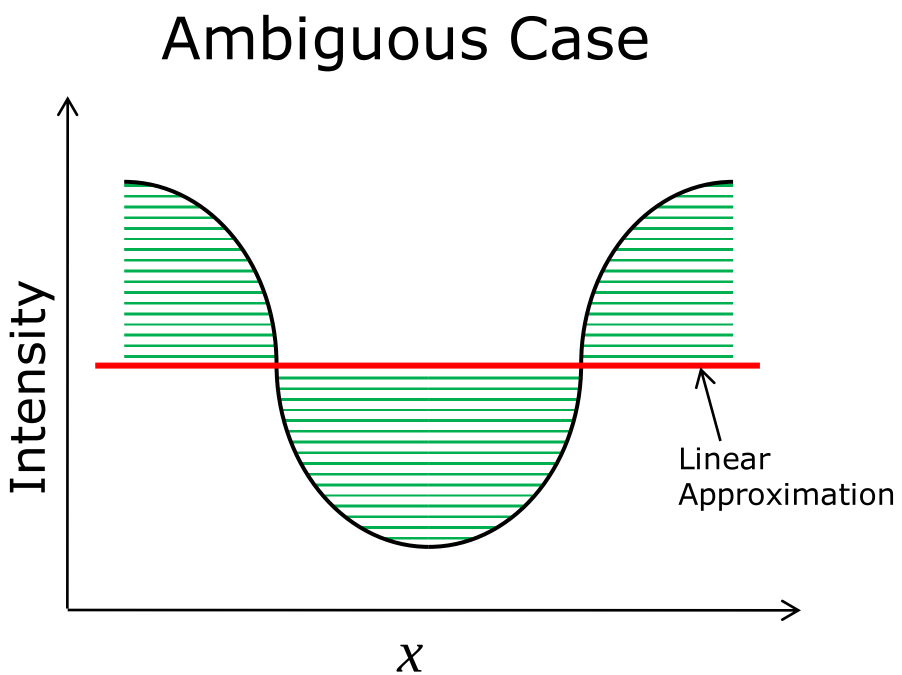
- Construct a square around the pixel, with two sides orthogonal to $\boldsymbol{g}$
- Extend in the direction orthogonal to $\boldsymbol{g}$ until it meets the 3x3 pixel boundary
- The inscribed length of the line passing though sample $\boldsymbol{v}_i$ and orthogonal to $\boldsymbol{g}$ is its weight w_i
- Calculate the weighted sum of all samples in the rectangle

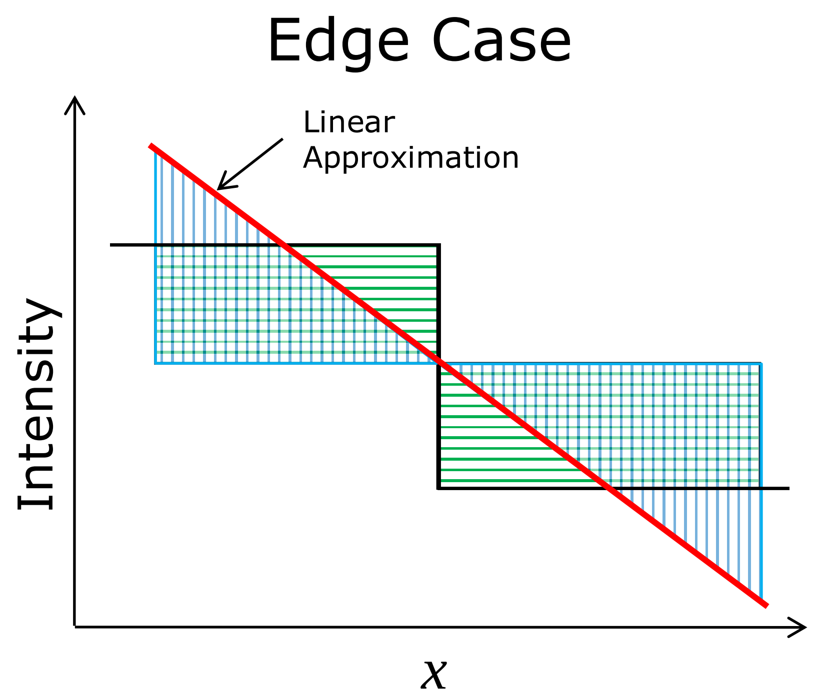


Thresholding

- Need to exclude g =0 cases
- Need to exclude pixels with high isoline curvature from the filtering above and use “standard” resolve
  - Avoids excessive corner smoothing
  - Reduces processing time
  - Threshold value is application dependent
- Based on how well the variance of the original function is preserved by the linear function approximation in the 3x3 pixel region
  - Threshold cannot be “too tight”, as it will reject step functions (edges)

Thresholding





Thresholding and Step Functions (Edges)

- The fitting and thresholding using a step function as an approximation is possible
  - Would provide more reliable edge detection
  - Would cost much more to implement because different positions of the step function discontinuity line among samples need to be tested separately

Masking

- Similar purpose as thresholding
- If a pixel has two or more samples with different colors it is an "edge" pixel, otherwise it is "non-edge"
  - "Internal" primitive pixels will have samples all of the same color due to MSAA (some edges can generate such pixels too; we ignore this)

Masking

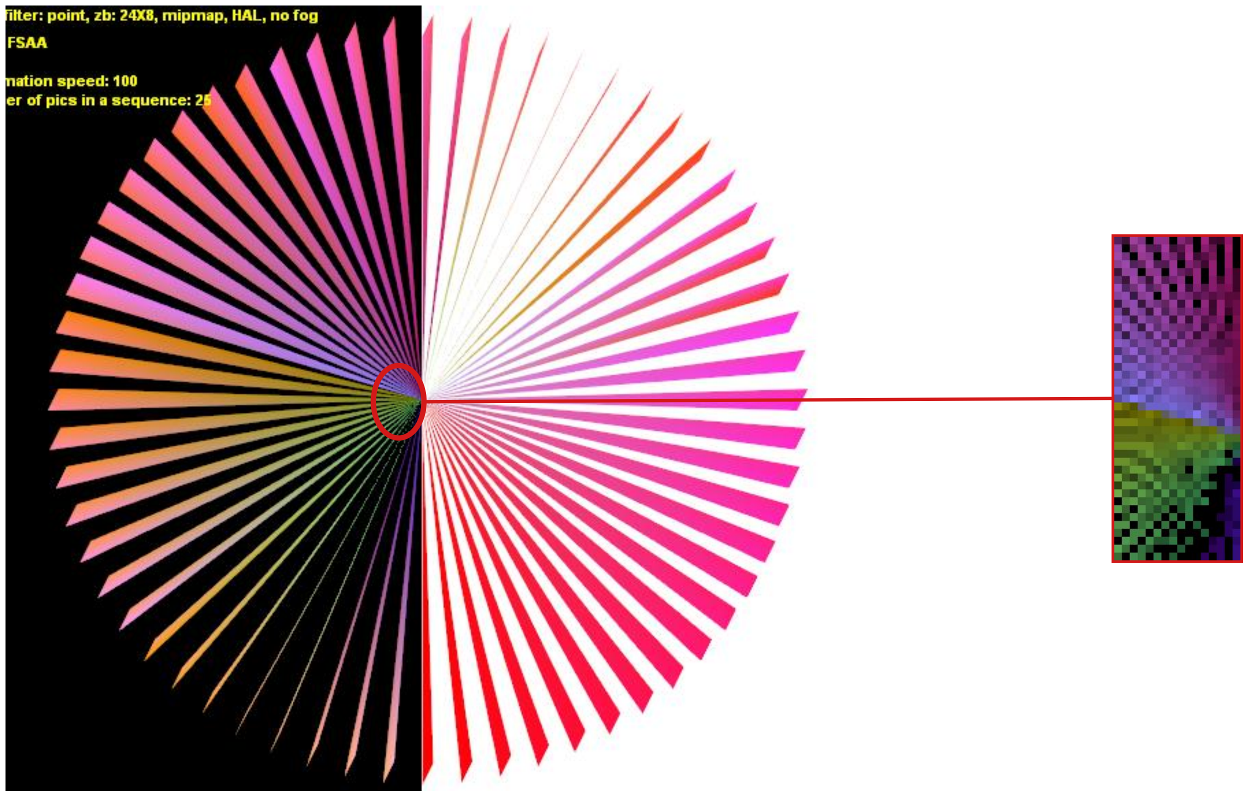
- We use "edge" (X) / "non-edge" (O) 3x3 pixel patterns to eliminate cases where filtering should not be performed. (Compare with [Lau 2003])
  - "Accept" patterns:

$$\begin{pmatrix} & X & \\ & \boxed{X} & O \\ & X & O \end{pmatrix} \quad \begin{pmatrix} & & X \\ X & \boxed{X} & \\ O & O & \end{pmatrix}$$

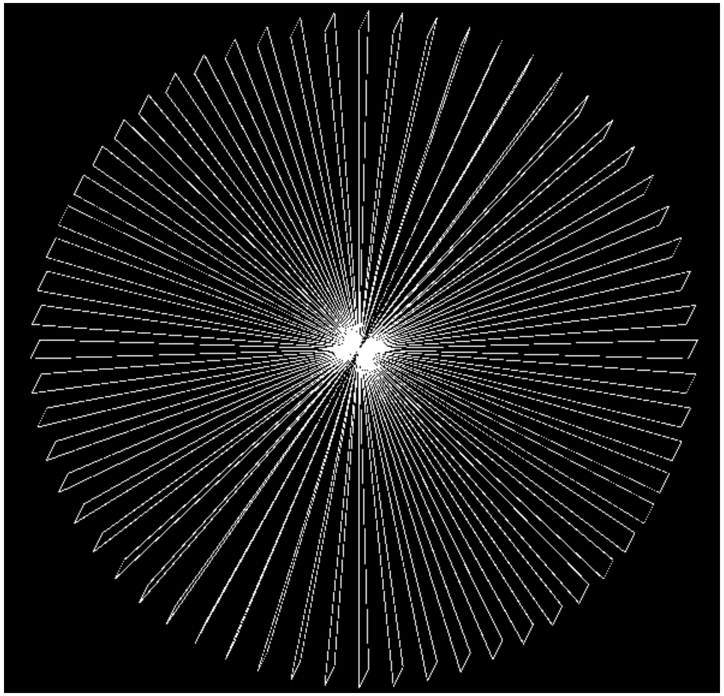
$$\begin{pmatrix} & X & \\ & \boxed{X} & X \\ O & O & X \end{pmatrix} \quad \begin{pmatrix} & X & \\ & \boxed{X} & O \\ X & & O \end{pmatrix}$$

Masking

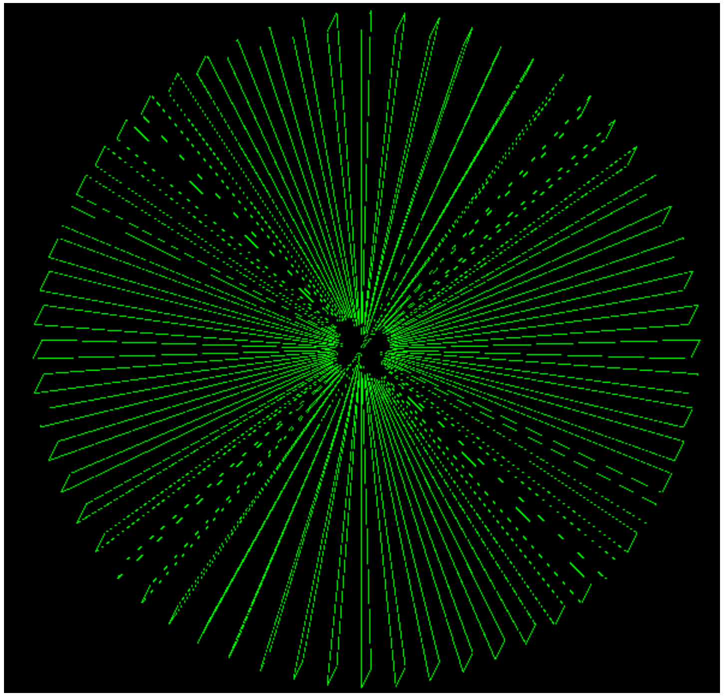
- For instance, if all 3x3 pixels are “edge” ones, there is no long dominating edge, and we do not want to smooth-out high-frequency in this case, etc.



Masking



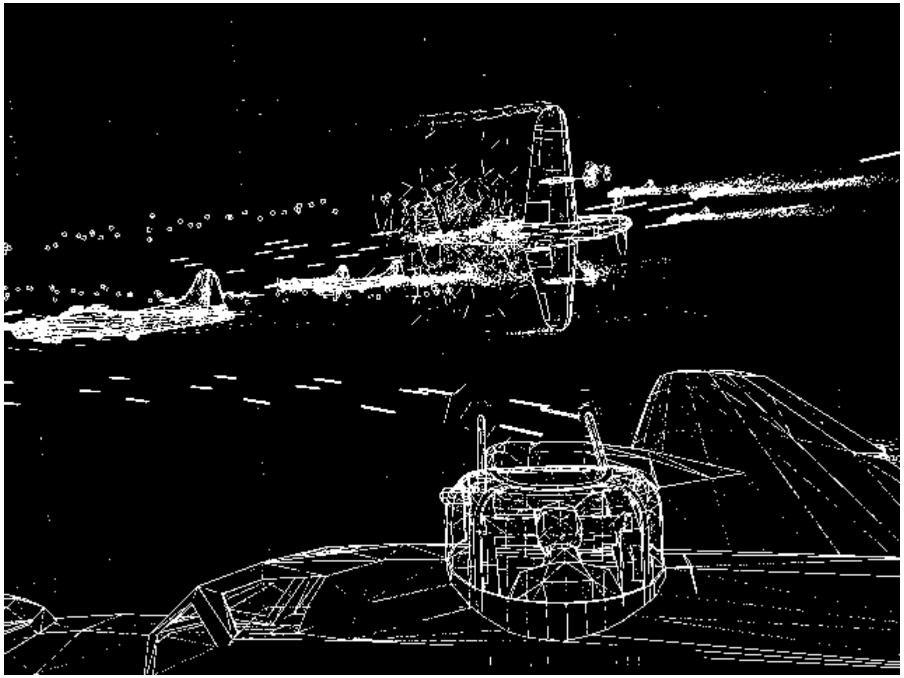
"Found" Edges



After Masking

Bringing It All Together: Sample Implementation

- Four shader passes implemented using DirectX 10.1
- Pass 1: Identify edge pixels using the MSAA buffer. Seed the frame buffer by performing a standard resolve at each pixel



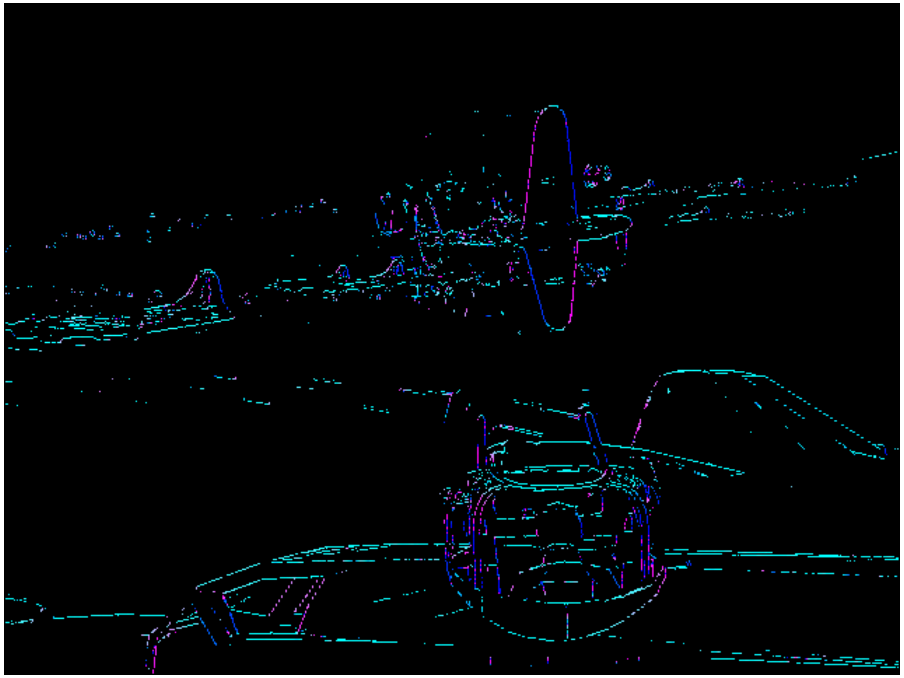
AMD The future is fusion

Pass 2

- Mask out candidate pixels using edge patterns

Pass 3

- Compute "gradients"
- Perform thresholding to further eliminate pixels



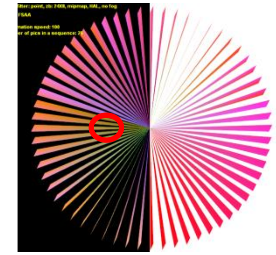
Pass 4

- Calculate the final frame buffer color for the pixels from Pass 3 using the presented integration method with input samples from a 3x3 pixel neighborhood
- Integration and weights are computed in shader
- All other pixels were already filtered in Pass 1

Performance Results

- Shipping as a driver feature on ATI Radeon HD GPUs: *Edge-Detect Custom Filter AA* since 2007
- Filtering performance on scenes from Futuremark 3DMark03 using an ATI Radeon HD 4890 on an AMD Phenom II X4 3.0 GHz
  - 0.25 to 1.7 ms at 800x600
  - 0.5 to 3 ms at 1024x768
  - 1 to 5 ms at 1280x1024
- Performance dependent on the number of edges in the scene

Quality



4x AA

4 levels of gradation

New filter using 4x AA samples

10 levels of gradation

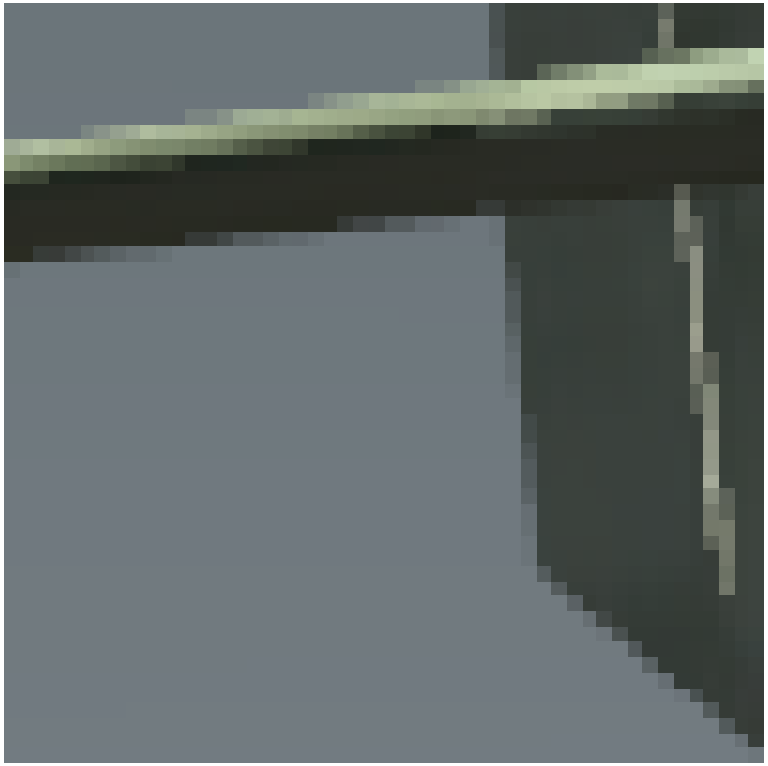
8x AA

8 levels of gradation

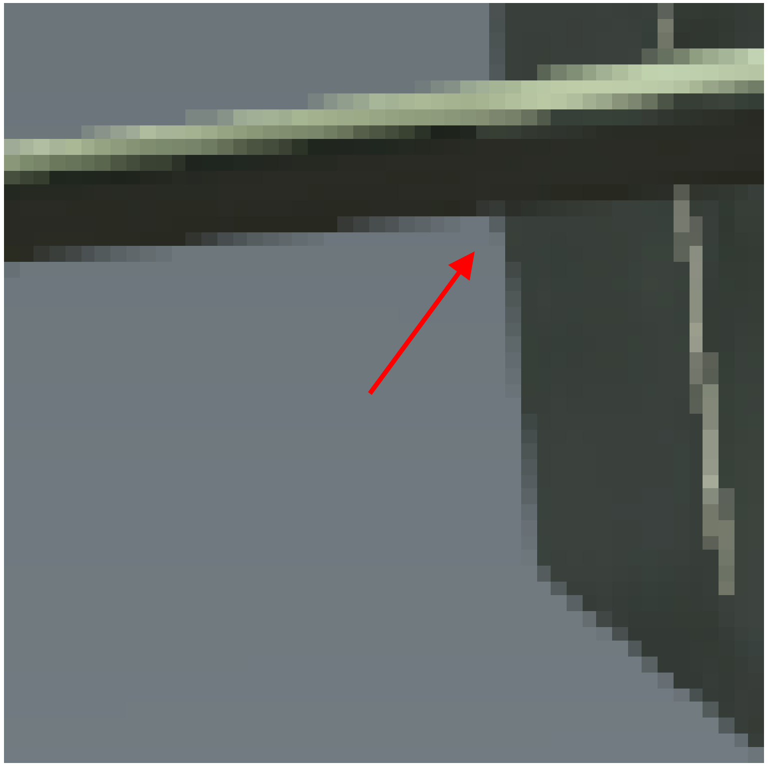
New filter using 8x AA samples

22 levels of gradation

Quality (“Corner Smoothing”)



8x AA



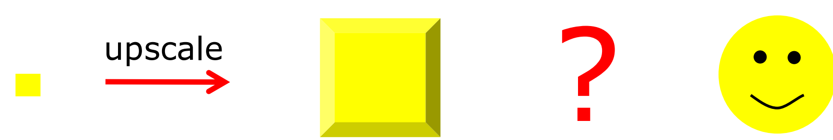
New filter using 8x AA samples

Results Evaluation

- We did not run any numerical error evaluations, as it is not clear how they can be interpreted
- No apparent temporal artifacts; temporal artifacts are difficult to characterize numerically
- Numerous reviewers evaluated productized version and found visual results to be good
- Thin objects (grass blades, etc.) have more gaps than with high factor multisampling, but this is expected (and can be fixed to a degree)

Future Work

- Use more complex approximating functions for better edge classification (edge detection)
- Handle cases of curved isolines
- Morph filter kernel shape based on isoline curvature (approximating function parameters and/or approximation error)
- Sample patterns improvement
- Try to apply this to upscaling
  - Image semantics problems need to be solved



Thank You!

- Acknowledgments:
  - Jeff Golds (AMD)
  - Futuremark
  - Tommti Systems

- Some additional implementation details will be presented during the AMD talk in “Advances in Real-Time Rendering in 3D Graphics and Games” at SIGGRAPH Monday August 3rd

Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.