

Análise e Projeto de Filtros Digitais para Correção do Efeito de *Aliasing* Aplicados na Computação Gráfica

Autores: Daniel Centeno Einloft e Jean Larrocca
Pontifícia Universidade Católica do Rio Grande do Sul
(PUCRS)

Disciplina: Aplicações de Processamento Digital de Sinais

Professor: Dr. Denis Fernandes

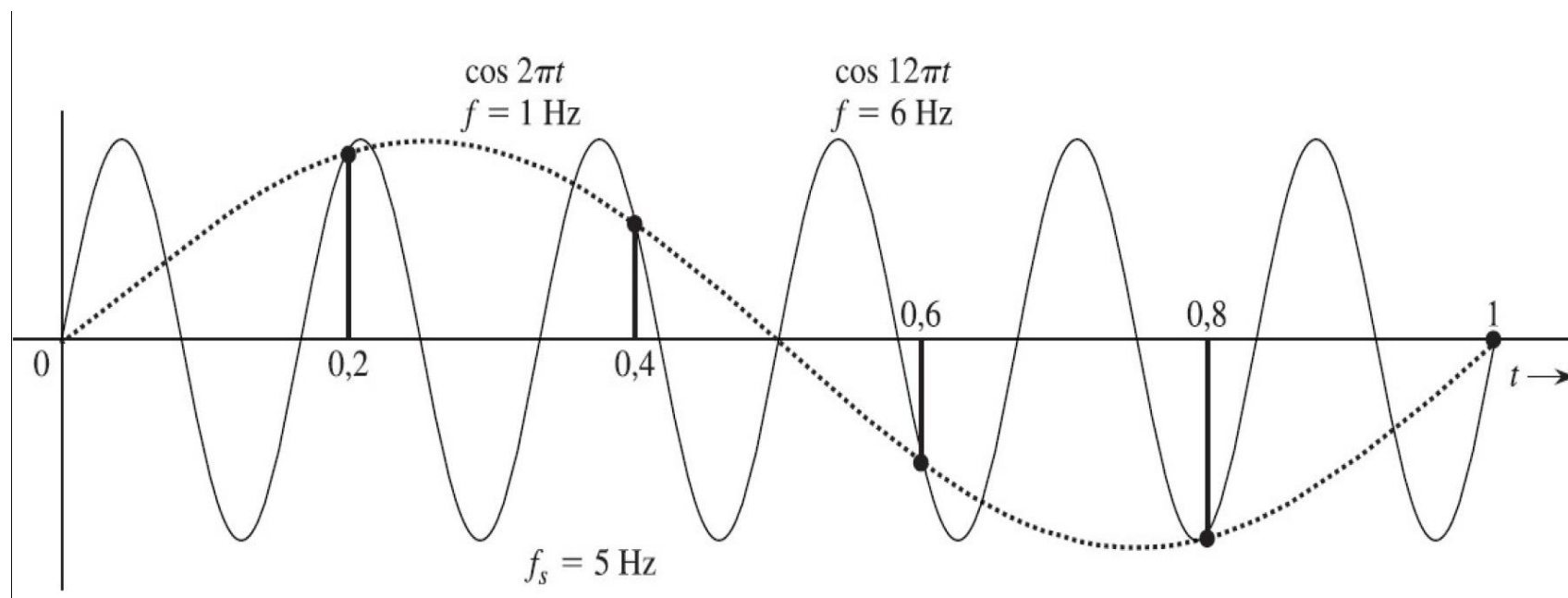
Sumário

- 1 - Introdução
- 2 - Motivação
- 3 - Filtros de Anti-Aliasing (AA)
- 4 - Filtros de Texturas (TF)
- 5 - Implementação de 5 filtros
- 6-Resultados
- 7 - Conclusão
- 8 - Trabalhos Futuros
- 9 - Agradecimentos
- 10 - Referências

Introdução: O Problema do *Aliasing*

- Taxa da Amostragem de Nyquist:

$$f_a > 2 * f_m$$

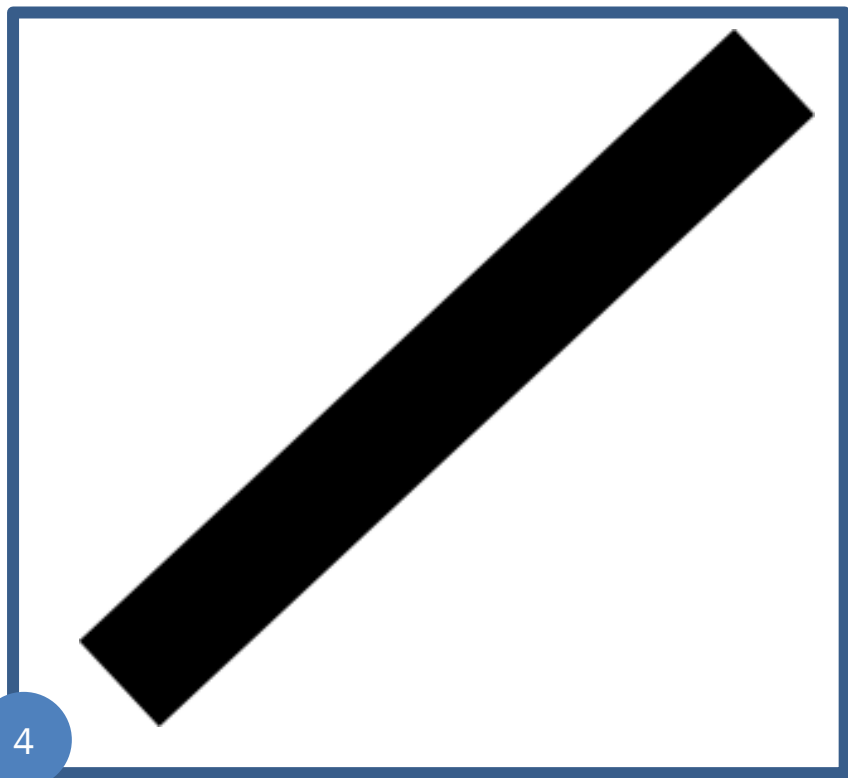


Fonte: B. P. Lathi, Sinais e Sistemas Lineares

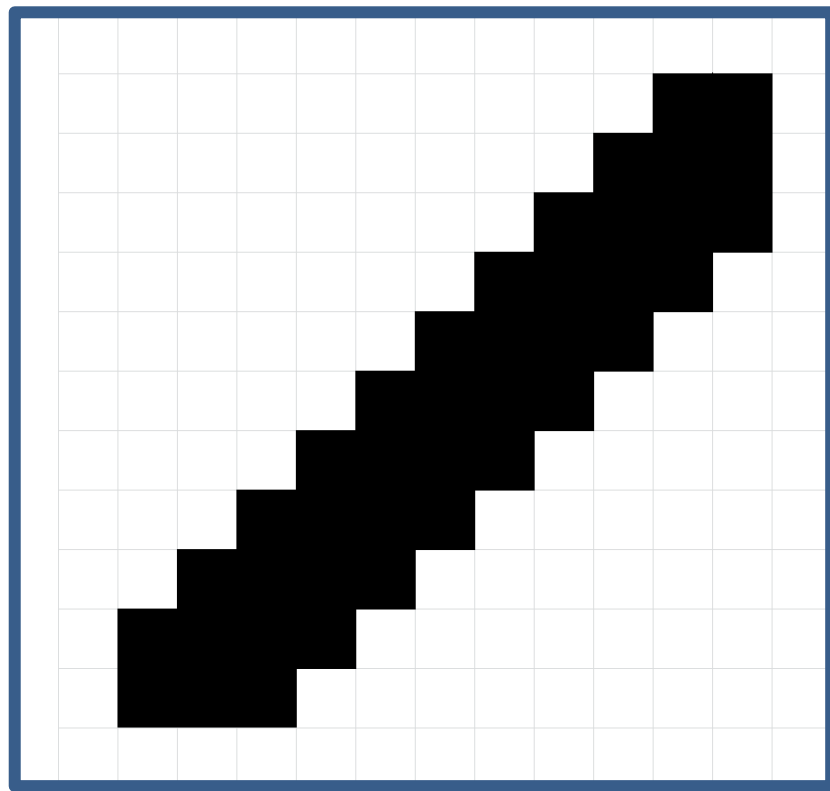
Introdução: *Aliasing* em Computação Gráfica

- Poucas Amostras → Poucos Pixels
- **Consequência 1:** Bordas Serrilhadas (*jagged edges*)

Linha Original

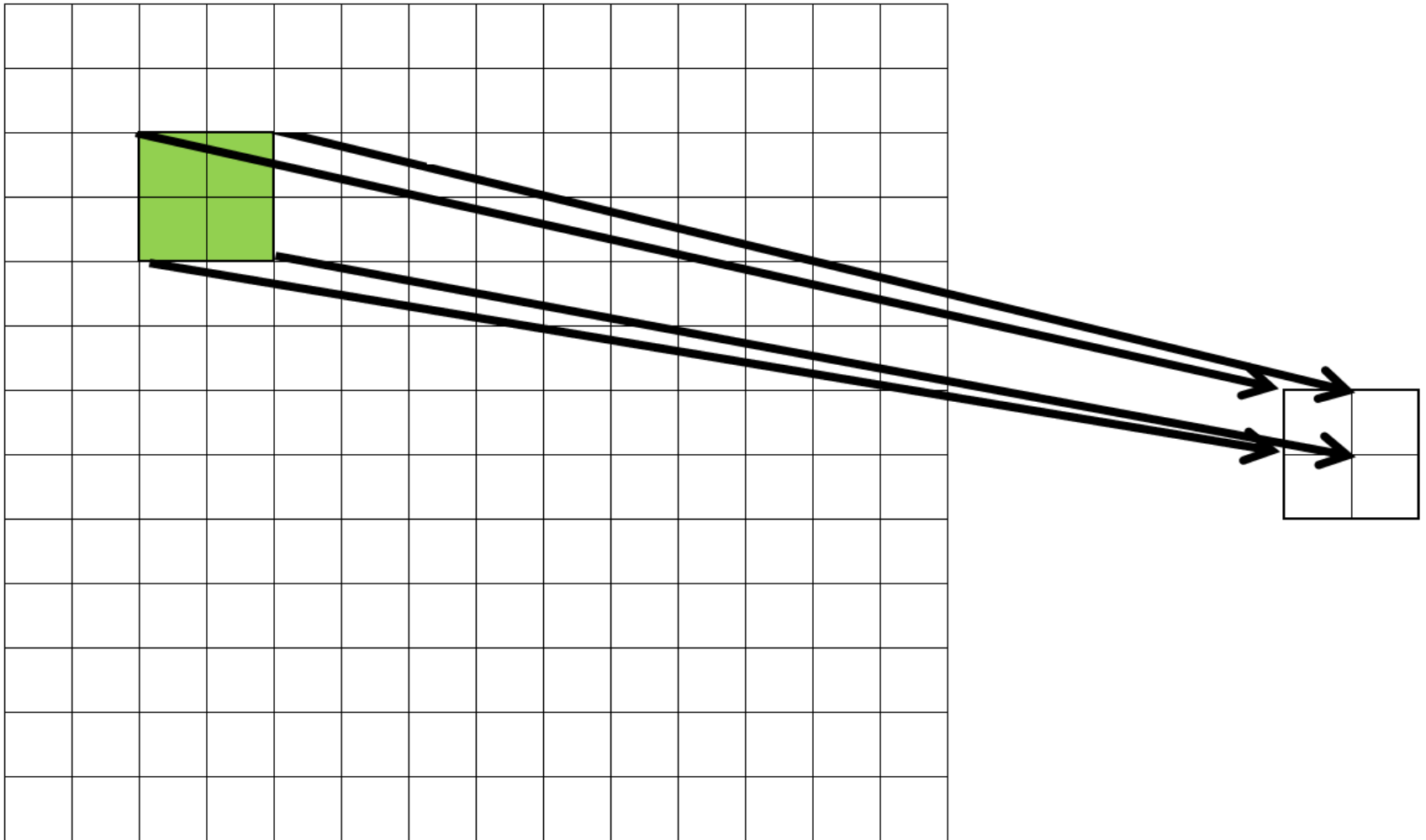


Linha Digitalizada



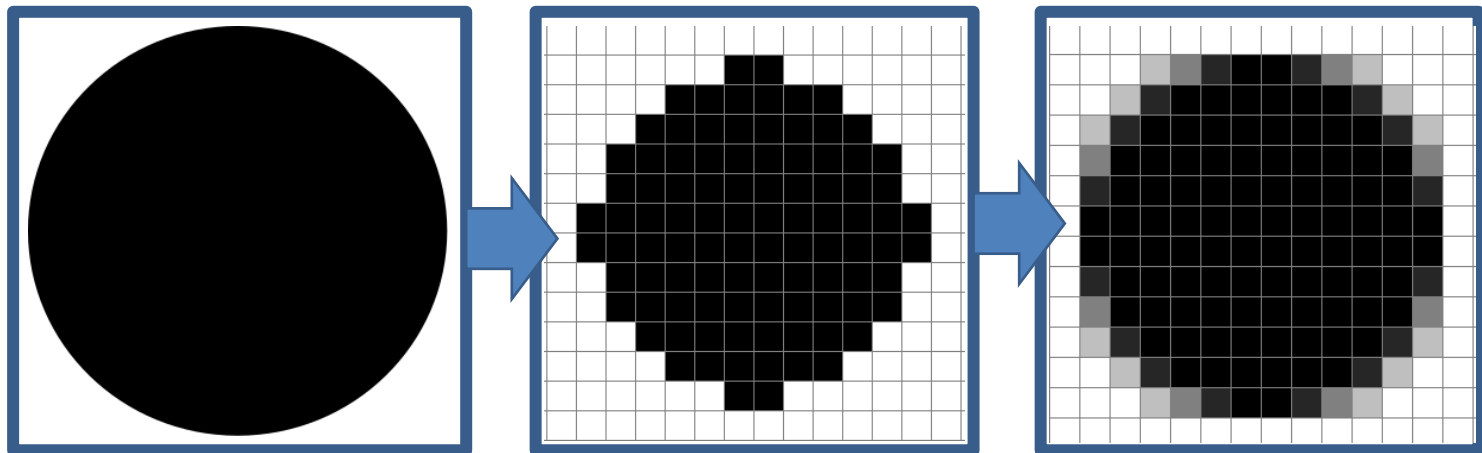
Introdução: *Aliasing* em Computação Gráfica

- **Consequência 2:** Magnificação e Minificação de Texturas



Introdução: Soluções

- Aplica-se diversos filtros para reduzir efeitos de serrilhados (*Anti-Aliasing*) e texturas borradas (*Texture Filtering*).
- Em jogos eletrônicos, são utilizados filtros diferentes para cada caso.
- Exemplo de filtro de AA:



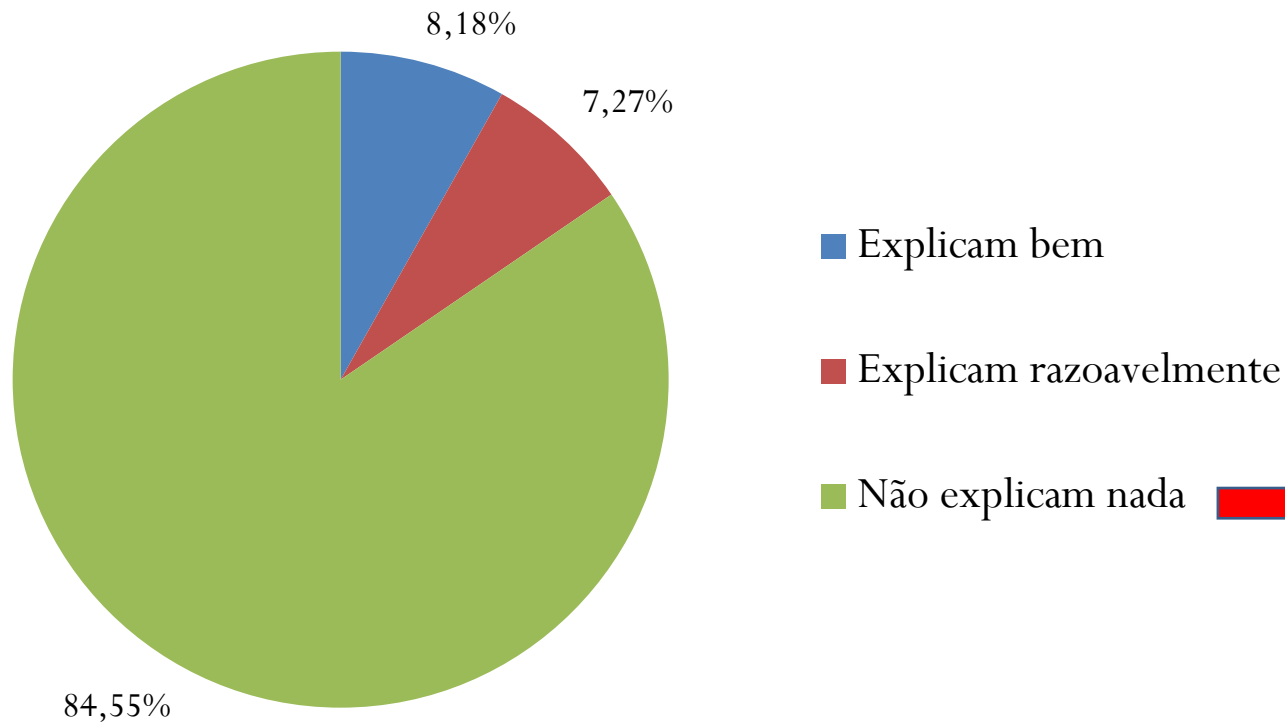
Introdução: Exemplo de Painel de Configurações dos Jogos



Fonte: Jogo World of Warcraft

Introdução: Divulgação de Informações

Quantidade de jogos que possuem explicações sobre AA/TF utilizados



Motivação

- Foi feita uma pesquisa com 10 pessoas (jogadores de PC e Consoles).
- Jogam aproximadamente +30 horas semanais (nas férias!).
- 80% sabiam definir AA, mas ninguém soube definir TF.
- 90% não sabiam diferenciar os filtros de ambas as classificações.



- 100% acharam que estas informações não são bem divulgadas pelas desenvolvedoras. Apontaram que o ideal seria uma tela explicativa de cada filtro e sua performance e, se possível, comparação visual.

Motivação: Pesquisa Inicial

- Existem muitos trabalhos envolvendo Anti-Aliasing!
 - FXAA, MLAA, SMAA, CSAA, EQAA, TAA, TXAA, DEAA, AXA, GBAA, GPAA, ACAA, CMAA, CFAA, TAAA, TrMSAA, SRAA, TMLAA...



- Filtros de Texturas são mais antigos e estão mais disseminados.
 - Filtros Bilinear, Trilinear e Anisotrópico.

Proposta

- Pesquisar como os filtros são divulgados nos jogos.
- Redução de filtros: considerar apenas os filtros que são mais utilizados em jogos dos últimos 11 anos, estudá-los e apresentá-los.
- Desenvolver alguns destes filtros e testá-los com usuários.

Pesquisa dos Filtros

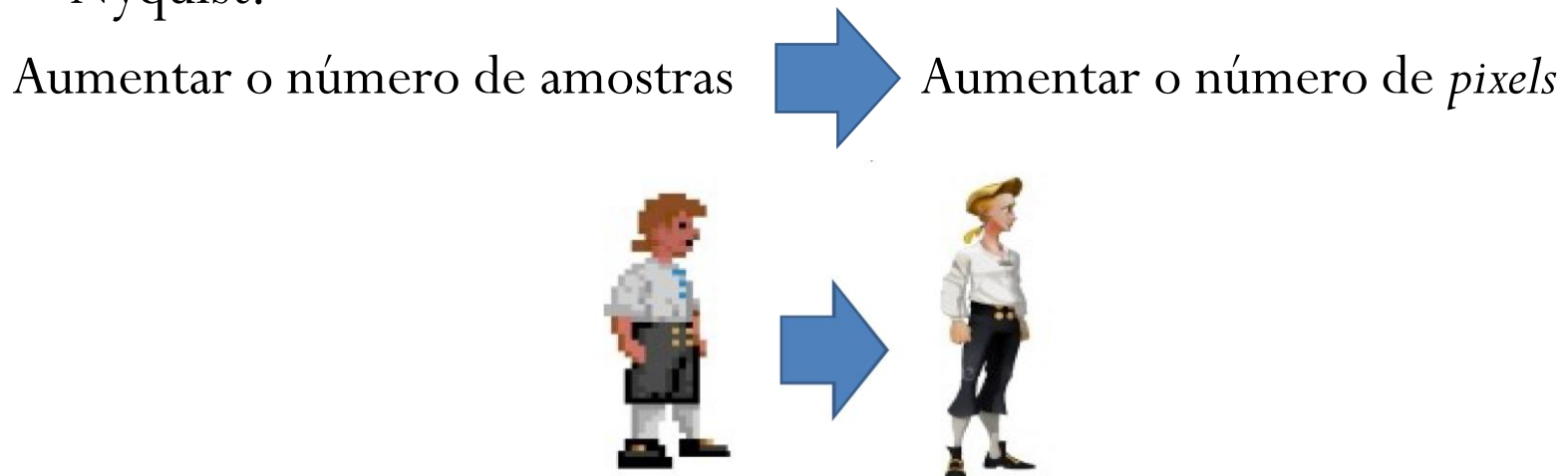
- Número de jogos: 110 jogos dos últimos 11 anos (2007-2017).
- Filtros de AA:
 - **SSAA**: *Super Sampling Anti-Aliasing*
 - **MSAA**: *Multi Sampling Anti-Aliasing*
 - **FXAA**: *Fast Approximate Anti-Aliasing*
 - **CSAA (nVidia)/ EQAA (AMD)**: *Coverage Sampling Anti Aliasing/ Enhanced Quality Anti-Aliasing*
 - **TXAA (nVidia)**: *Temporal Anti-Aliasing*
 - **MLAA/SMAA**: *Morphological Anti-Aliasing/ Enhanced Subpixel Morphological Anti-Aliasing*
 - **Outros/ Desconhecido**
- Filtros de Textura:
 - **Bilinear**
 - **Trilinear**
 - **Anisotropico**

Trabalhos Relacionados: Filtros de *Anti-Aliasing* em Jogos Digitais

- Em jogos: Podem ser divididos em duas classificações:
 - Filtros de Pré-Processamento: são utilizados durante a renderização.
 - Filtros de Pós-Processamento: são utilizados depois da renderização.
- Desenvolvedores podem misturar filtros de pré e pós processamento!

Trabalhos Relacionados: AA de Pré-Processamento

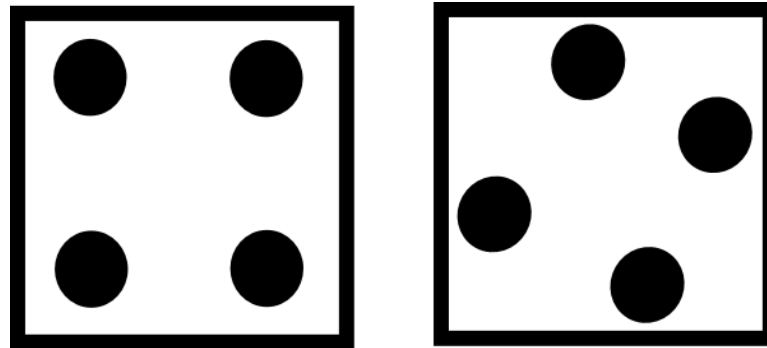
- São executados durante a renderização dos gráficos.
- Abordam o problema baseando-se no princípio da amostragem de Nyquist:



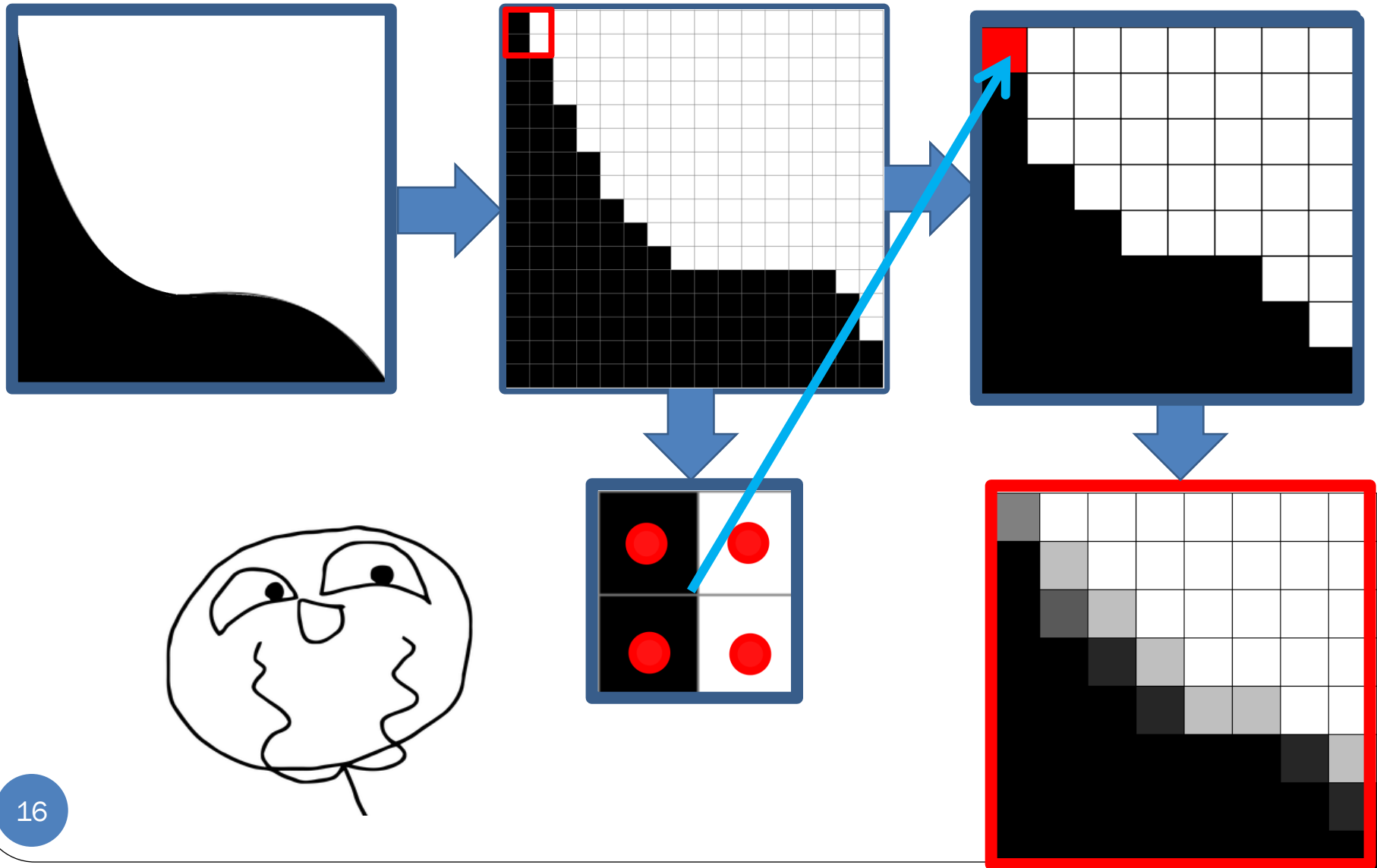
- Filtros estudados: *Super Sampling Anti-Aliasing (SSAA)*, *Multi Sampling Anti-Aliasing (MSAA)*, *Coverage Sampling Anti-Aliasing (CSAA)*, *Enhanced Quality Anti-Aliasing (EQAA)* e *Temporal Anti-Aliasing (TXAA)*.
- Características gerais: resolvem bem o problema, mas utiliza muita memória e processamento.

Trabalhos Relacionados: *Super Sampling Anti-Aliasing (SSAA)*

- Aumenta a resolução da renderização 2x, 4x ou 8x,
- “Comprime” a imagem para a resolução desejada.
- Utiliza amostras (*sub samples*) para fazer a compressão dos *pixels*.
- Em decadência devido ao uso excessivo de memória
- Disponível utilização através do Painel de Controle *nVidia*.
- Exemplos do tipo de amostragem:



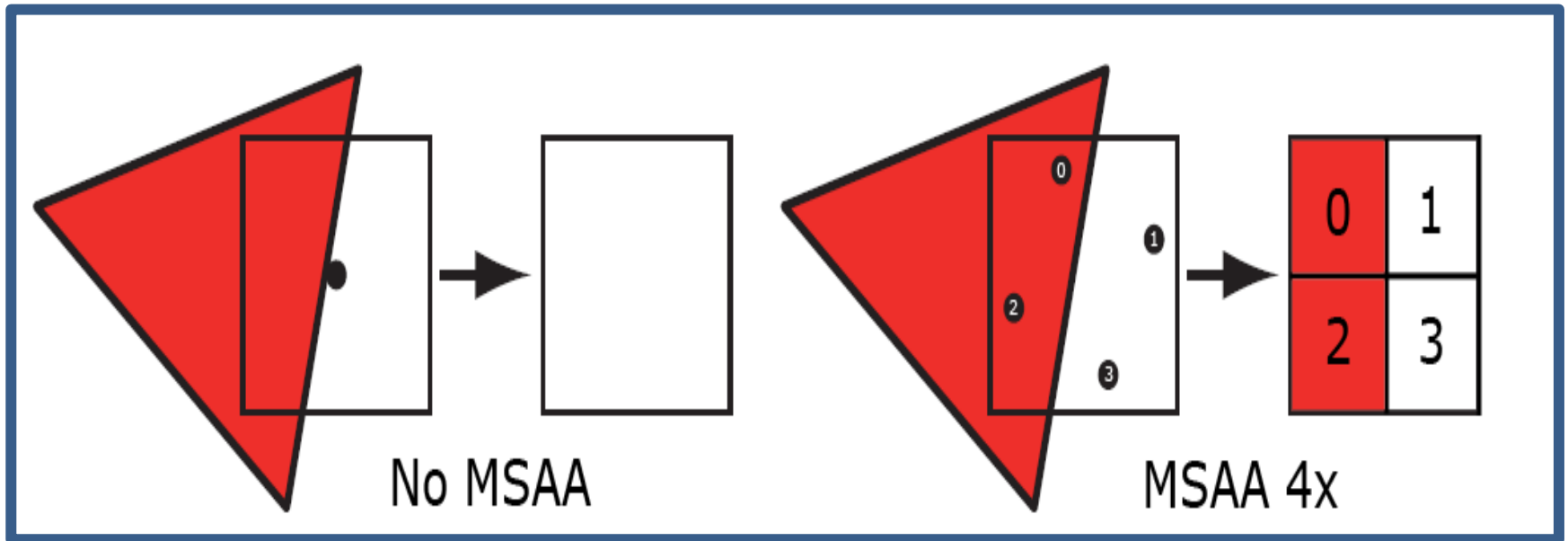
Super Sampling Anti-Aliasing (SSAA) Exemplo



Trabalhos Relacionados: *Multi Sampling Anti-Aliasing* (MSAA)

- Versão melhorada do SSAA.
- Faz a amostragem apenas nas bordas dos polígonos, reduzindo processamento!
- Consome menos memória que o SSAA, mas ainda gasta bastante.
- Resultado um pouco pior que SSAA, mas o ganho de performance o torna muito mais utilizado nos jogos.

Multi Sampling Anti-Aliasing (MSAA)

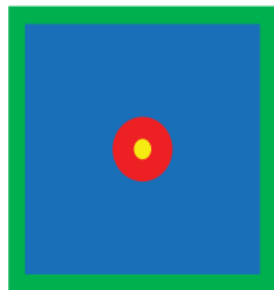


Fonte: <https://mynameismjp.wordpress.com/2012/10/24/msaa-overview/>

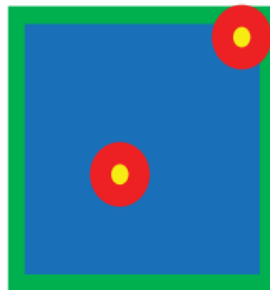
Trabalhos Relacionados: *Coverage Sampling Anti-Aliasing (CSAA)/ Enhanced Quality Anti-Aliasing (EQAA)*

- Funcionam da mesma forma.
- Reconhecem padrões nas amostras e aplica a filtragem só onde a borda é detectada.
- Mesmo analisando várias amostras, o cálculo só é feito nas amostras que detectam variações de cores.

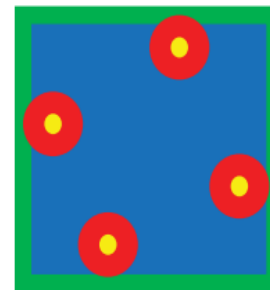
Coverage Sampling Anti-Aliasing (CSAA)/ Enhanced Quality Anti-Aliasing (EQAA)



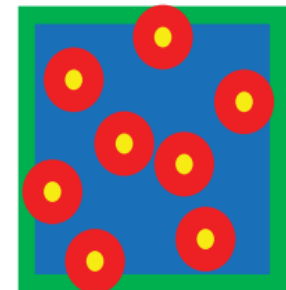
No AA





2x MSAA

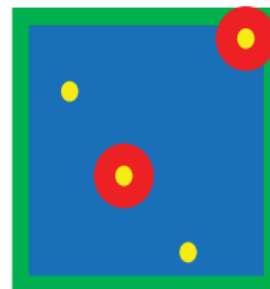


4x MSAA



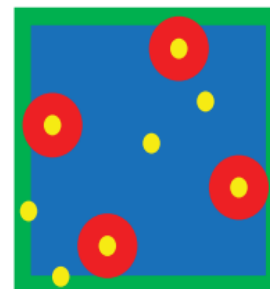
8x MSAA

-  = Pixel Boundary
-  = Color Sample
-  = Coverage Sample



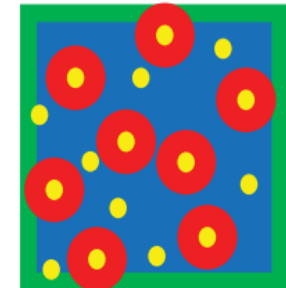
2f4x EQAA

4 coverage samples



4f8x EQAA

8 coverage samples



8f16x EQAA

16 coverage samples

Fonte: AMD: EQAA Modes for AMD 6900 Series Graphics Cards. Tech report

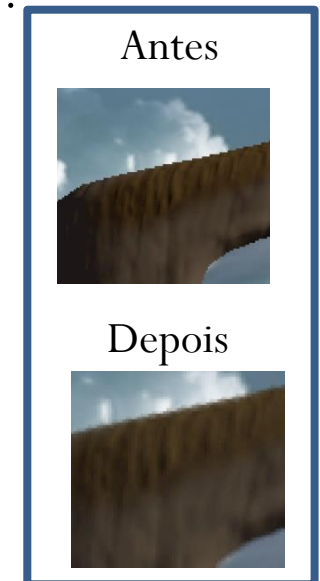
Trabalhos Relacionados: *Temporal Anti-Aliasing (TXAA)*

- Exclusivo para GPUs da *nVidia*.
- Desenvolvido para reduzir o serrilhado durante a movimentação de objetos.
- *Multi Sampling* + filtros temporais



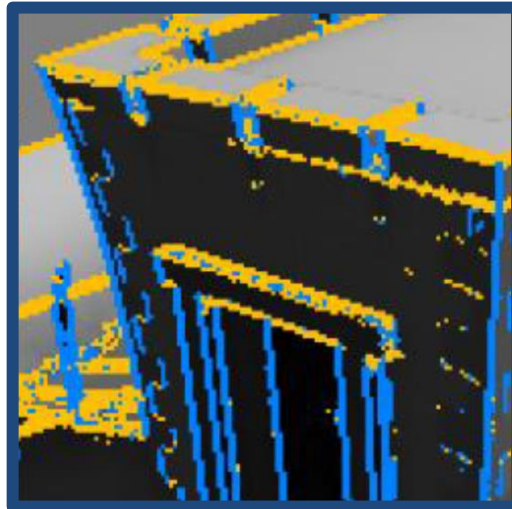
Trabalhos Relacionados: Filtros de Pós-Processamento

- Pós Renderização.
- Não se baseiam no princípio da amostragem.
- Algoritmos que focam na detecção de bordas e formas, e aplicam borramentos específicos.
- Rápidos e Leves
- Filtros estudados: *Fast Approximate Anti-Aliasing (FXAA)*, *Morphological Anti-Aliasing (MLAA)* e *Subpixel Morphological Anti-Aliasing (SMAA)*.



Trabalhos Relacionados: *Fast Approximate Anti-Aliasing (FXAA)*

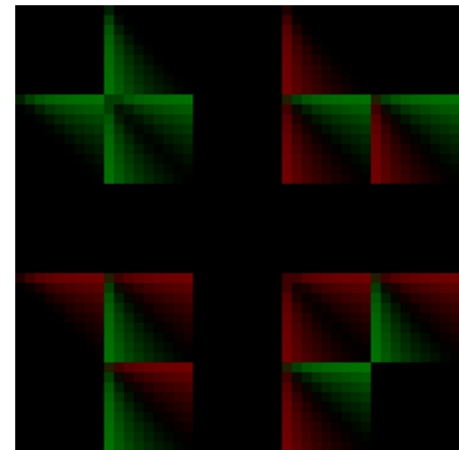
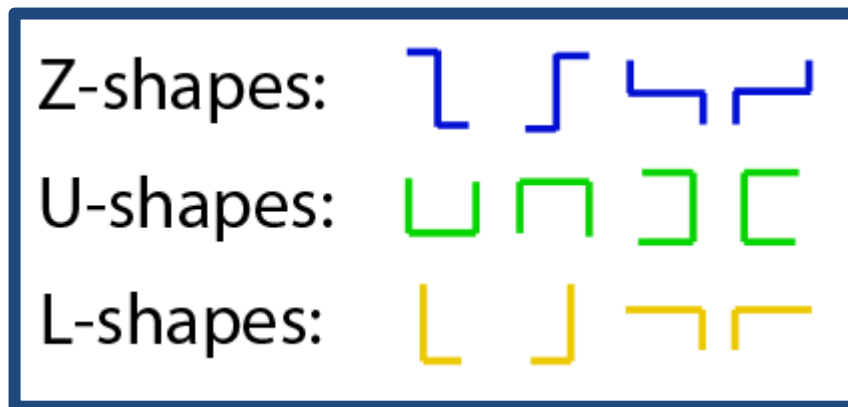
- Primeiro da “nova geração” de filtros de AA.
- Timothy Lottes, *nVidia*.
- Detecção de bordas verticais e horizontais.
- Aplicação de *blurring* de acordo com os limites das bordas.
- Explicações mais detalhadas na seção 5.



Fonte: Lottes, T.: FXAA Tech Report.

Trabalhos Relacionados: *Morphological Anti-Aliasing (MLAA)*

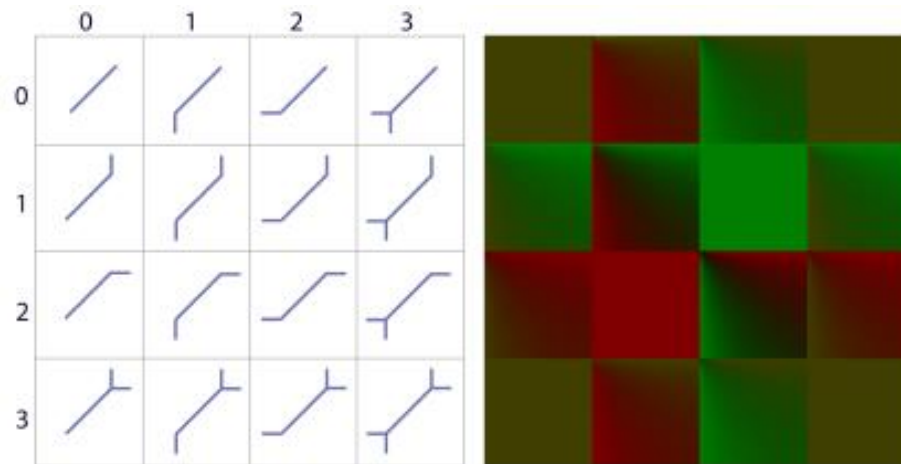
- Jorge Jimenez, Universidade de Zaragoza.
- Um pouco mais lento que o FXAA.
- Baseia-se na detecção de formas em bordas.
- Para cada forma detectada, aplica um *blur* diferente e usa padrões pré computados para aumentar a performance.
- Será mais explicado com mais detalhes na seção 5.



Fonte: SMAA- Enhanced Subpixel Morphological Anti-Aliasing

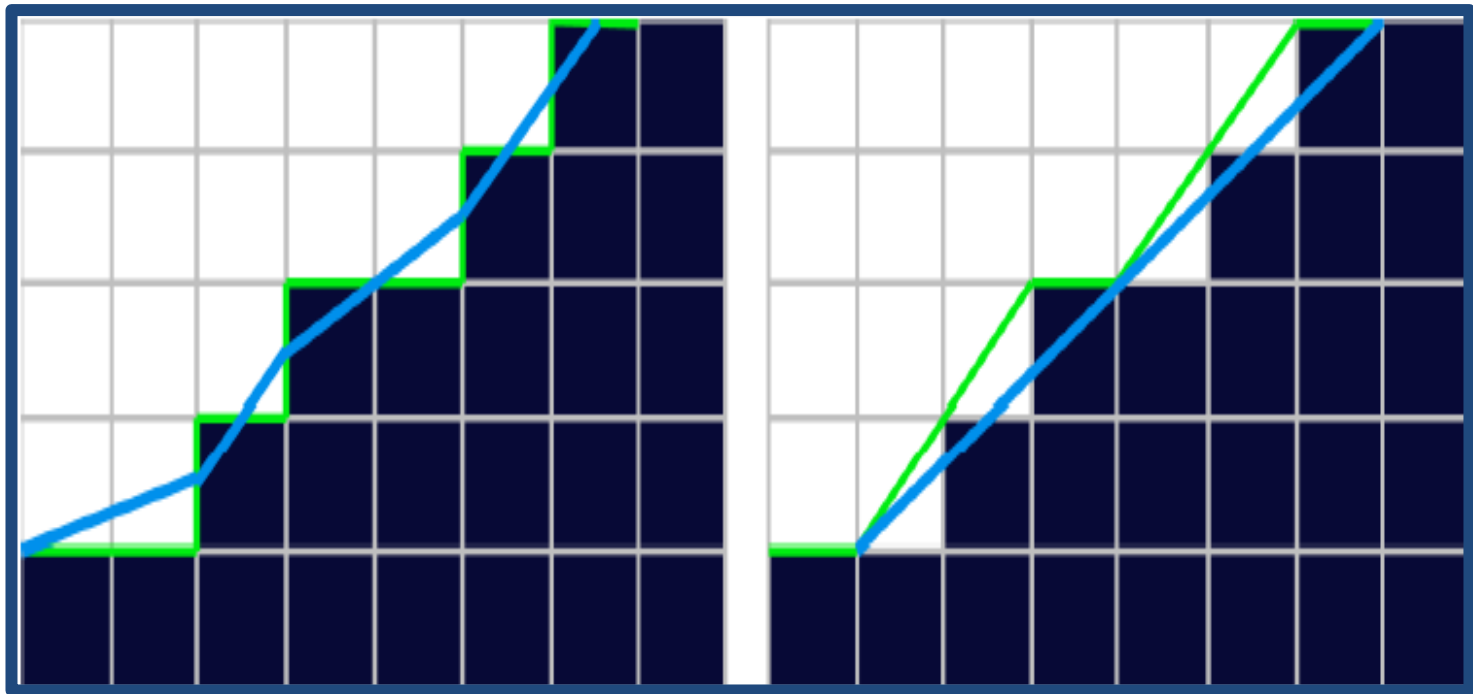
Trabalhos Relacionados: *Enhanced Subpixel Morphological Anti-Aliasing* (SMAA)

- Evolução direta do MLAA.
- Algoritmo de identificação de bordas é melhorado, possuindo mais padrões.
- Adiciona filtragem temporal (SMAA T).



Fonte: SMAA- Enhanced Subpixel Morphological Anti-Aliasing

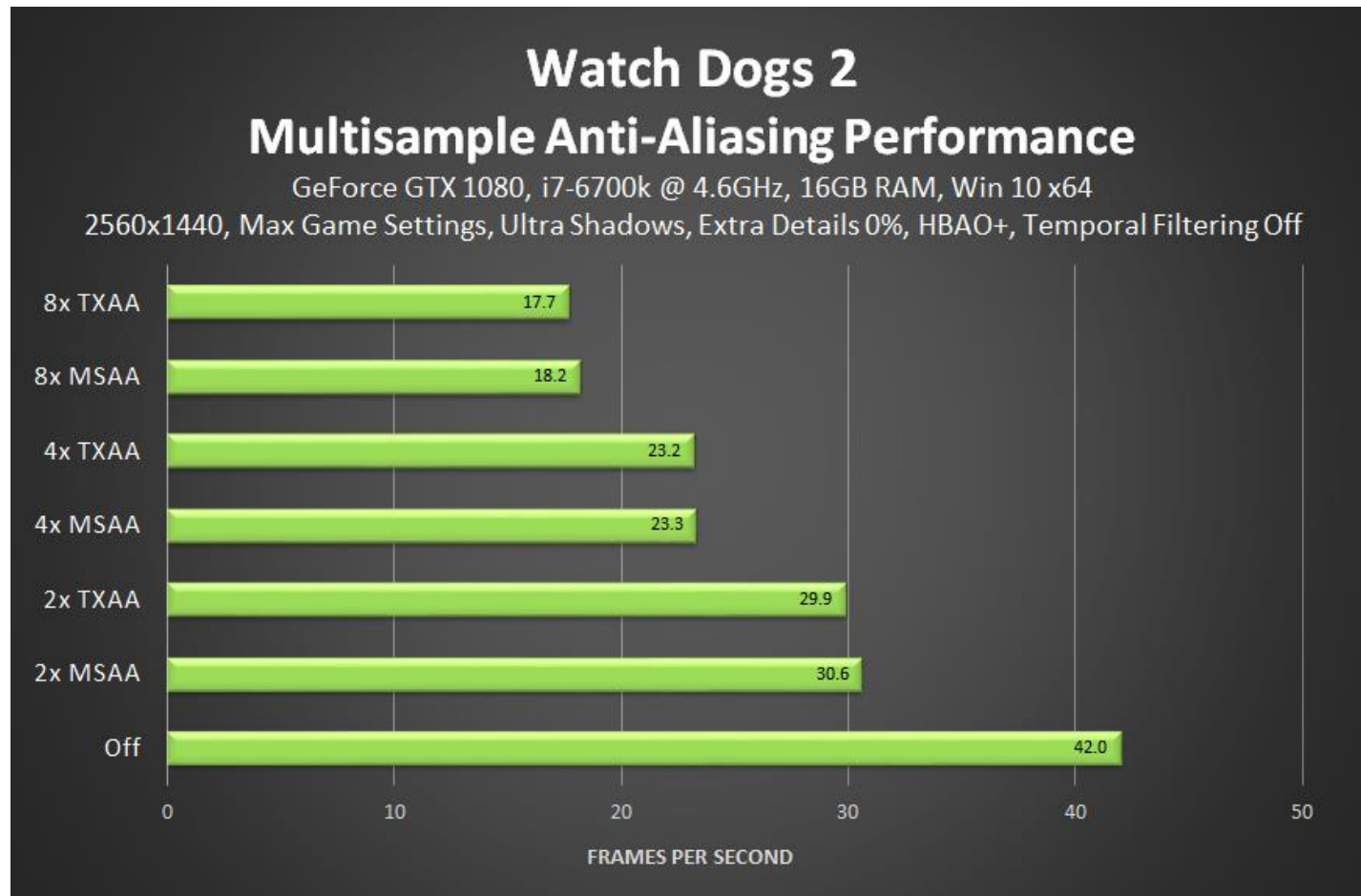
MLAA x SMAA



Fonte: SMAA- Enhanced Subpixel Morphological Anti-Aliasing

Trabalhos Relacionados: Comparação de Performance: Pré Processamento

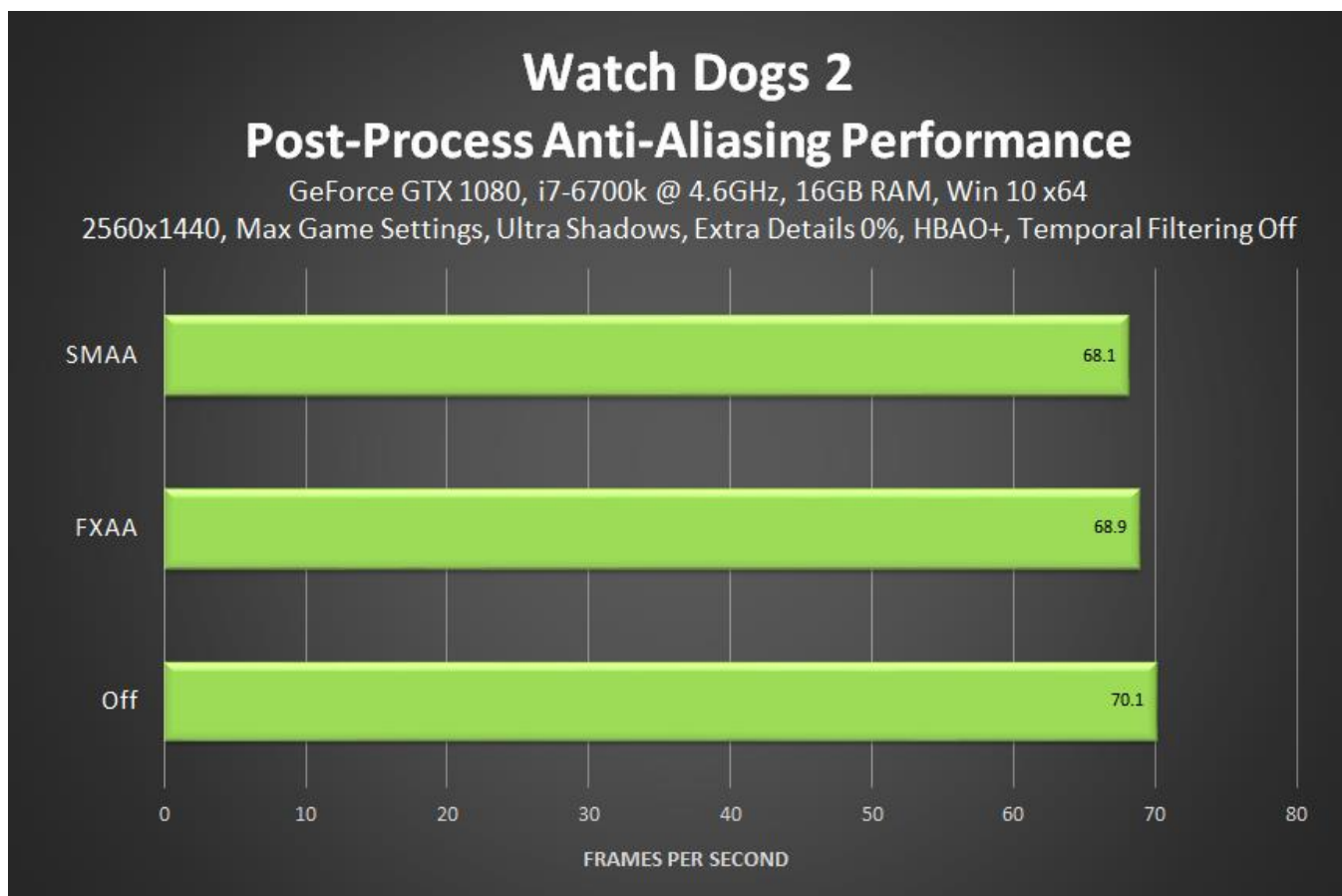
- Jogo: Watch Dogs 2 (2016)



Fonte: <http://www.geforce.com/whats-new/guides/watch-dogs-2-graphics-and-performance-guide#watch-dogs-2-multisample-anti-aliasing>

Trabalhos Relacionados: Comparação de Performance: Pós Processamento

- Jogo: Watch Dogs 2 (2016)



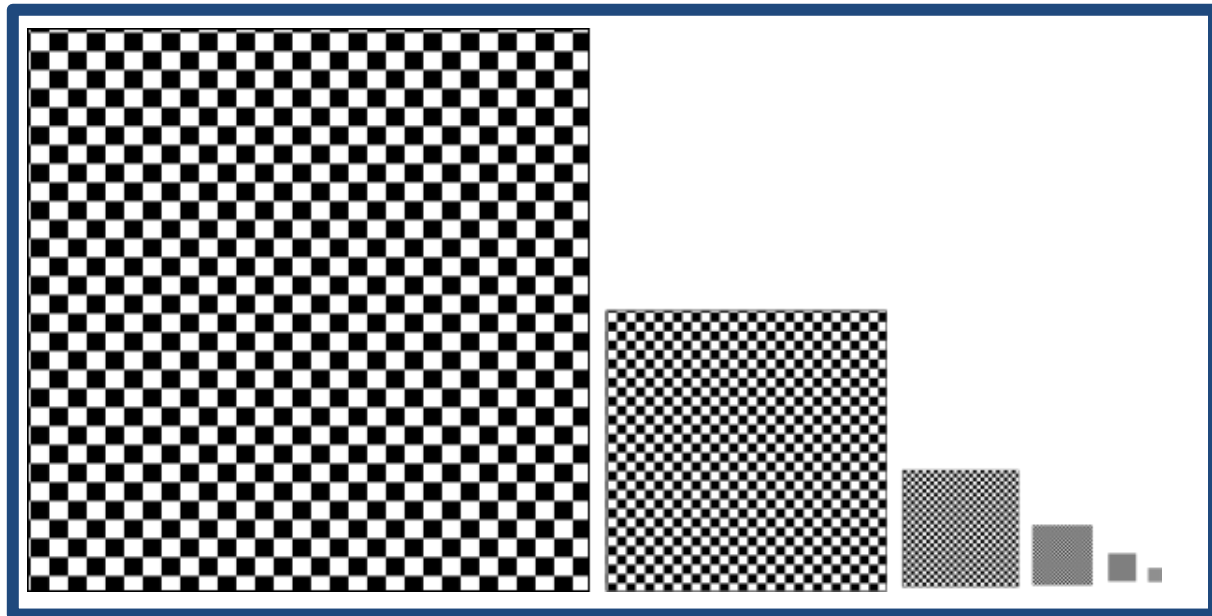
Fonte: <http://www.geforce.com/whats-new/guides/watch-dogs-2-graphics-and-performance-guide#watch-dogs-2-multisample-anti-aliasing>

Trabalhos Relacionados: Filtros de Texturas

- Filtros de *Anti-Aliasing* normalmente pioram a qualidade da textura.
- Solução: Filtro específico para texturas!
- Utilizados para resolver os problemas de magnificação e minificação de texturas.

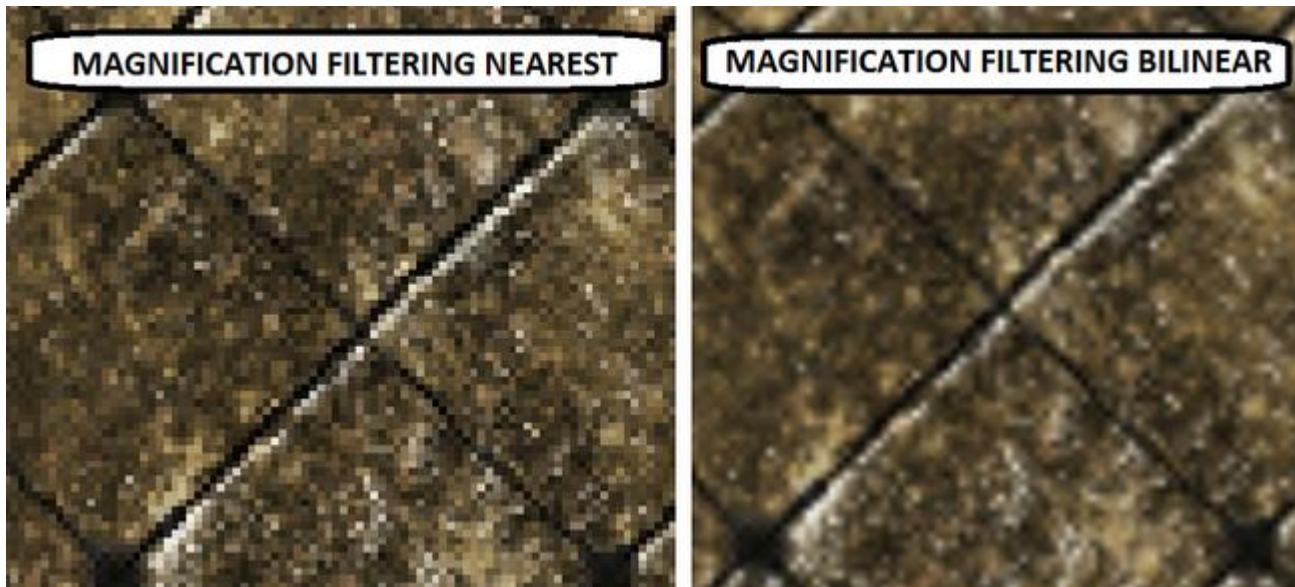
Trabalhos Relacionados: Mipmaps

- Servem para reduzir o cálculo na hora texturização de gráficos.
- Reduzem a textura em vários pedaços menores.



Trabalhos Relacionados: Filtro Bilinear

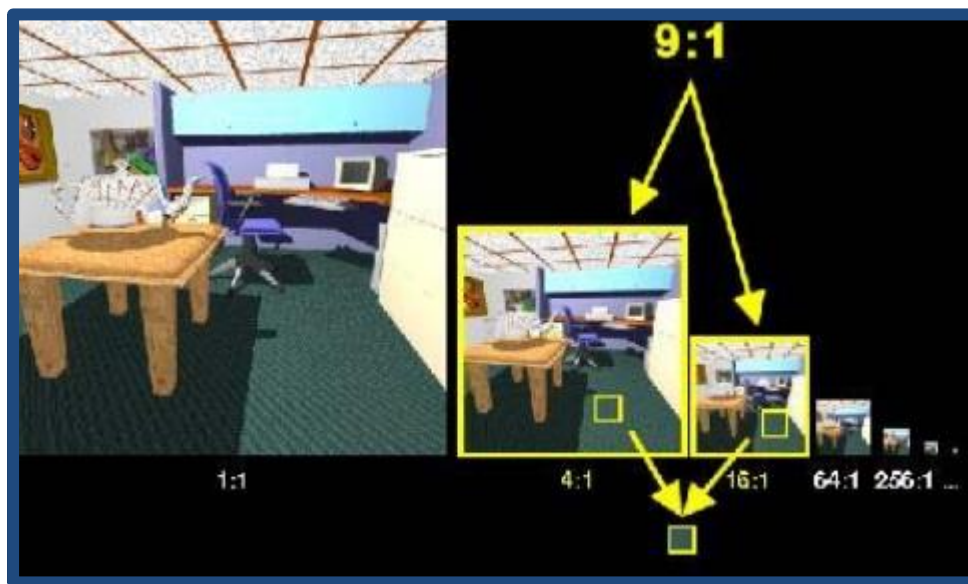
- Interpolação bilinear.
- Resolve o problema de magnificação.
- Faz a reconstrução da textura numa resolução maior (*upsampling*) a partir da aproximação de 4 pontos.



Fonte: https://gdbooks.gitbooks.io/legacyopengl/content/Chapter7/mag_filter.png

Trabalhos Relacionados: Filtro Trilinear

- Filtragem bilinear resolve o problema de texturas magnificadas, mas não resolve o problema todo!
- Filtragem trilinear utiliza o mesmo princípio, porém entre vários Mipmaps.
- Garante transição suave entre Mipmaps.



Bilinear x Trilinear

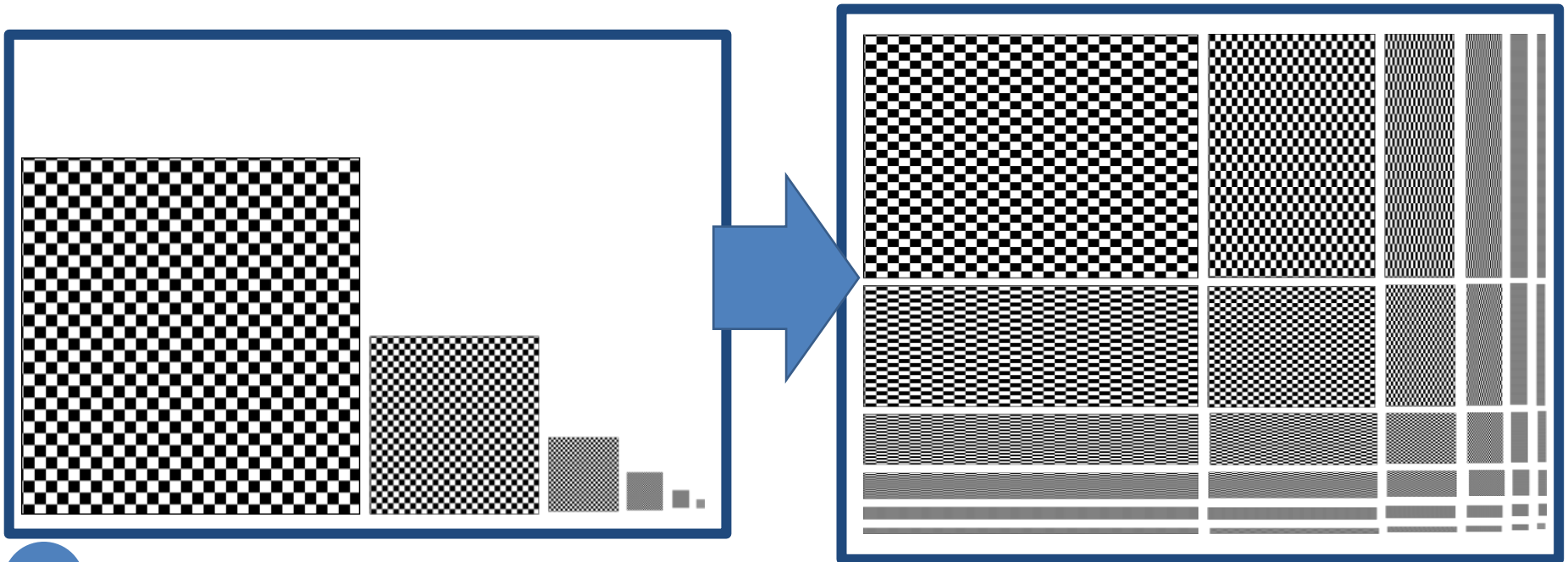


Fonte:

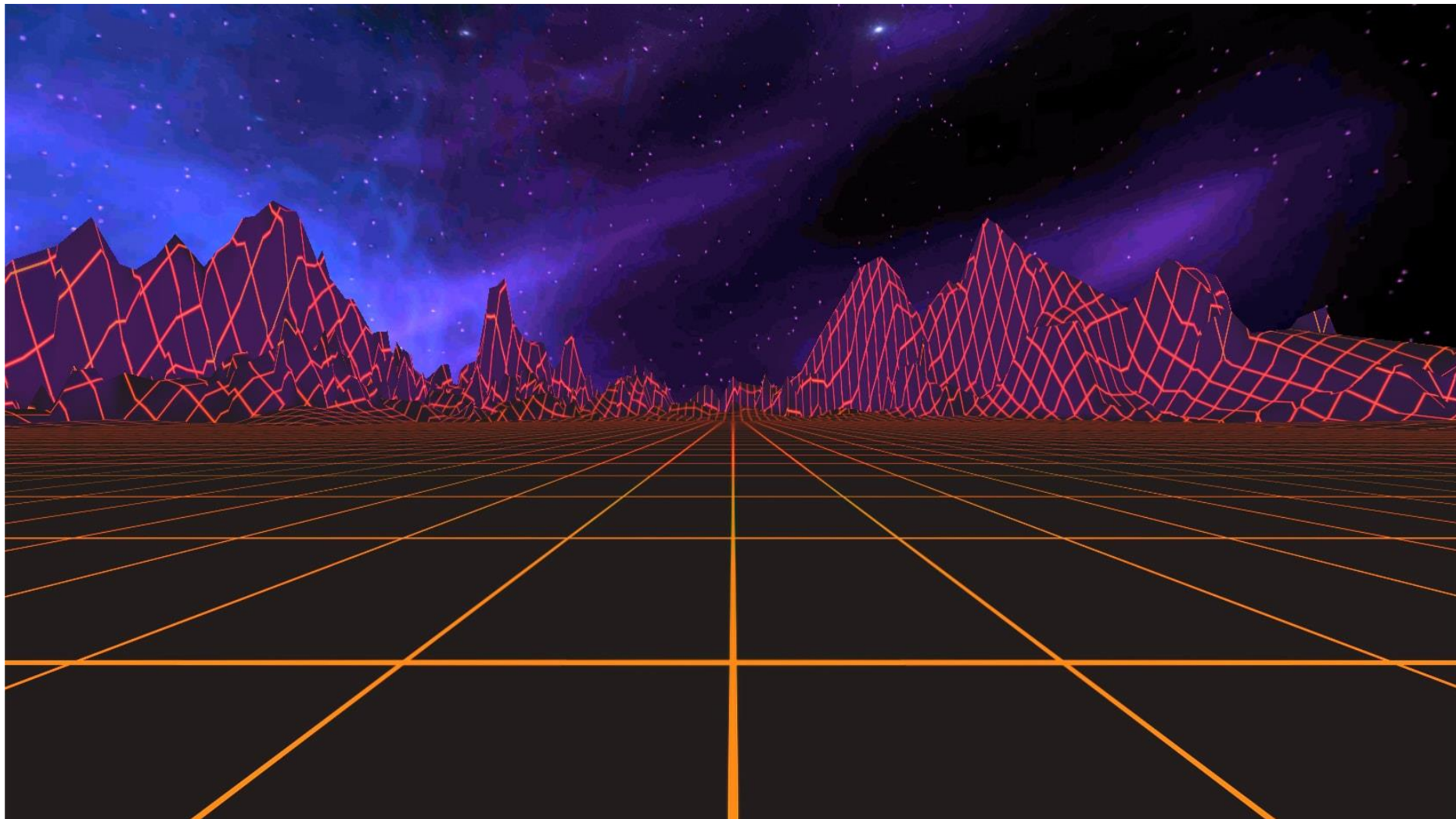
https://faculty.kaust.edu.sa/sites/markushadwiger/Documents/CS380_spring2015_lecture_12.pdf

Trabalhos Relacionados: Filtro Anisotrópico

- Filtro Bilinear e Trilinear são isotrópicos (uniformes!).
- Filtros anisotrópicos são recomendados para texturas oblíquas!
- Conta com Mipmaps anisotrópicos.



Filtro Anisotrópico



Implementação de Filtros de *Anti-Aliasing*



Implementação de Filtros de Anti-Aliasing

- Foi feito 5 tipos de filtros diferentes.
- Um filtro “genérico”, testado com filtro Gaussiano e de Mediana, aplicados com um detector de bordas Canny.
- Duas versões do SSAA, utilizando duas técnicas de amostragem diferentes.
- O algoritmo desenvolvido pela Timothy Lottes, da nVidia, em 2009: FXAA.
- O algoritmo desenvolvido por Jorge Jimenez em 2011: MLAA.

1- Aplicando filtros genéricos

- Inicialmente, aplicou-se filtros de mediana e gaussiano em imagens para selecionar o melhor filtro.



1- Aplicando filtros genéricos

- Em seguida, foi comparado a qualidade da imagem dos melhores resultados (filtro gaussiano, passa baixa) com a aplicação de um detector de bordas Canny.



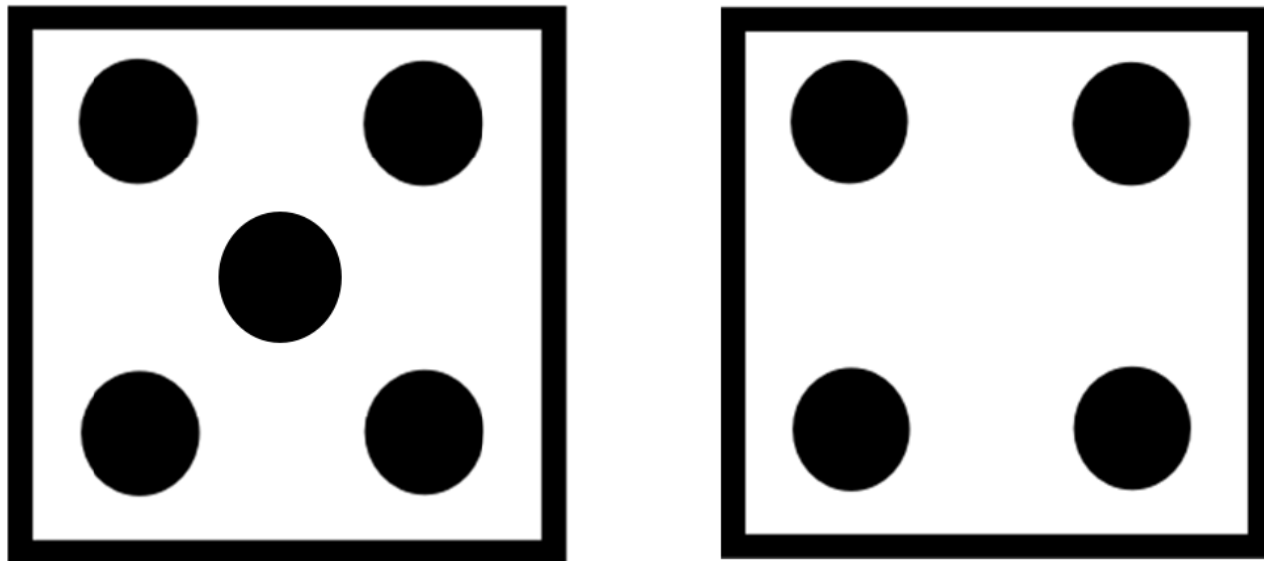
1- Filtro Genérico: Resultado Inicial

- Jogo: Borderlands Pre-Sequel



Implementação SSAA

- Gravado vídeos de jogos numa resolução mais alta e reduzindo-a pela metade.
- Dois tipos de amostragem:
 - *SSAA-1: High Resolution Anti-Aliasing (HRAA)* ou Quincunx.
 - *SSAA-2: Ordered Grid Super Sampling (OGSS)*



Resultados: SSAA-1

- Jogo: Borderlands Pre-Sequel



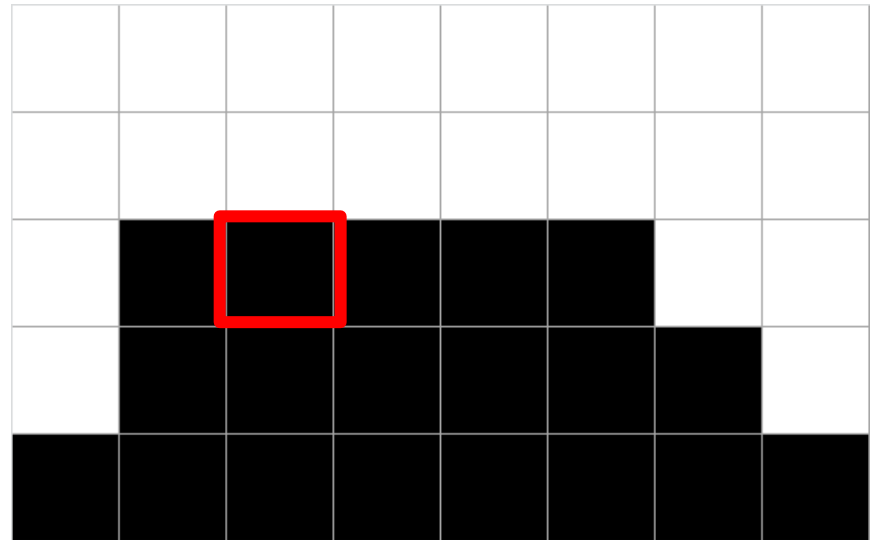
Resultados: SSAA-2

- Jogo: Borderlands Pre-Sequel



Implementação FXAA

- Um dos filtros mais utilizados em jogos.
- Pós-Processamento.



Exemplos baseados nas figuras de: http://blog.simonrodriguez.fr/articles/30-07-2016_implementing_fxaa.html

Implementação FXAA – Etapa 1

- Analisa-se a luminosidade da imagem (padrão CCIR 601):

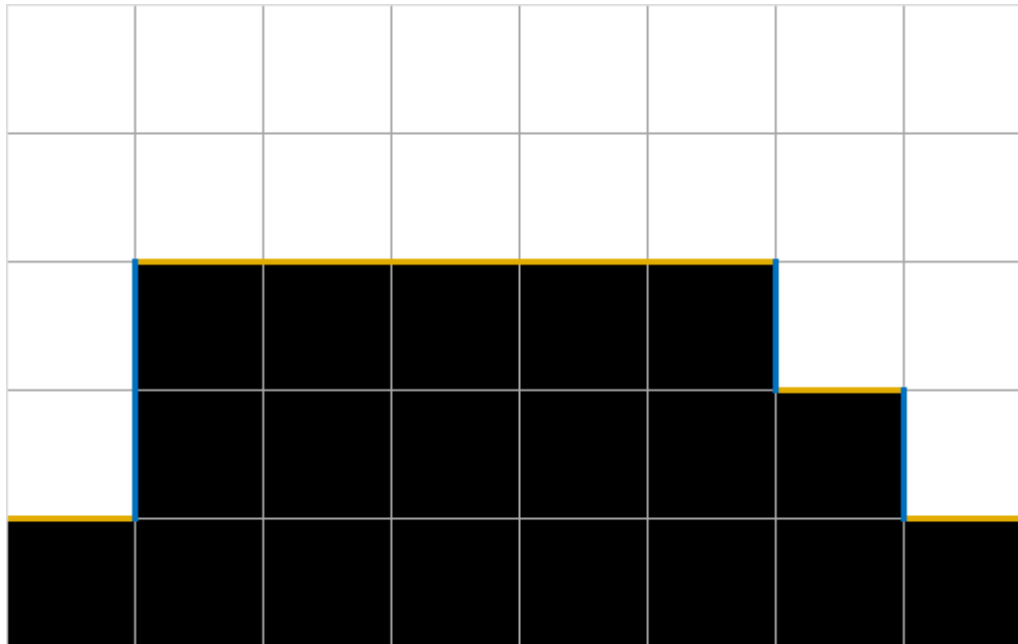
$$Y' = \sqrt{(0.299 * R^2 + 0.587 * G^2 + 0.114 * B^2)}$$

- Transforma-se a escala da imagem de 0-255 para 0-1.

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1
1	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0

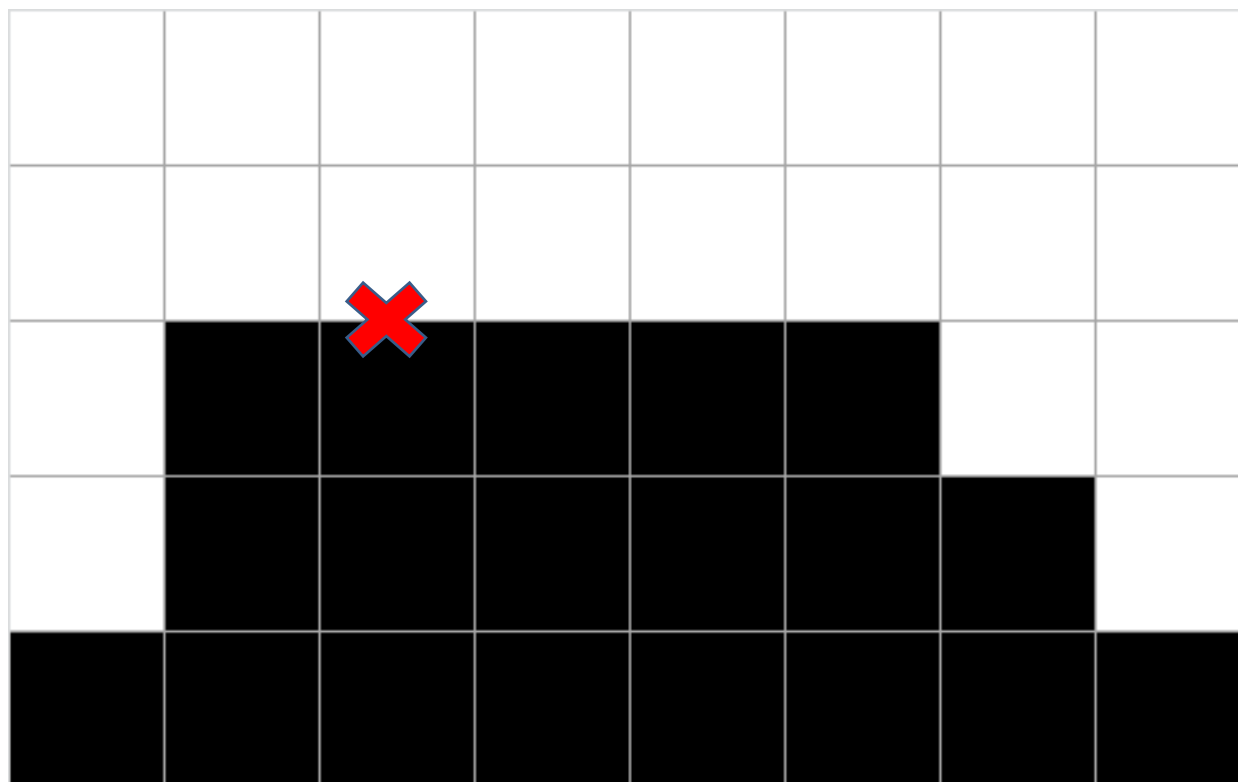
Implementação FXAA - Etapa 2

- Detector de bordas
 - Se detecta linhas verticais e horizontais através do algoritmo desenvolvido por Timothy Lottes.
 - Algoritmo considera a diferença de luminosidade entre os pixels para definir se o pixel em análise é uma borda ou não.
 - Utiliza uma série de limiares para limitar a detecção



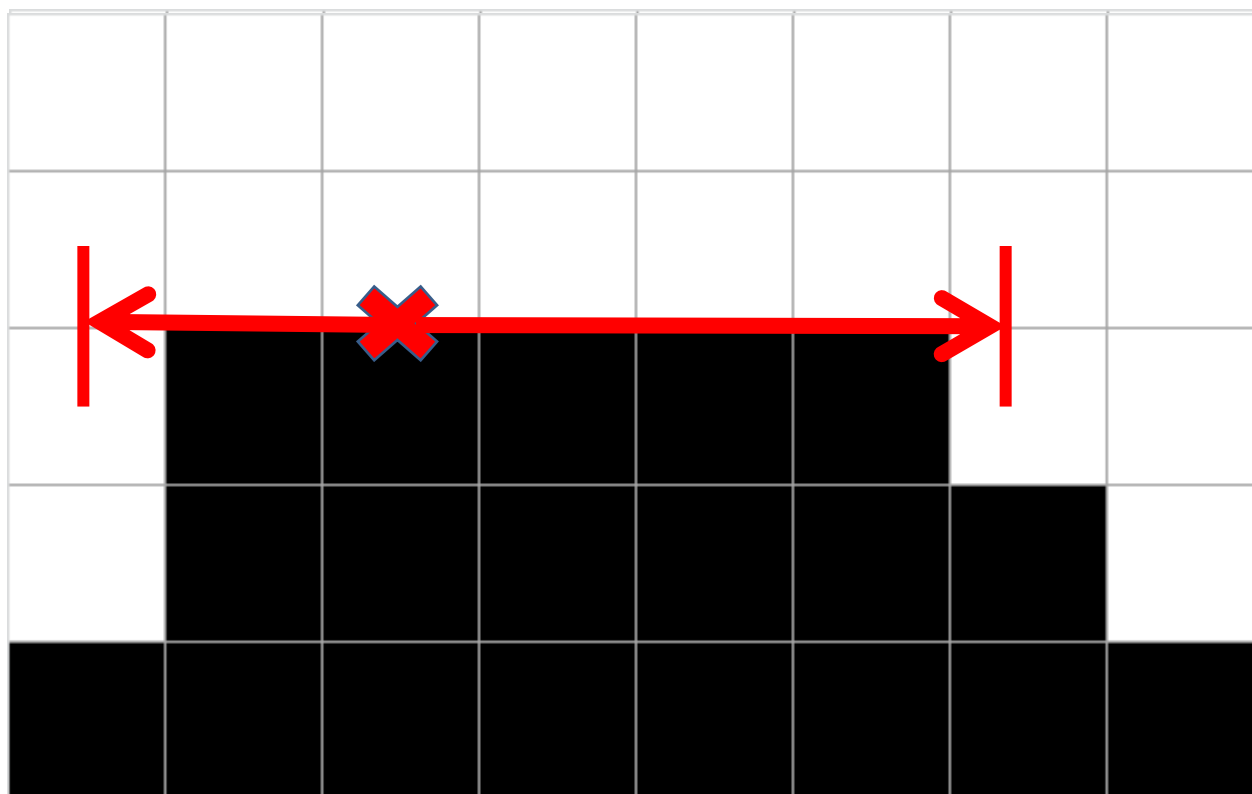
Implementação FXAA – Etapa 3

- Definição da orientação da borda (cima/baixo, direita/esquerda).



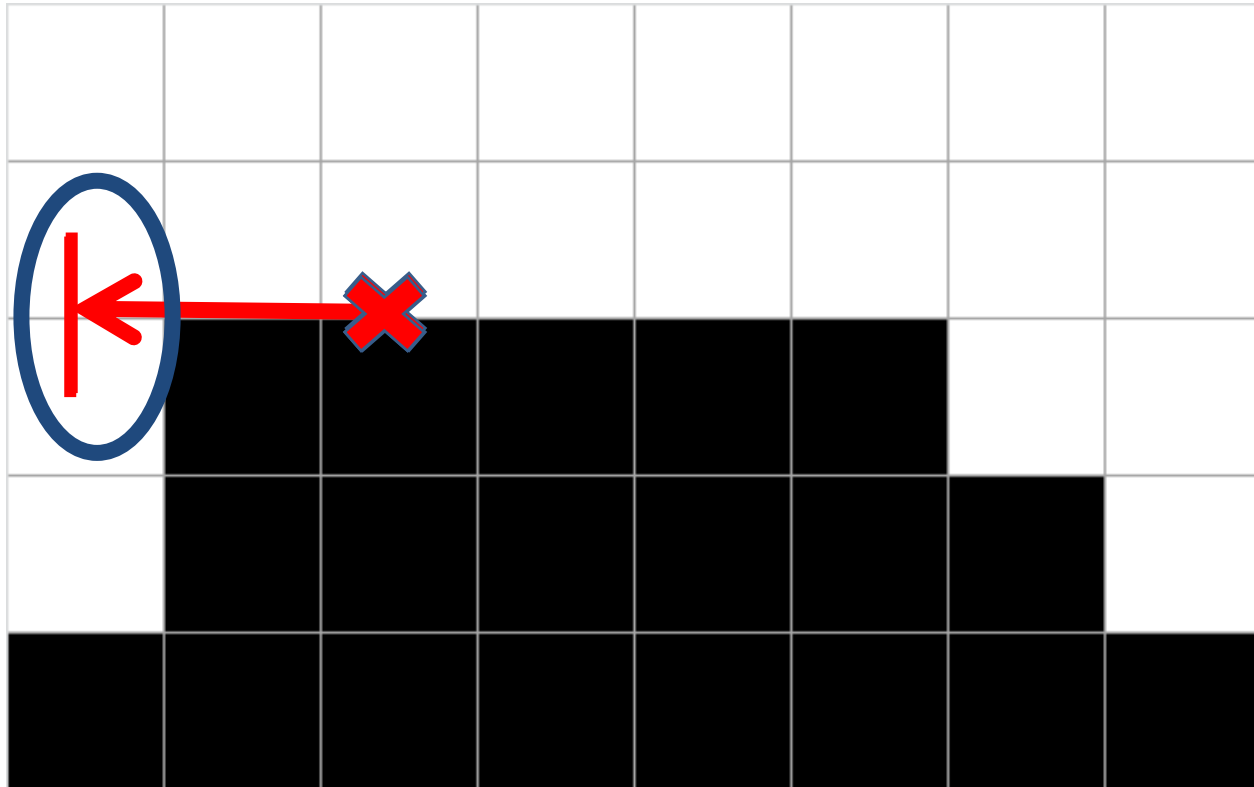
Implementação FXAA – Etapa 4

- A partir da orientação correta, se descobre o tamanho da borda.



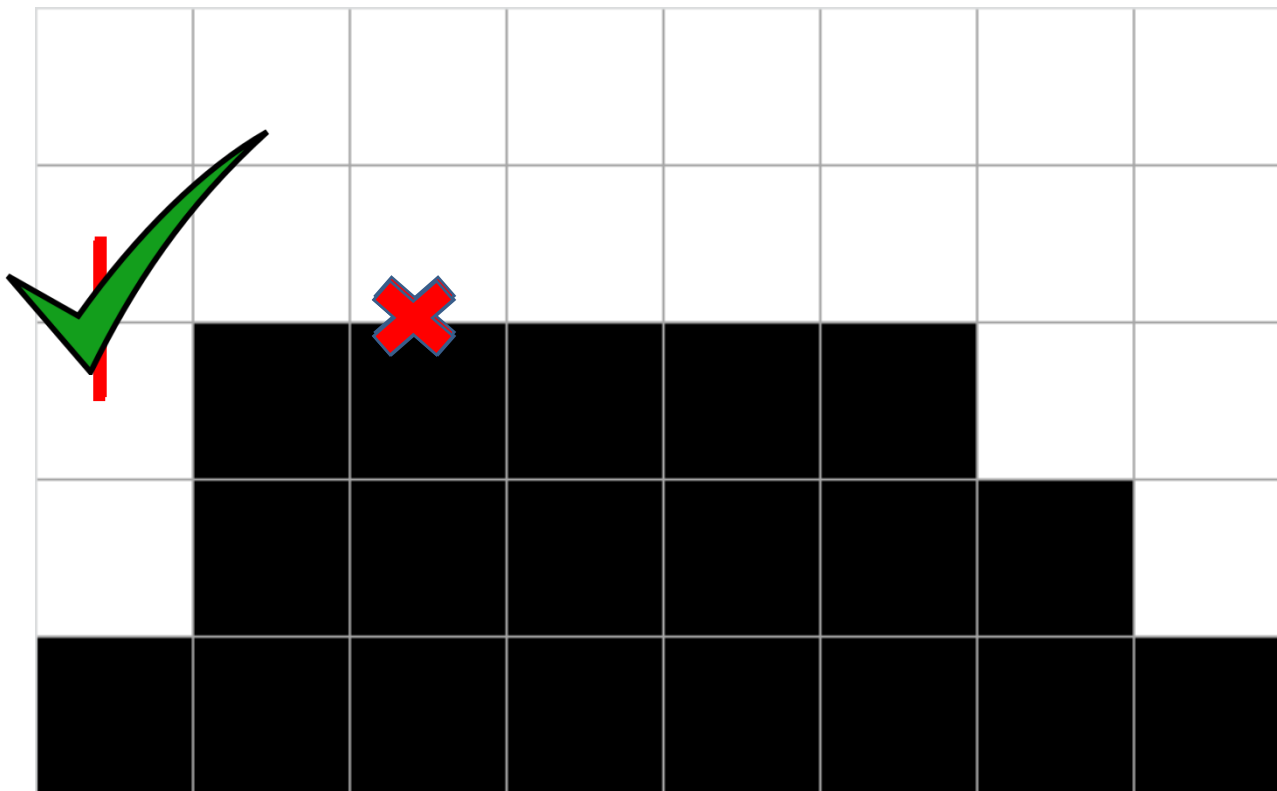
Implementação FXAA – Etapa 5

- Estima-se um *offset* inicial para a nova intensidade do pixel baseado na distância e intensidade dos *pixels*.



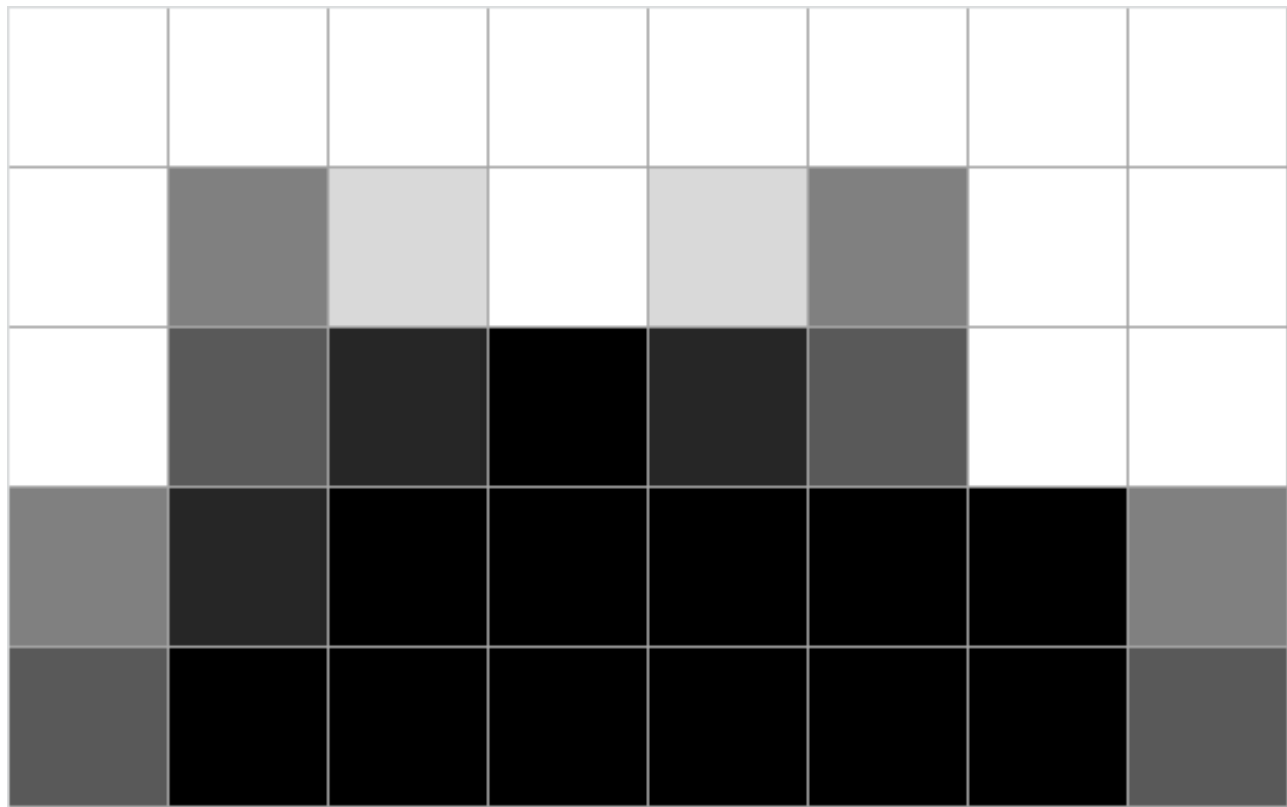
Implementação FXAA – Etapa 6

- Verifica-se se o offset desejado não está muito diferente do esperado, para evitar um borramento errado.



Implementação FXAA – Etapa 7

- Com o offset decidido, é modificado o valor do pixel em questão.



FXAA - Resultados

- Jogo: Path of Exile



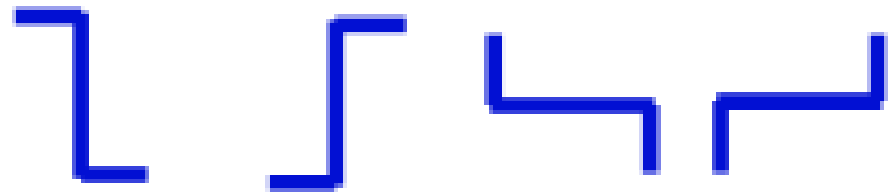
Implementação MLAA

- Pós-Processamento!
- Pode ser dividido em 3 etapas: detecção de bordas, detecção de formas e aplicação dos pesos.
- Algoritmo original não foca na detecção de bordas, por isso foi utilizado um detector de bordas Canny.

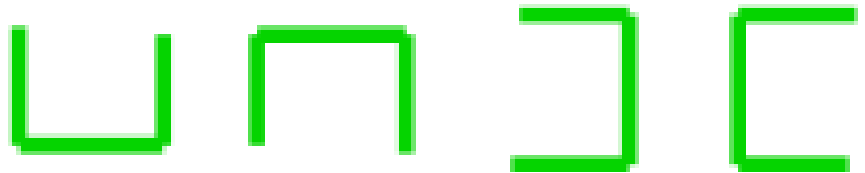
Implementação MLAA: Detecção de Formas

- O algoritmo detecta 3 tipos de formas: L (a), Z (b) e U (c)
- Cada forma específica pode ter orientações e tamanhos diferentes.

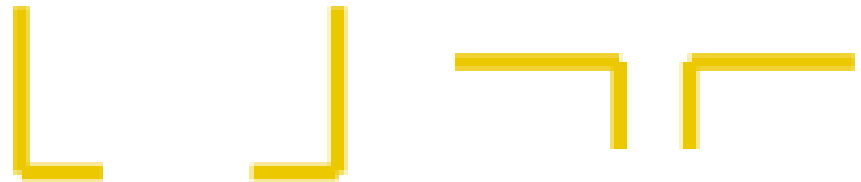
Z-shapes:



U-shapes:

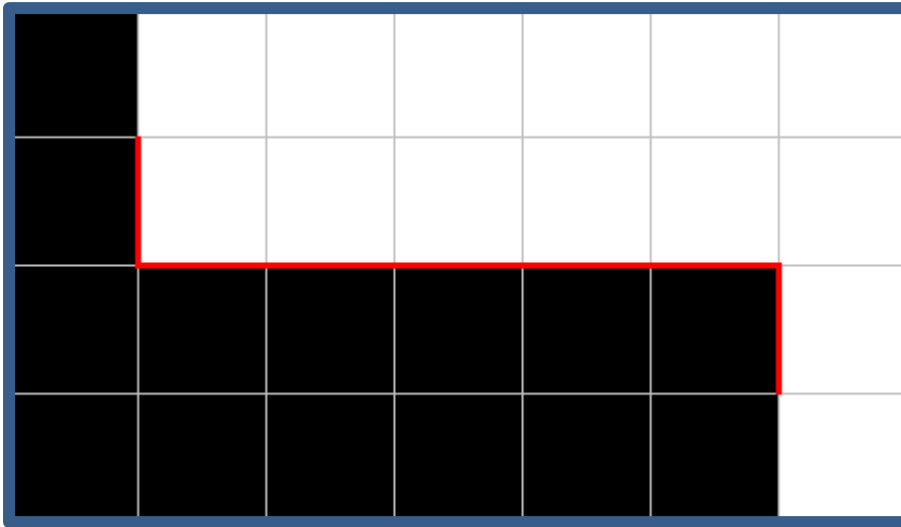


L-shapes:



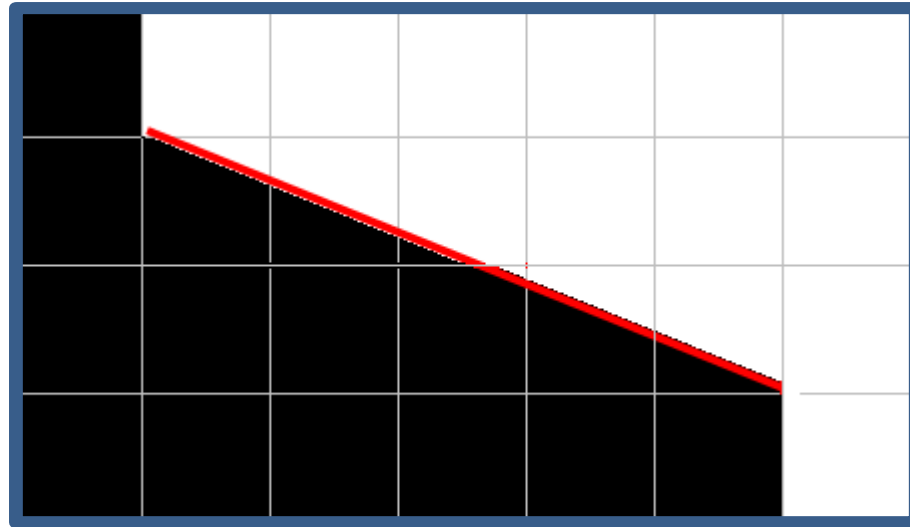
Implementação MLAA: Detecção de Formas

- Cada orientação possui uma análise própria.
- Com o tipo e tamanho da forma definida, é feita a análise de pesos.



Implementação MLAA: Aplicação dos Pesos

- É definido a área que fica abaixo/acima da linha detectada.

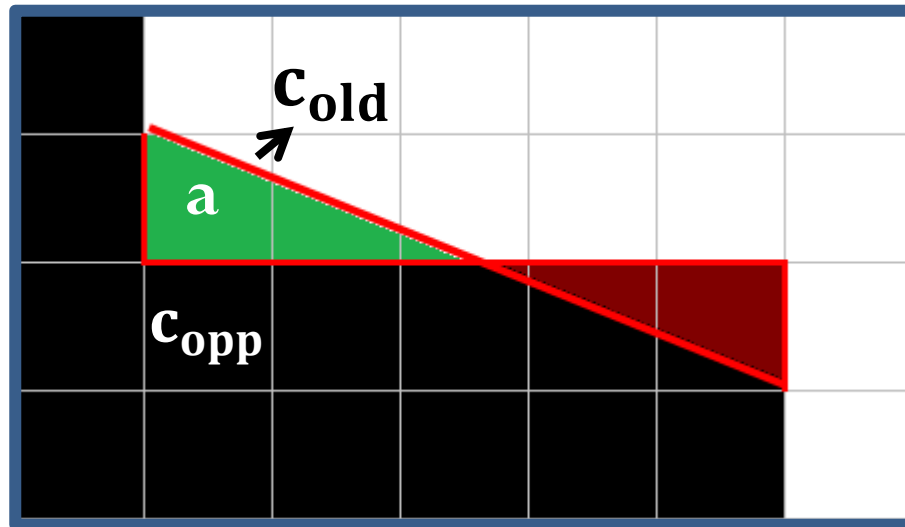


Como definir valores em posições
menores que um pixel???



Implementação MLAAA: Aplicação dos Pesos

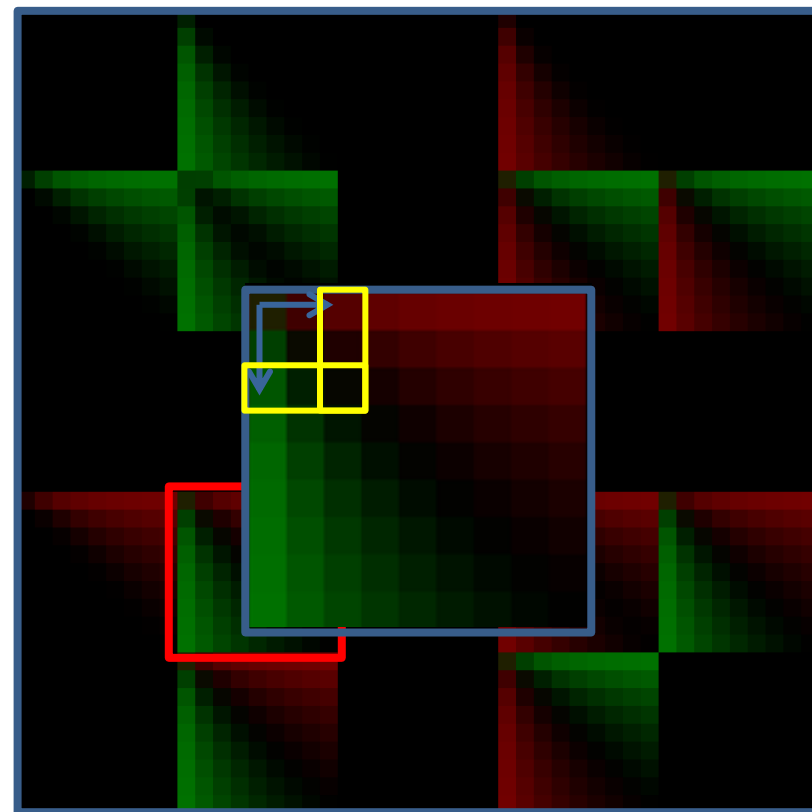
- Métodos são feitos em OpenGL.



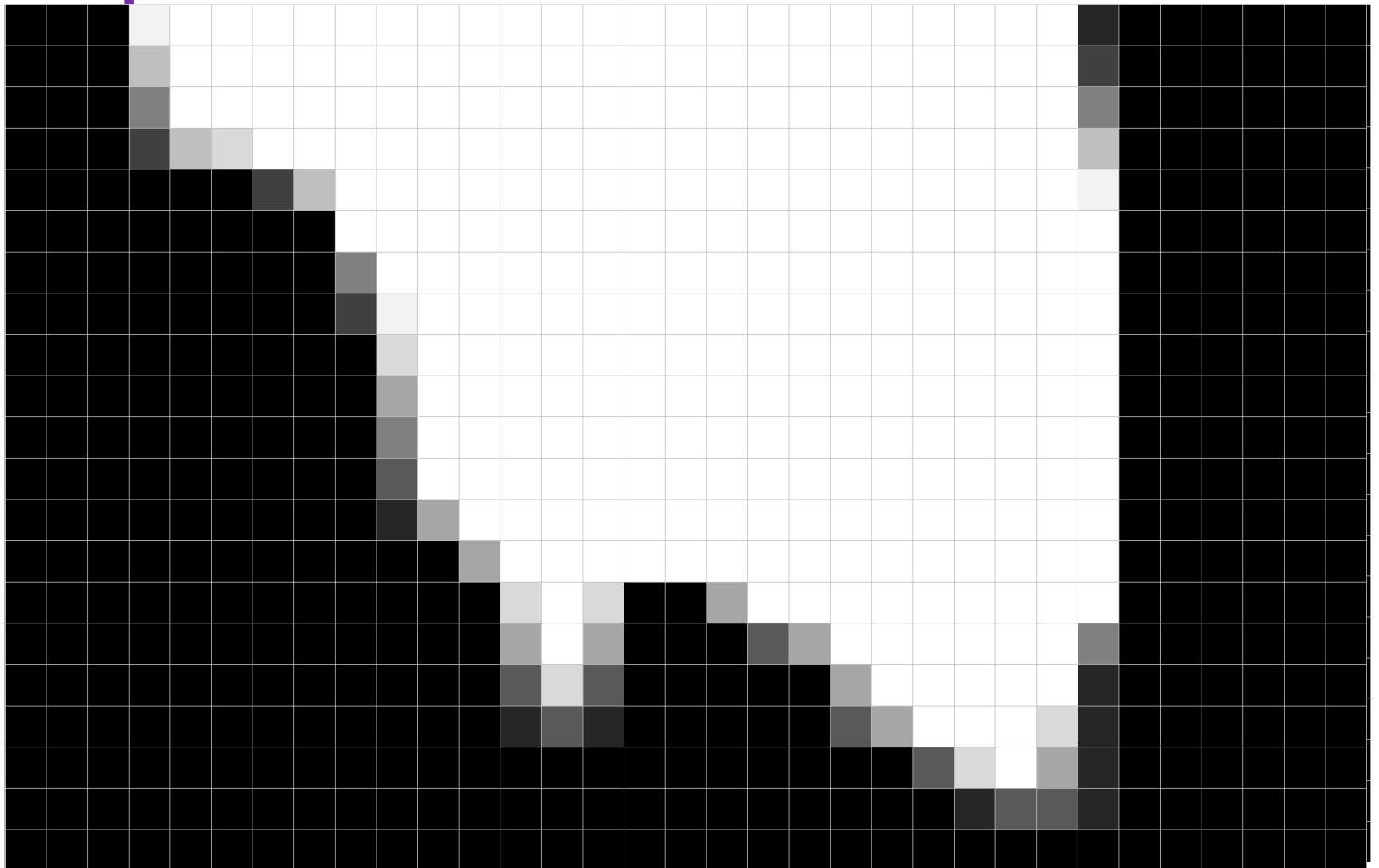
$$C_{new} = (1 - a) * C_{old} + a * C_{opp}$$

- Interpolação ainda é uma operação custosa.
- Existem formas finitas e tamanhos finitos (tamanho máximo).
- Solução: Usar pesos pré definidos!

Implementação MLAA: Aplicação dos Pesos



MLAA: Exemplo da Aplicação dos Pesos



MLAA - Resultados

- Exemplo: Borderlands Pre-Sequel;



Testes

- Foi feita a aplicação de todos os filtros nos jogos: Borderlands Pre-Sequel (2013), Dark Souls 3 (2016), Path of Exile (2013), Half Life 2 (2004) .
- Critério de escolha de jogos: estilo gráfico.

Testes



Resultados: Borderlands Pre-Sequel

Sem filtro



Generico



SSAA — 1



SSAA-2



FXAA

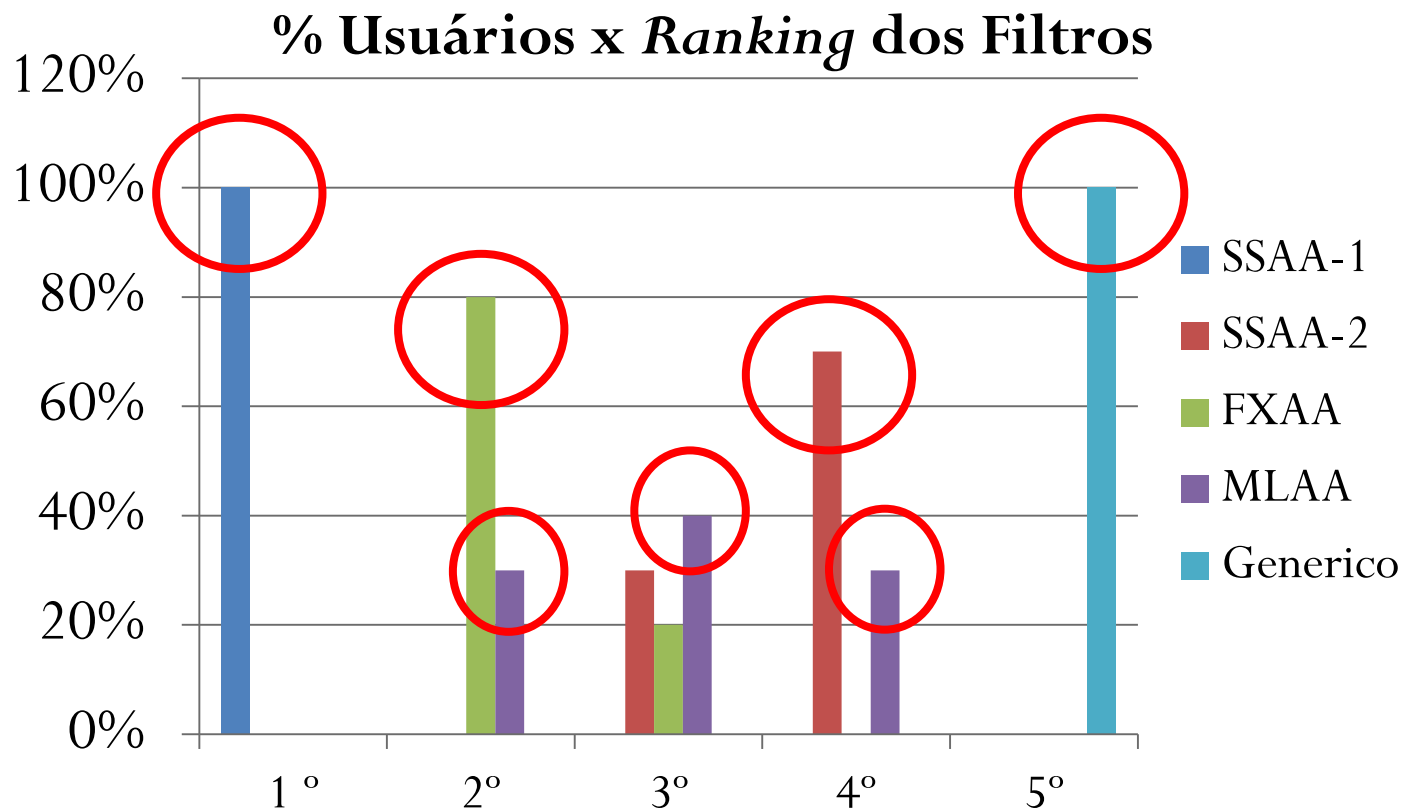


MLAA



Testes com usuários

- Mesmo conjunto de usuários: 10 pessoas.
- Para cada jogo, 6 imagens comparativas contendo todos os métodos.

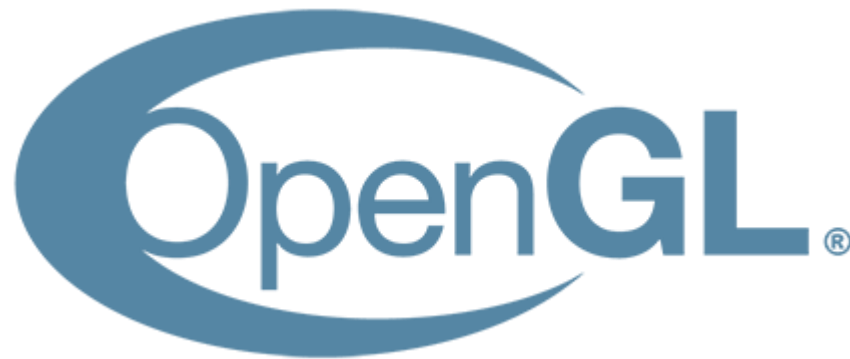


Conclusões

- Não existe “verdade universal” sobre qual é o melhor filtro.
- Qualidade de imagem é subjetiva.
- Grande necessidade de mais informações configuração dos jogos, indicando rapidamente diferenças gráficas e de performance.

Trabalhos Futuros

- Implementar os filtros desenvolvidos em OpenGL para ter informações de desempenho “reais”.
- Implementar mais filtros, tanto de pré-processamento quanto pós-processamento e analisar seus lados positivos/negativos.
- Desenvolver os filtros de textura.



Agradecimentos

- Professor Denis por possibilitar a realização deste trabalho.
- Aos colegas e amigos que disponibilizaram seu tempo pra responder perguntas e realizar testes.



Obrigado!

