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

Daniel Centeno Einloft
Faculdade de Engenharia de Computação
Pontifícia Universidade Católica do Rio Grande do Sul
Disciplina de Aplicação de Processamento Digital de Sinais
Porto Alegre, Rio Grande do Sul
daniel.einloft@acad.pucrs.br

Jean Larrocca
Faculdade de Engenharia de Computação
Pontifícia Universidade Católica do Rio Grande do Sul
Disciplina de Aplicação de Processamento Digital de Sinais
Porto Alegre, Rio Grande do Sul
jean.larrocca@acad.pucrs.br

***Abstract*—O problema de *aliasing* já é antigo na área de processamento digital de sinais. Quando se trata de computação gráfica, mais especificamente em jogos digitais, onde é necessário uma renderização em tempo real, o *aliasing* é um problema decorrente e é tratado a partir de uma série de filtros. Cada filtro possui características próprias, das quais não são divulgadas pelos desenvolvedores de jogos (salve algumas exceções), causando muitas dúvidas por parte dos usuários qual filtro deve ser escolhido. Este trabalho tem como objetivo fazer um estudo mais aprofundado nos filtros utilizados em jogos digitais dos últimos 11 anos, mostrando seu funcionamento, aspectos positivos e negativos. Para validar o estudo teórico, foi desenvolvido alguns destes filtros e testado com 10 pessoas, com o intúito de compará-los lado a lado e decidir até onde jogadores notam suas diferenças.**

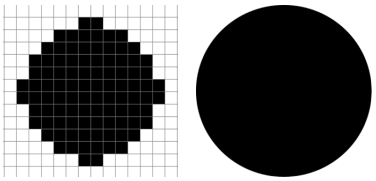
***Keywords*—Anti-Aliasing, Computação Gráfica, Jogos Digitais, Texture Filtering**

I. Introduction

A computação gráfica tem muita importância no mundo moderno, tanto na indústria do entreterimento, em jogos digitais e filmes, quanto no campo da pesquisa, em aplicações CAD e simulações. A necessidade da geração de modelos gráficos de alta qualidade está cada vez maior, e com o aumento do poder de processamento de placas gráficas (*Graphics Processing Unit*, ou GPU) e capacidade de memória, estamos dando saltos largos em direção de modelos foto realistas. Mesmo assim, os motores gráficos mais modernos ainda possuem grandes dificuldades em representar um objeto 3D de forma que se deseja em tempo real. Isso se dá pela imprecisão dos computadores ao representar um objeto real num ambiente virtual, devido à quantidade limitada de o pontos de representação, comparado com a infinidade que temos no mundo real. Esta diferença define a importância do estudo de processamento digital de sinais, onde é necessário representar um sinal analógico através de sinais digitais.

Quando deseja-se representar um sinal analógico periódico, é necessário levar em consideração quantas amostras desse sinal será utilizado para representa-lo digitalmente. Se for utilizado uma taxa de amostragem suficiente, o sinal digital possuirá um comportamento muito semelhante ao comportamento analógico. Por outro lado, quando existe poucas amostras, a representação ficará falha, não possuindo o comportamento esperado, havendo grandes perdas de informações. Este problema de amostragem, chamado de *aliasing*, foi estudado por Harry Nyquist, que definiu a chamada "Taxa de amostragem de Nyquist". De acordo com Nyquist, a frequência de amostragem precisa ser duas vezes o valor da frequência do sinal para que não haja perda de informações [1].

Este problema também é muito comum na computação gráfica. Quando não existem amostras, no caso, *pixels*, para representar uma imagem (sinal bidimensional), esta fica com uma série de descontinuidades, perdendo a forma original [2]. O efeito do *aliasing* em imagens fica evidente em duas situações: nas bordas de imagens [3] e no desenho de texturas bidimensionais [4]. No caso das bordas, o efeito de *aliasing* acarreta bordas "serrilhadas" (*jagged edges*), que deformam o objeto. A Figura 1 demonstra este fenômeno. Já no caso de texturas, o efeito possui duas consequências: na magnificação da imagem (quando ocorre um *upscaling* na textura) e minificação (quando ocorre um *downscaling* na textura) [5].



(a) Com *Aliasing* (b) Com *Anti-Aliasing*

Fig. 1: Circulos com e sem problema de serrilhamento.

Para resolver estes problemas, existem duas soluções: a) aumentar a resolução da imagem para conseguir representar os objetos de forma correta, devido ao aumento de *pixels*, ou b) aplicar uma série de filtros que vão aplicar uma suavidez na imagem, melhorando a qualidade de texturas (filtros de textura, ou TF) e bordas (filtros de *Anti-Aliasing*, ou AA).

Em jogos digitais de computador, existem várias opções de customizações gráficas que podem ser aplicadas, dando a liberdade para o usuário de escolher qual filtro deseja utilizar. Porém, na grande maioria dos jogos, não existe nenhuma informação sobre o que são estes problemas e como estes

filtros podem resolve-los, sendo que cada um possui vantagens, desvantagens e diferentes performances.

O objetivo deste trabalho consiste em estudar os filtros mais utilizados em jogos digitais e a forma que eles são apresentados para o consumidor. Além disso, foi desenvolvido cinco filtros diferentes, todos de AA, para que sejam testados com usuários (todos jogadores) e analisar até onde este tipo de detalhe gráfico é notado.

II. Motivação

Diferente dos consoles como Playstation e Xbox, jogos digitais de computadores são muito famosos pelo sua versatilidade, possibilitando aos jogadores escolherem se querem melhorar ou piorar a qualidade de texturas, sombras, filtros de *Anti-Aliasing*, filtros de texturas, etc. A Figura 2 mostra esta tela de configuração, retirada do jogo World of Warcraft. A vantagem dessa alta customização se dá no fato que jogadores que não possuem o computador mais moderno consigam rodar o jogo, utilizando as configurações mínimas.

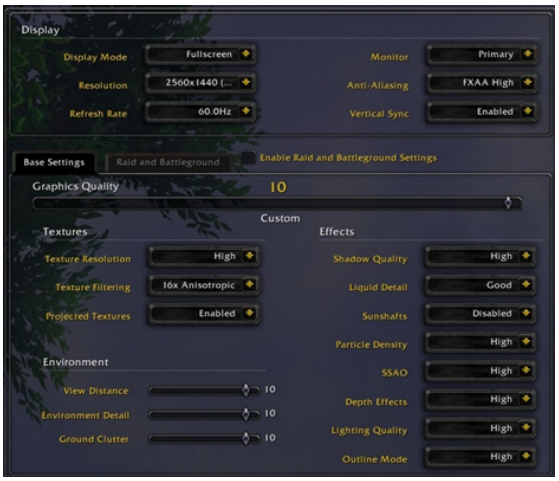


Fig. 2: Exemplo da tela de configuração gráfica do jogo World of Warcraft.

Porém, por mais que algumas destas informações possuem um significado intuitivo (como ”qualidade/resolução de texturas”), existem alguns casos que o nome da configuração não possui significado óbvio, como filtros de *Anti-Aliasing* e texturas. Quando um jogador se depara com uma configuração desconhecida, ele pode ignora-la ou procurar o significado na internet. Todavia, nenhuma dessas soluções são ideais, pois se é possível modificar os gráficos do jogo, o mínimo é ter informações de fácil acesso ao usuário.

Neste contexto, foi feito uma pesquisa envolvendo jogos dos últimos 11 anos (2007-2017), onde foi estudado como é feita a divulgação destas informações. Dos 110 jogos pesquisados, apenas 8% possuem informações referentes a cada tipo de filtros de AA e TF utilizados, e 7% explicam apenas o que significa a configuração. O restante dos jogos, 85%, não possui nenhuma informação.

A partir daí, foi feita outra pesquisa, envolvendo 10 jogadores de computador, onde foi perguntado se tinham conhecimento do significado de filtros de AA/TF, e se sabiam a diferença entre os filtros mais utilizados. Desta pesquisa, 80% sabiam definir o que significava *Anti-Aliasing*, mas 100% não sabiam o que era filtro de textura. Além disso, 90% dos jogadores não sabiam diferenciar os filtros, usando apenas o ordenamento destes filtros na tela como método de escolha. Por fim, todos acham que os desenvolvedores não divulgam estas informações corretamente, apontando que o ideal seria uma identificação concisa de cada filtro, mostrando uma comparação visual e indicando a performance de cada um.

Desta forma, a proposta deste trabalho consiste em fazer um estudo mais aprofundado nas tecnologias de *Anti-Aliasing* e filtros de textura, mostrando quais são os filtros mais utilizados em jogos digitais, quais são os seus funcionamentos básicos, suas vantagens, desvantagens e quais possuem maior e pior performance. Além disso, será apresentado cinco filtros desenvolvidos, feitos para validar as informações teóricas estudadas e para identificar quais são os melhores filtros para características gráficas diferentes.

III. Trabalhos Relacionados

Grande parte da motivação deste trabalho consiste no estudo mais aprofundado dos filtros de *Anti-Aliasing* e de textura, mostrando qual as principais diferenças entre eles, em termos de funcionamento e performance, e quais possuem os melhores resultados. É importante ressaltar que estes melhores resultados vem da opinião de diversos desenvolvedores, fazendo comparações diretas dos métodos com um alto nível de *zoom*. Porém, como vai ser visto mais para frente, a percepção dos usuários não necessariamente segue a visão dos desenvolvedores, pois cada pessoa possui um olhar subjetivo sobre o assunto. Alguns jogadores notam mais bordas serrilhadas que texturas borradas, e vice-versa, por exemplo.

Outro fator importante a ser considerado é a grande quantidade de filtros de AA desenvolvidos por diversos grupos de pesquisa e desenvolvedoras como a *n*Vidia e AMD. Nesta classificação, existem pelo menos dezoito filtros estudados. Como a base da pesquisa consiste em filtros utilizados em jogos digitais, foi considerado apenas os filtros utilizados nos últimos onze anos (2007-2017). Da mesma pesquisa com jogos feitas anteriormente, os filtros de AA mais utilizados são o *Super-Sampling Anti-Aliasing* (SSAA), *Multi-Sampling Anti-Aliasing* (MSAA), *Coverage Sampling Anti-Aliasing* (CSAA), *Enhanced Quality Anti-Aliasing* (EQAA), *Temporal Anti-Aliasing* (TXAA), *Fast-Approximate Anti-Aliasing* (FXAA), *Morphological Anti-Aliasing* (MLAA) e *Enhanced Sub-pixel Morphological Anti-Aliasing* (SMAA). Para os filtros de textura, os mais utilizados são o filtro Bilinear, Trilinear e Anisotrópico.

A. Filtros de Anti-Aliasing

De acordo com Jiang [6], existem várias classificações de filtros de AA. Porém, em jogos digitais, a classificação mais comum consiste em separá-los em duas categorias: Filtros de Pré-Processamento e Pós-Processamento. O termo processamento, neste caso, é referente ao processo de renderização do jogo. Sendo assim, os filtros podem ser aplicados durante (Pré-Processamento) ou após (Pós-Processamento) a renderização.

1) Filtros de Pré-Processamento

Os filtros de Pré-Processamento, em geral, abordam o problema se focando no pricípio da amostragem de Nyquist, aumentando o número de amostras para melhorar a qualidade dos gráficos como um todo. Para esta classificação, encontram-se os filtros SSAA, MSAA, CSAA, EQAA e TXAA. Mesmo seguindo o mesmo princípio, cada filtro trata o problema de formas diferentes.

O *Super-Sampling Anti-Aliasing* é o método mais antigo e considerados por muitos o mais eficaz na remoção de bordas serrilhadas, abordando o problema de forma muito simples: aumentando a resolução durante a renderização e fazendo um *downsampling* para a resolução desejada. Esta técnica também permite selecionar a resolução de renderização, como por exemplo, se o usuário selecionar SSAA 4x e a resolução do monitor for FullHD (1080p), a GPU irá renderizar tudo em quatro vezes (4K). As configurações mais comuns variam de 2x à 8x [6].

O SSAA não define um tipo de amostragem específica, dando liberdade aos desenvolvedores criarem novas técnicas de *downsampling*. Os métodos mais comuns consistem no *Ordered Grid Super-Sampling* (OGSS), *Rotaded Grid Suer-Sampling* (RGSS) [7] e *High-resolution antialiasing* (HRAA, ou formato Quincunx) [8], que podem ser visualizados na Figura 3 mostra estes padrões.

(a) OGSS (b) RGSS (c) HRRA

Fig. 3: Diferentes padrões de amostragens para *Super Sampling*.

Porém, por mais que seja eficaz, o SSAA peca pela quantidade de memória e banda de memória utilizada [6], [9], o que fez a tecnologia entrar em declínio nos últimos anos. Mesmo com um suporte nativo das GPUs da *n*Vidia e consoles Xbox One X e Playstation 4 Pro, os desenvolvedores de jogos optam pela utilização de um outro filtro com um resultado similar, que reduz a quantidade de memória utilizada: o MSAA. Este filtro utiliza o mesmo princípio que o SSAA, porém detecta as bordas dos polígonos e aplica a filtragem apenas nestas regiões. Os padrões de amostragem são os mesmos que o SSAA, podendo, da mesma forma, ter resultados diferentes [6], [10].

Em 2008, a *n*Vidia apresentou uma nova tecnologia, parecida com o MSAA, porém mais rápida, chamada CSAA. Em 2011, a AMD desenvolveu uma resposta à este filtro, com o seu EQAA. Disponível apenas nas GPUs das suas respectivas desenvolvedoras, ambos os algoritmos funcionam a mesma forma. Ambas ganham em performance através da adição de mais amostras em geral, possuindo uma precisão maior, porém reduzindo o número de amostras de cores na amostragem. Isso se dá através de uma máscara padrões de borda, encontrando apenas as variações de cores e utilizando somente elas para os cálculos. Desta forma, tanto o CSAA quando EQAA em 8x possui a mesma performance que o MSAA 4x [11], [12].

O último filtro de pré-processamento é o TXAA, também da *n*Vidia, lançado em 2012. Este filtro baseia-se na análise temporal dos frames, combinando com características espaciais do MSAA. Similar à filtros de AA utilizados na computação gráfica em filmes, esta análise temporal consiste na utilização de um conjunto de amostras tanto do frame atual quanto anteriores, melhorando a qualidade de objetos em movimento, mantendo a performance similar ao MSAA [13].

2) Filtros de Pós-Processamento

Como os filtros de pré-processamento consumem muita memória, a partir de 2009 diversos desenvolvedores começaram a apresentar um novo tipo de AA, deixando de tratar o problema a partir do número de amostras, passando para uma análise de imagens após os gráficos serem renderizado, examinando formas não de polígonos, mas *pixels*. Dos diversos filtros desenvolvidos, destacam-se três: o FXAA, MLAA e SMAA.

O FXAA, desenvolvido em 2009 por Timothy Lottes, da *n*Vidia, aplica técnicas de *blurring* de acordo com o tamanho e orientação da borda. Utilizando um algoritmo próprio, o FXAA procura bordas verticais e horizontais e analisa a sua orientação e comprimento. Através destas informações, é calculado um *offset* de cores para o *pixel* em questão, dando o efeito de borda suavizada [14]. Este processo será explicado mais detalhadamente na seção IV.

Em 2011, pesquisadores da universidade de Zaragoza, Espanha, liderados por Jorge Jimenez, desenvolveram o MLAA, que apresenta uma análise de formas mais robusta que o FXAA. Este algoritmo procura bordas no formato de L, U e Z, aplicado uma suavização de acordo com sua forma e tamanho. Outra característica do MLAA consiste na utilização de uma série de pesos pré-difinidos, aumentando drasticamente sua performance [15].

No ano seguinte, o mesmo grupo de Jimenez [9] desenvolveu o SMAA, uma versão melhorada do algoritmo anterior. O SMAA conta com uma série de melhorarias sobre o algoritmo anterior, adicionando novas detecções de formas, possibilitando detecção de grandes bordas diagonais, somado com um novo conjunto de pesos. Além destas melhorias, o SMAA utiliza em conjunto o MSAA, e conta com a possiblidade da utilização de um algoritmo temporal (o SMAAT), adicionando uma grande estabilidade temporal e espacial na imagem final.

B. Filtros de Textura

Texturização é o processo de aplicação de texturas em objetos 2D ou 3D durante a renderização. Para isso, desenvolvedores fazem o mapeamento de textura através de Mipmaps, que consistem numa sequência de imagens da mesma textura, em forma piramidal, que gradualmente vão diminuindo de resolução (Figura 4 (a)). Os Mipmaps são muito utilizados em jogos digitais pois é muito mais vantajoso acessar uma textura pequena pré-calculada do que reduzi-la ”manualmente”. A utilização de Mipmaps para o processo de filtragem de textura é muito importante, pois é sobre eles que os cálculos serão realizados [16].

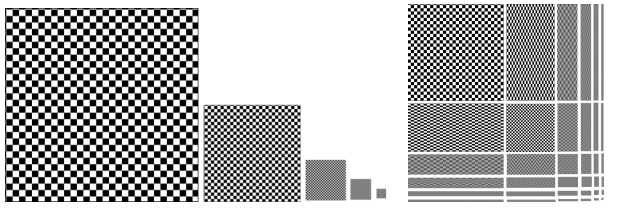
Nos jogos digitais, quando a posição do jogador está muito próximo de um objeto, a textura é mostrada numa

resolução maior que ela foi criada, gerando o fenômeno de magnificação. A textura ”esticada” acontece pois existem poucos *texels* (unidades de texturas) para serem representados em alta resolução. Note que, para este caso, o problema de *aliasing* se dá não pela quantidade reduzida de *pixels*, mas pela quantidade reduzida de *texels*. Por causa deste problema, foi desenvolvido uma série de filtros de textura, para amenizar efeitos de *aliasing* tanto por poucos *texels* quanto por *pixels* [17].

Um dos filtros mais famosos, e muito presente em jogos digitais, consite no filtro Bilinear [5], [17], que é utilizado para o *upsampling* de texturas. A partir das texturas do Mipmap, o filtro aplica uma interpolação bilinear de 4 em 4 amostras, criando um efeito de *zoom* na imagem.

Contudo, a utilização de um filtro Bilinear em diversos pontos diferentes de uma imagem (em paredes, por exemplo) adicionam um novo problema: a intersecção de resoluções diferentes num modelo 3D ficam com um aspecto destoante, mostrando claramente onde termina uma resolução de textura e onde começa a próxima. Para resolver este problema, foi introduzido filtro Trilinear [17], que utiliza o mesmo princípio do filtro anterior, mas com uma adição: a interpolação é feita não só dentro, mas também entre Mipmaps consecutivos, tornando a transição de uma resolução para a outra de forma mais suave.

Para magnificação, os dois filtros citados acima resolvem bem o problema, mas para minificação, as texturas à distância permanecem borradas, mesmo com a utilização destes filtros. Aliás, estes filtros são classificados filtros isotrópicos, utilizados mais para formas uniformes (como quadrados e retângulos). Para texturas anisotrópicas (texturas com o comportamente diferente dependendo da direção), como acontece, por exemplo, numa textura de uma rua vista em ângulo, se utiliza o filtro anisotrópico (ou AF).



(a) Mipmap isotrópico (b) Mipmap anisotrópico

Fig. 4: Dois tipos de Mipmaps utilizados na filtragem de texturas.

Mais robusto que os outros filtros, o AF é o filtro mais utilizado dentro dos 110 jogos pesquisados, tornando texturas distantes borradas em formas bem definidas. Para isso, o AF utiliza o mesmo conceito de Mipmaps, porém com uma diferença importante: ao invés de variar a resolução uniformemente, agora a resolução varia nos dois eixos, aumentando o número de padrões e adicionando texturas pré-calculados mais ”esticadas” (Figura 4 (b)). Desta forma, a fluidez de texturas vistas em ângulo melhora drasticamente, acrescentando uma maior qualidade na imagem [18].

IV. Implementação de Filtros

A segunda parte do projeto consiste no desenvolvimento de alguns dos filtros mostrados anteriormente, para analisar até onde jogadores detectam as diferenças entre eles e quais filtros são recomendados para estilos gráficos específicos. Como os filtros de textura são executados durante a renderização, o foco desta seção encontra-se nos filtros de AA, pois o estudo de técnicas de renderização seria um grande *overhead* do trabalho, fugindo da proposta inicial. Desta forma, foram desenvolvidos cinco filtros diferentes: um filtro baseado em conceitos vistos em sala de aula, através da aplicação de um filtro Gaussiano, duas versões do SSAA, contando com dois tipos de amostragens diferentes, o FXAA e o MLAA.

A proposta do trabalho considerava o desenvolvimento em Matlab, mas implementação dos filtros foi feito na linguagem Python, realizando todas as operações manualmente, com excessão do filtro genérico, onde foi utilizados funções próprias do OpenCV [1], disponíveis também no Matlab de forma similar. Esta decisão foi tomada puramente pela facilidade de desenvolvimento, já que ambas linguagens são baseadas em *script*, eliminando uma comparação justa de performance (pois estes tipos de algoritmos são desenvolvidos OpenGL, utilizando GPU). Os resultados com comparações serão mostradas na seção V.

1) Filtro Genérico

Em sala de aula, foi visto uma série de filtros utilizados no processamento de imagens que detectam bordas e aplicam o borramento de fotos (trabalho E11). Desta forma, foi decidido desenvolver um filtro ”genérico” que, teoricamente, ajudaria a diminuir os serrilhados das fotos, baseando-se em métodos clássicos de processamento de imagens. A partir daí, foi realizado diversos testes em imagens para encontrar o melhor filtro possível para este propósito. Para isso, foi pesquisado e selecionado os filtros de mediana e guassiano.

Dentre estes dois filtros, notou-se que o resultado da aplicação do filtro de mediana piorou a qualidade da imagem, mantendo os serrilhados e adicionando um efeito ”emborrachado”, deteriorando drasticamente as texturas. Já o filtro gaussiano possuiu um resultado melhor, gerando numa imagem final fora de foco. Para reduzir o efeito na imagem inteira, o filtro foi aplicado apenas nas bordas, detectadas pelo algoritmo de Canny.

2) Super Sampling Anti-Aliasing (SSAA)

Os próximos dois filtros desenvolvidos consistem em versões diferentes do SSAA, utilizando dois tipos de amostragens diferentes: o *High-resolution antialiasing* (HRAA, ou formato Quincunx), que será referenciado neste trabalho como SSAA-1, e *Ordered Grid Super-Sampling* (OGSS), referenciado como SSAA-2. Como o SSAA é um filtro de pré-processamento, foi utilizado um ”truque”: foi executado e gravado os jogos numa resolução maior e depois comprimidos para uma resolução menor (metade da resolução de gravação, algoritmo rodando em 2x).

Para cada um dos algoritmos, foi utilizado um passo igual a 2. Isto significa que para o SSAA-1 foi analisado 2 *pixels* para todos os lados do *pixel* analisado, enquanto o SSAA-2

[1] http://opencv.org/

define uma janela com dois *pixels* à direita do pixel analisado. Para ambos os casos, foi feito uma média aritmética simples das amostras coletadas e aplicadas na posição resultante na imagem final.

3) Fast Approximate Anti-Aliasing (FXAA)

A implementação do FXAA baseia-se no *tech report* de Timothy Lottes, onde foi analisado o algoritmo e os trechos de código disponibilizados [14] e na implementação de Simon Rodriguez, PhD do *Institut National de Recherche en Informatique et en Automatique* [2].

O primeiro passo consiste na conversão da imagem RGB para um canal de luminosidade, utilizando o padrão CCIR 601 [3]. Em seguida, é feita uma conversão da escala da imagem, de 0-255 para 0-1. Esta conversão é feita por característica do algoritmo, que utiliza uma série de constantes nesta escala. No segundo passo, é feita a detecção de bordas, onde se define uma série de limiares (já estabelecidos pelo algoritmo) para definir se a borda é horizontal ou vertical a partir da diferença de intensidade de luminosidade entre o *pixel* em questão com a sua vizinhança

Caso o *pixel* não seja borda, o resto do processamento não é realizado e o próximo *pixel* passa a ser analisado. Caso contrário, é executado o terceiro passo, no qual é analisada a orientação da borda. Para isso, é examinado os pixels em 90 graus em relação à direção da borda, identificando se o *pixel* está acima/abaixo da borda horizontal ou à direita/esquerda da borda vertical.

Com a orientação definida, o algoritmo começa a procurar os limites da borda, analisando a diferença de luminosidade entre os pixels nas duas direções. Examinando a menor distância, é calculado um *offset* inicial, que indicará o quanto o *pixel* em questão irá mesclar sua cor de acordo com a orientação detectada. Se o *offset* for 0.80, por exemplo, será feito uma soma de intensidades onde 20% da cor do *pixel* atual com 80% da cor do *pixel* à +/- 90 graus (dependendo da orientação).

Caso a aplicação do *offset* resulte num valor muito diferente da vizinhança local do *pixel*, a operação é ignorada, pulando a análise para o próximo *pixel*. Ao terminar a operação, o algoritmo é executado no resto da imagem.

4) Morphological Anti-Aliasing (MLAA)

O último filtro desenvolvido consiste na implementação do MLAA de Jorge Jimenez (2011). A principal contribuição do algoritmo foi a introdução de um detector de formas de bordas, que são analisadas pelo algoritmo e servem como um mapa para a aplicação das novas das cores. O algoritmo pode ser explicado dividindo-o em três partes: detecção de bordas, detecção de formas e aplicação dos pesos de borramento. Como a detecção de bordas é uma parte pequena do algoritmo, não sendo a contribuição principal, para evitar atrasos no desenvolvimento foi utilizado o algoritmo de detecção Canny.

A segunda etapa, então, consiste na detecção de formas, que podem ter três formatos distintos: forma em L, U e Z. Cada uma destas formas pode apresentar quatro tipo de orientações, variando para cima, baixo, direita e esquerda, definindo a direção borramento. Considere a Figura 5, onde é mostrado a detecção do padrão Z numa borda.

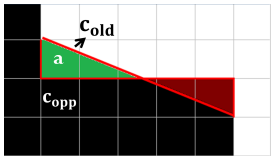


Fig. 5: Padrão Z com regiões para aplicação de peso.

Das extremidades da forma encontrada, é traçado uma linha que às une, definindo uma região abaixo e acima da borda. A região verde significa que o borramento será realizado misturando com as cores dos *pixels* de baixo, em quanto a região em vermelho indica o oposto. Enquanto o FXAA utiliza um *offset* para a definição de cores, o MLAA utiliza a área da região cortada, estabelecendo a seguinte relação para cálculo da nova cor:

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

Para calcular uma área menor que um *pixel*, seria necessário fazer uma interpolação. Porém, é uma operação custosa para realizar em todos os *pixels* em todas as bordas detectadas. Como existe um número finito de formas e um tamanho máximo estabelecido para as formas, existe um número finito de padrões, podendo utilizar pesos pré calculados para reduzir o *overhead*. A Figura 6 mostra a representação gráfica dos pesos, onde cada forma possui seus respectivos valores, com comprimentos indicados pelo gradiente das cores.

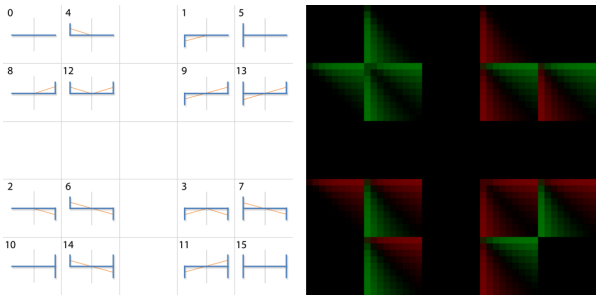


Fig. 6: Tipos de padrões com seus respectivos pesos.

A partir dos pesos pré-calculados, finalmente é aplicado as intensidades nos respectivos *pixels* e o algoritmo passa a analisar o resto da imagem.

V. Resultados

Com o término do desenvolvimento dos filtros, selecionou-se quatro jogos para executar os testes, sendo eles: Borderlands Pre-Sequel (2013), Dark Souls 3 (2016), Half Life 2 (2004) e Path of Exile (2013). O critério de seleção para cada um destes títulos foi o estilo gráfico, onde o primeiro possui características mais "desenhadas", com bordas bem definidas, o segundo possui um contraste grande entre cores claras e escuras, o terceiro, sendo mais antigo, possui modelos com menos detalhes e o último possui uma visão isométrica, mostrando todos o cenário, personagens e objetos de forma reduzida. Para cada jogo, foi feito uma filmagem de 30

[2] http://simonrodriguez.fr/
[3] https://goo.gl/zhDtFJ

segundos, onde foram retirados 6 *frames* escolhidos à mão e aplicado os cinco filtros. O software utilizado para as filmagens foi o Fraps [4]. A Figura 7, mostra os resultados das imagens filtradas do jogo Half Life 2, com a utilização de um *zoom* numa região que possui uma grade.

(a) Original (b) Genérico (c) SSAA-1 (d) SSAA-2 (e) FXAA (f) MLAA

Fig. 7: Resultados de todos os filtros no jogo Half Life 2.

Como pode-se ver, o filtro que utiliza a função Gaussiana (b) borra bastante a imagem, perdendo bastante a qualidade original. O SSAA-1 (c) e o FXAA (e) adicionam também um pouco de *blur*, mas de uma forma bem mais reduzida que o filtro anterior, sem deformar muito o formato da grade. O MLAA (f) não adiciona *blur*, porém deforma levemente o aspecto da figura. Por fim, o SSAA-2 (d) também não adiciona *blur* e ainda mantém bastante o formato original, parecendo, a princípio, o melhor resultado.

Para validar os resultados, foi mostrado as 24 imagens para os 10 usuários, onde foi comparado os filtros lado a lado. Esperava-se que os resultados fossem variar com os jogos devido aos seus estilos, porém no final da pesquisa, notou-se que as respostas de todos os jogos seguiam o mesmo padrão, com leves variações. A Figura 8 mostra a porcentagem de usuários em relação ao *ranking* dos filtros.

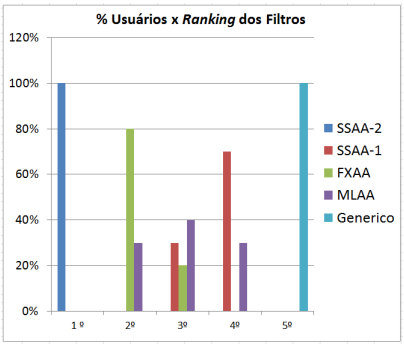


Fig. 8: Preferências dos Usuários

VI. Conclusão

Com a finalização do trabalho, conclui-se que uma grande quantidade de pessoas desconhece os significados e as diferenças dos filtros, indicando que as desenvolvedoras, em geral, não divulgam estas informações o suficiente. A ordenação dos filtros não indica necessáriamente qual é o melhor filtro pois, como pode ser visualizado na Figura 8, vários usuários possuem opiniões diferentes quando se trata de fidelidade visual. O SSAA utilizando o algoritmo de OGSS mostrou-se o melhor resultado, reafirmando que, aplicando-se uma boa técnica de amostragem, garante bons resultados. Com a aprovação de 80% dos usuários, o segundo lugar ficou para o FXAA, provando que um pouco de *blur* é aceitável no resultado final quando o algoritmo remove os serrilhados de forma competente. O terceiro filtro, o MLAA, dividiu a opinião de todos os usuários, pois não soluciona o problema tão bem quanto os dois anteriores. Por último, tanto o SSAA com HRAA e o filtro genérico apontam que quando o *blur* aumenta demais, a qualidade da imagem cai drasticamente.

Os resultados apontam claramente que existem muitas divergências entre as opiniões, mostrando que, se tivesse uma forma de comparar a qualidade da imagem diretamente das opções do jogo, sem a necessidade da realização de testes manuais ou buscas na internet, seria um grande facilitador para os usuários na hora da escolha dos filtros.

[4] http://www.fraps.com/

Acknowledgment

Inicialmente, gostaríamos de agradecer ao professor Dr. Denis Fernandes, por possibilitar o desenvolvimento desta proposta como Projeto Final da disciplina de Aplicações de Processamento Digital de Sinais, sempre disponibilizando seu tempo para aconselhar e ajudar na pesquisa e desenvolvimento do trabalho. Em segundo, gostaríamos de agradecer aos amigos e colegas, que disponibilizaram seu tempo no final do semestre, durante o período de provas, para responder o questionário e realizar os testes finais, sendo assim responsáveis por grande parte dos resultados apresentados.

References

[1] B. P. Lathi, *Sinais e Sistemas Lineares*, 2nd ed. Bookman, 2007.

[2] B. M. P. Hearn, Donald, *Computer Graphics: C Version*, 2nd ed. Prentice Hall, Inc., 1997.

[3] F. C. Crow, "A comparison of antialiasing techniques," *IEEE Computer Graphics and Applications*, 1981.

[4] P. S. Heckbert, "Survey of texture mapping," *IEEE Computer Graphics and Applications*, 1986.

[5] R. Glidden, *Graphics Programming with Direct3d*. Addison Wesley Longman, Inc, 1996.

[6] W. Y. L. W. L. L. Z. M. Xu Dong Jiang, Bin Sheng, "Image anti-aliasing techniques for internet visual media processing: a review," *Journal of Zhejiang University SCIENCE C*, 2014.

[7] K. Beets and D. Barron, "Super-sampling anti-aliasing analyzed."

[8] T. Akemine-Moller, "An extremelt inexpensive multisampling scheme," Chalmers University of Techonolgy, Tech. Rep., 2003.

[9] J. Jimenez, J. I. Echevarria, T. Sousa, and D. Gutierrez, "Smaa: Enhanced morphological antialiasing," *Computer Graphics Forum (Proc. EUROGRAPHICS 2012)*, vol. 31, no. 2, 2012.

[10] C. R. O. K. Eric Enderton, Eric Lum, "Accumulative anti-aliasing," NVIDIA, Tech. Rep., 2015.

[11] A. Developers, "Eqaa modes for amd 6900 series graphics cards," Advanced Micro Devices, AMD, Tech. Rep., 2011.

[12] P. Young, "Csaa (coverage sampling antialiasing)," nVidia, Tech. Rep., 2008.

[13] nVidia. (2012) nvidia technologies txaa. [Online]. Available: http://www.geforce.co.uk/hardware/technology/txaa/technology

[14] T. Lottes, "Fxaa," NVIDIA, Tech. Rep., 2009, white Paper.

[15] J. I. E. F. N. D. G. Jorge Jimenez, Balen Masia. (2011) Practical morphological anti-aliasing. [Online]. Available: http://www.iryoku.com/mlaa/

[16] I. H. M. Marcelo Cohen, *OpenGL Uma Abordagem Prática e Objetiva*. Novatec, 2006.

[17] S. R. Buss, *3D Computer Graphics: A Mathematical Introduction with OpenGL*. Cambridge University Press, 2003.

[18] nvidia guides anisotropic filtering. [Online]. Available: http://www.geforce.com/whats-new/guides/aa-af-guide1