UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO - UFERSA
PRÓ-REITORIA DE GRADUAÇÃO
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Rodolfo Felipe Medeiros Alves

Aplicação de *Antialiasing* em Cenas Gráficas Utilizando Filtragem Espacial de Imagens

MOSSORÓ
2018

Rodolfo Felipe Medeiros Alves

Aplicação de *Antialiasing* em Cenas Gráficas Utilizando Filtragem Espacial de Imagens

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal Rural do Semi-Árido como requisito para a obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Leandro Carlos de Souza
Co-orientador: Prof. Dr. Leiva Casemiro Oliveira

MOSSORÓ
2018

A474a Alves, Rodolfo Felipe Medeiros.
Aplicação de Antialiasing em Cenas Gráficas Utilizando Filtragem Espacial de Imagens / Rodolfo Felipe Medeiros Alves. - 2018.
49 f. : il.

Orientador: Leandro Carlos de Souza.
Coorientador: Leiva Casemiro Oliveira.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Ciência da Computação, 2018.

1. Aliasing. 2. Antialiasing. 3. Filtragem Espacial. 4. Computação Gráfica. 5. Processamento Digital de Imagens. I. Souza, Leandro Carlos de, orient. II. Oliveira, Leiva Casemiro, co-orient. III. Título.

Rodolfo Felipe Medeiros Alves

Aplicação de *Antialiasing* em Cenas Gráficas Utilizando Filtragem Espacial de Imagens

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal Rural do Semi-Árido como requisito para a obtenção do grau de bacharel em Ciência da Computação.

Defendida em: 04/09/2018.

BANCA EXAMINADORA

Prof. Dr. Leandro Carlos de Souza (UFERSA)
Orientador e Presidente

Prof. Dr. Leiva Casemiro Oliveira (UFERSA)
Coorientador e Primeiro Membro

Prof. Dr. Angélica Félix de Castro (UFERSA)
Segundo Membro

MOSSORÓ
2018

Agradecimentos

Agradeço primeiramente a Deus, nosso senhor!

Agradeço a minha mãe que sempre me apoiou em todos os momentos e em todas as minhas escolhas.

Agradeço aos meus orientadores que me auxiliaram e me ajudaram em todos os momentos.

Agradeço ao quadro de professores do curso de ciência da computação da instituição, por todo conhecimento repassado e pela vontade do crescimento dos alunos.

Agradeço a todos os meus amigos.

Uma cena vale mais do que mil palavras.

(Adaptado de Confúcio)

Resumo

O efeito do *aliasing* em cenas gráficas decorre da discretização que deve ser aplicada na imagem da cena. Esta discretização decorre em virtude da limitação física que os *pixels* impõem na representação das cores comprometendo o aspecto visual da cena criada. Como resultado, obtêm-se uma redução na qualidade e na atratividade visual. Neste contexto, métodos de *antialiasing* foram propostos com o intuito de minimizar os efeitos causados pelo *aliasing*. Eles atuam na suavização das cores diminuindo a variação brusca em certas regiões da cena. A maioria destes métodos de *antialiasing* atuam na divisão dos *pixels* gerando novos elementos, denominados subpixels, que são utilizados para o cálculo mais preciso da cor de um *pixel*. Apesar da melhora visual nas cenas, eles se tornam bastantes custosos em termos de consumo de memória e de processamento, o que inviabiliza a sua aplicação efetiva em aplicações de tempo real como simuladores ou jogos, por exemplo. Uma limitação inerente a estes métodos é que são projetados para atuarem na rasterização de figuras específicas como retas ou curvas especiais. O presente trabalho tem como proposta utilizar os métodos de filtragem espacial de imagens para criar um novo método genérico que utilize os *pixels* gerados com o objetivo de corrigir o efeito de *aliasing*. Esta abordagem baseia-se em filtragem espacial, no qual é mais eficiente na utilização da memória e em processamento. Além disso, esta estratégia permite a combinação de diferentes transformações geométricas para detecção de diversas formas de objetos. O método agrupa os filtros de aguçamento e suavização aliados com transformação geométrica de rotação para realizar as operações no domínio espacial da imagem.

Palavras-chave: *Aliasing*. *Antialiasing*. Filtragem Espacial. Computação Gráfica. Processamento Digital de Imagens.

Abstract

The effect of aliasing on graphic scenes stems from the discretization that must be applied to the scene image. This discretization takes place due to the physical limitation that the pixels impose in the representation of the colors compromising the visual aspect of the created scene. As a result, a reduction in quality and visual attractiveness is achieved. Antialiasing methods were proposed in order to minimize the effects caused by aliasing. They work on color smoothing reducing abrupt variation in certain regions of the scene. Most of these antialiasing methods work in the division of pixels generating new elements, called subpixels that are used for the more precise calculation of the color of a pixel. Despite the visual improvement in the scenes, they become quite costly in terms of memory consumption and processing, which makes it impossible to apply effectively in real-time applications such as simulators or games. An inherent limitation to these methods is that they are designed to act on rasterization of specific figures such as straight lines or special curves. The purpose of this work is to use spatial image filtering methods to create a new generic method that uses the generated pixels to correct the aliasing effect. This approach is based on spatial filtering, which is more efficient in memory utilization and processing. This strategy allows the combination of different geometric transformations to detect different forms of objects. The method groups the enhanced and smoothing filters allied with geometric transformation of rotation to perform operations on the spatial domain of the image.

Keywords: Aliasing. Antialiasing. Spatial Filtering. Computer Graphics. Digital Image Processing.

Lista de Ilustrações

Lista de Abreviações e Siglas

OpenGL	Biblioteca livre para geração de cenas gráficas
API	Interface de programação de aplicativos
PDI	Processamento digital de imagens
OpenCV	Biblioteca livre para manipular imagens
CMYK	Modelo de cor formado por: ciano, magenta, amarelo e preto
HSI	Modelo de cor formado por: matiz, saturação e valor
YIQ	Modelo de cor formado por: luminância, interpolação e quadratura.
RGB	Modelo de cor formado por: vermelho, verde e azul.

Sumário

1 Introdução

A computação gráfica se tornou uma ferramenta essencial em diversas áreas de conhecimento que necessitam criar e visualizar objetos digitais, tais como: nos Jogos, na Simulação, na Publicidade, no Cinema e em diversas áreas da Multimídia. Não obstante, o realismo se tornou um objetivo fundamental na computação gráfica, seja no entretenimento, na educação e em muitas outras aplicações (AZEVEDO; CONCI, 2003).

A representação de cenas gráficas cada vez mais realistas tornou-se uma tarefa árdua, pois em alguns dispositivos encontram-se limitações físicas quanto à quantidade de pontos para representar a informação. Estas limitações existentes no *hardware* de exibição provocam o efeito *aliasing*, ocorrido devido a uma subamostragem das cores projetadas e que é caracterizado como uma descontinuidade nas cores da cena. Um resultado comum decorrente do *aliasing* é a exibição de serrilhamento nas bordas dos objetos da cena.

Métodos de *antialiasing* foram propostos com o objetivo de melhorar a qualidade visual das imagens resultantes das cenas gráficas. Uma das principais estratégias utilizadas por estes métodos é a divisão da cena gráfica em sub-regiões que não existem fisicamente e que são logicamente representadas em memória. Estas estruturas lógicas são utilizadas na suavização das transições de intensidade. Desta forma, a cena final gerada apresenta uma redução na descontinuidade das cores (HEARN; BAKER; CARITHERS, 2014).

Em geral, os métodos de *antialiasing* necessitam de muita memória e processamento, e em muitos casos, o usuário deve dispor de um *hardware* específico (como placas gráficas) para conseguir incluir esta funcionalidade. Mesmo assim, dependendo dos gráficos e informações presentes na cena exigida pela aplicação, o usuário deve escolher entre um efeito visual de baixa qualidade para que seja possível obter uma resposta visual em tempo aceitável.

A utilização de técnicas de Processamento Digital de Imagens (PDI) pode ser combinada a métodos de computação gráfica para criação de cenas com maior realismo. Em especial, as técnicas de PDI no domínio espacial são fáceis de implementar e não exigem computação de alto desempenho, na sua maioria, uma vez que lidam diretamente com os *pixels* formadores da matriz da imagem (GONZALEZ; WOODS, 2006). Assim, o uso de técnicas para ampliação, deformação, rotação, suavização e aguçamento de imagens digitais combinadas conseguiriam reduzir o consumo da memória para realizar as operações necessárias no processo de amenização do efeito *aliasing*. Neste contexto, o trabalho apresenta um novo algoritmo de *antialiasing* construído a partir de métodos de filtragem espacial utilizados em PDI.

1.1. Problema

Ao realizar a amostragem da função contínua que representa a imagem original, em um processo de amostragem com baixa frequência, nota-se que essa nova imagem amostrada ao ser visualizada ou reestruturada possui redução de qualidade quando comparada a original, justamente pelo fato de que a frequência de amostragem é inferior em relação à frequência contínua da imagem original (CONCI; AZEVEDO, 2008).

Um dos efeitos percebidos com um processo de amostragem deficiente são a presença de *aliasing* na imagem/cena. Desde os primórdios da computação gráfica tem-se estudado diversos métodos para reduzir ou amenizar tal efeito. No entanto, os métodos existentes são bastante custosos tanto em processamento quanto em consumo de memória, além de possuírem uma restrição de atuação em determinadas formas geométricas (HEARN; BAKER; CARITHERS, 2014). Desse modo, um dos problemas comuns do processamento de imagens é encontrar/desenvolver algoritmos mais eficientes em consumo de memória, processamento e na generalização do tratamento do efeito *aliasing*.

1.2. Objetivos

Este trabalho tem como objetivo principal desenvolver um algoritmo genérico de antialiasing para ajudar na redução da percepção do efeito aliasing em cenas gráficas. Serão utilizados filtros de aguçamento e suavização de imagens juntamente com transformações geométricas, para realizar operações de rotação na detecção dinâmica dos melhores filtros que devem ser utilizados em cada região da cena. Para consecução do objetivo principal, os seguintes objetivos específicos foram definidos:

- Elaboração de uma forma de detecção da existência de variação de cores.
- Elaboração de um mecanismo para suavizar a variação de cores encontradas.
- Combinação dos pontos anteriores de forma genérica em diferentes graus de projeção.
- Construção de cenários de testes para validação do método.

1.3. Justificativa

Vários métodos de *antialiasing* foram propostos para solucionar o problema do efeito *aliasing*. No entanto, eles não são eficientes na utilização da memória, possuem deficiência na detecção de diferentes objetos, e em muitos casos, são difíceis de implementar. Desta forma, os métodos precisam ser mais efetivos na utilização da memória e possuírem generalização no seu funcionamento. Foi pensando nisso que o algoritmo proposto foi desenvolvido. Procura-se um melhor custo-benefício na utilização da memória e uma única forma de detecção dos objetos das cenas gráficas.

1.4. Organização

Além deste capítulo de Introdução, este trabalho possui mais 3 Capítulos, são eles:

Capítulo 2 – Referencial Teórico
Este capítulo apresenta as informações mais relevantes sobre a teoria contida na formulação do problema e solução propostos para reduzir o efeito. É descrito o problema gerador do efeito. Além disso, ele apresenta a teoria utilizada como forma de solução na criação do novo método genérico de *antialiasing* baseado em filtragem espacial de imagens.

Capítulo 3 – *Antialiasing* Baseado em Filtragem Espacial de Imagens
Este capítulo apresenta o algoritmo proposto e resultados obtidos na execução de cada parte do código.

Capítulo 4 – Conclusão e Trabalhos Futuros
O presente capítulo apresenta as informações conclusivas para o método de *antialiasing* desenvolvido, além de direcionamentos e possíveis evoluções em trabalhos futuros.

2 Referencial Teórico

Neste capítulo serão descritos os conceitos mais importantes sobre o trabalho proposto. Inicialmente apresenta uma introdução sobre processamento digital de imagens, sobre a representação do *pixel*, o sistema de cores, filtragem espacial, filtros de suavização (passa-baixa) e filtros de aguçamento (passa-alta) e transformação geométrica. Em seguida, descreve sobre a computação gráfica, com destaque para: o efeito *aliasing* e sobre os métodos de *antialiasi*ng descritos na literatura. Por fim, são descritas as API's (Interface de Programação de Aplicativos) OpenGL e OpenCV utilizadas para incluir o algoritmo desenvolvido no processamento da cena e na geração das imagens para visualização dos resultados obtidos, respectivamente.

2.1. Processamento Digital de Imagens

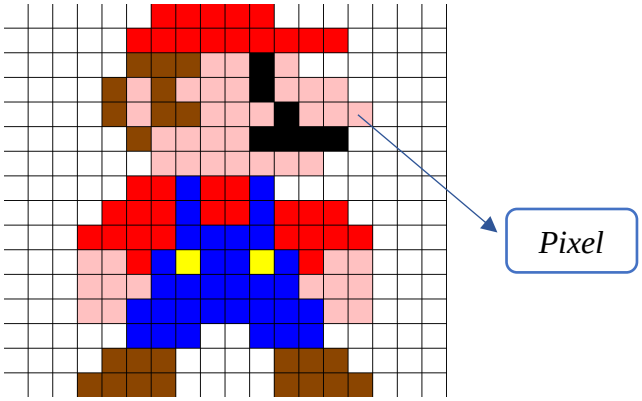
No processamento de imagem digital (PDI), geralmente trabalha-se com matrizes de números obtidos por pontos de amostragem espacial de uma imagem física. Após o processamento, outra matriz de números é produzida e esses números são usados para reconstruir uma imagem contínua para visualização (GONZALEZ; WOODS, 2006).

Ao se referir ao processamento digital de imagens estamos identificando que um computador digital está processando imagens digitais. Os métodos utilizados recebem imagens como entrada e na saída geram imagens ou atributos que são extraídos do corpo da imagem. Portanto, o processamento de imagens pode ser utilizado com intuito de obter diferentes resultados ou objetivos (MARQUES FILHO; VIEIRA NETO, 1999).

2.1.1. Pixel e Vizinhança

O *pixel* também chamado de elementos pictóricos, elementos de imagem ou *pels* é o elemento de menor unidade sobre a qual se pode realizar operações e como termo mais utilizado para representar os elementos de uma imagem digital. O *pixel* mensura um conjunto de pontos que não possuem área, comprimento, volume ou qualquer dimensão semelhante (PRATT, 2007). Uma representação visual do pixel pode ser observada na Figura 1.

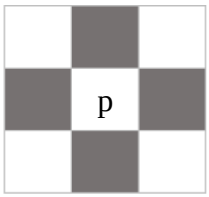
Figura 1 - Representatividade do pixel.



Fonte: Autoria própria.

O termo vizinhança é utilizado para descrever os *pixels* que estão vizinhos diretamente com o *pixel* em ênfase. Por definição o *pixel p* com coordenadas (x, y) possuem quatro vizinhos horizontais e verticais, com coordenadas descritas na Figura 2.

Figura 2 - 4-vizinhança. Em (a) os *pixels* horizontais e verticais. Em (b) as coordenadas dos *pixels*.



(a)

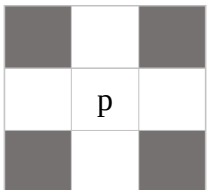
	(x, y-1)	
(x-1, y)	(x, y)	(x+1, y)
	(x, y+1)	

(b)

Fonte: Autoria própria.

Continuando com o *pixel p* com coordenadas (x, y) o mesmo possui quatro vizinhos diagonais, com coordenadas descritas na Figura 3.

Figura 3 - Vizinhança diagonal.



(x-1, y-1)		(x+1, y-1)
	(x, y)	
(x-1, y+1)		(x+1, y+1)

Fonte: Autoria própria.

Com a junção dos dois tipos de vizinhança apresentados, temos a matriz de 8-vizinhança descrita na Figura 4.

Figura 4 - 8-vizinhança.

(x-1, y-1)	(x, y-1)	(x+1, y-1)
(x-1, y)	(x, y)	(x+1, y)
(x-1, y+1)	(x, y+1)	(x+1, y+1)

Fonte: Autoria própria.

2.1.2. Representação da Intensidade

Em PDI, a imagem pode ser definida como uma função bidimensional f(x, y), em que x e y são as coordenadas espaciais, e a amplitude de f em qualquer par de coordenadas (x, y) é chamada de intensidade ou nível de cinza da imagem nesse ponto, como ilustra a Figura 5.

Figura 5 - Imagem em escala de cinza. Em (a) a imagem original. Em (b) uma porção da imagem indicando a intensidade de alguns *pixels*.

(a) (b)

Fonte: Adaptado de Gonzalez e Woods (2006).

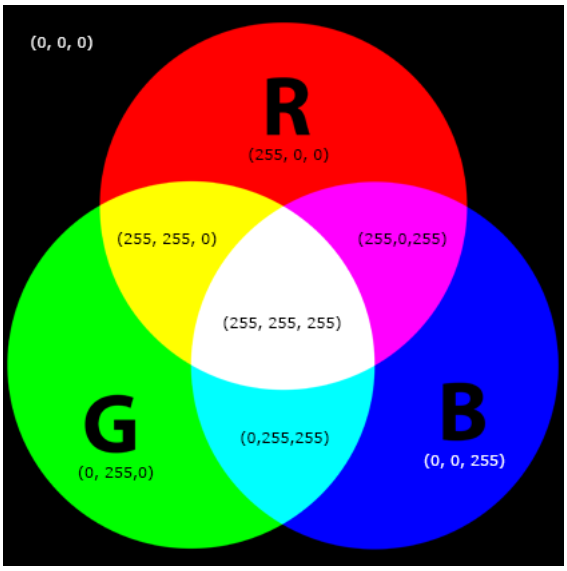
Diz-se que uma imagem é digital quando os valores da intensidade formada por x e y são quantidades finitas e discretas. Uma imagem digital é composta de um número finito de elementos, cada um com localização e valores específicos (Figura 5(b)) (PRATT, 2007).

2.1.3. Sistema de Cores

O modelo de cores ou sistema de cores foi criado com o intuito de padronizar e facilitar a especificação das cores na busca de ser um modelo amplamente aceito e usado. Existem inúmeros sistemas de cores em uso atualmente, cada um com sua usabilidade para determinado dispositivo ou viabilidade de cores, destacando-se os sistemas CMYK, HSI e especialmente o RGB (HUNT, 2004).

Nos sistemas de cores aditivos utilizados por monitores de vídeo e televisão ocorre uma mistura de vários comprimentos de onda luminosa provocando uma sensação de cor ao atingir e sensibilizar o olho. O sistema de cor RGB forma a cor com adição de três cores primárias: vermelho, verde e azul. Durante a adição das cores, o preto é formado pela ausência delas, e na geração do branco, todas as cores são utilizadas na sua totalidade, como pode ser visto na Figura 6 (AZEVEDO; CONCI, 2003).

Figura 6 - Os três canais de cores do modelo RGB.

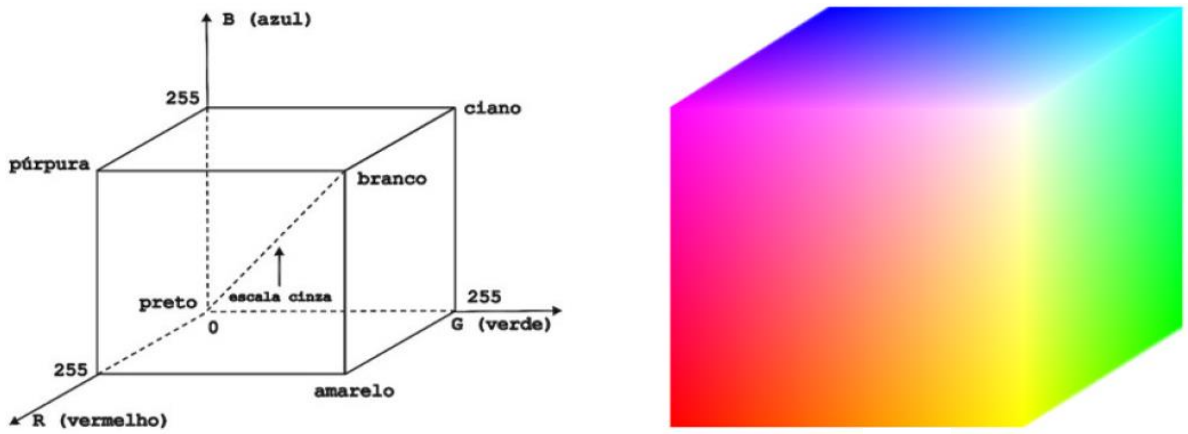


Fonte: Autoria própria.

Uma imagem que utilizada o sistema RGB é constituída pela junção de três canais de cor que representam as matrizes independentes. O modelo RGB é um dos mais amplamente utilizados, a variação de cores se dar entre 0 a 255 (8 bits), sendo possível representar mais de 16 milhões de cores (HUNT, 2004).

Na Figura 7(b) pode ser visualizado um cubo degradê de cores formadas pelo modelo de cor RGB. A Figura 7(a) representa as direções de cada cor e sua variação ao longo do espaço tridimensional para formar as cores do modelo RGB no cubo.

Figura 7 - Cubo de cores RGB. Em (a) uma representação da variação de cor ao longo do espaço. Em (b) o cubo degradê formado pelo modelo de cor RGB.



(a) (b)

Fonte: Adaptado de Gonzalez e Woods (2006).

2.1.4. Filtragem Espacial

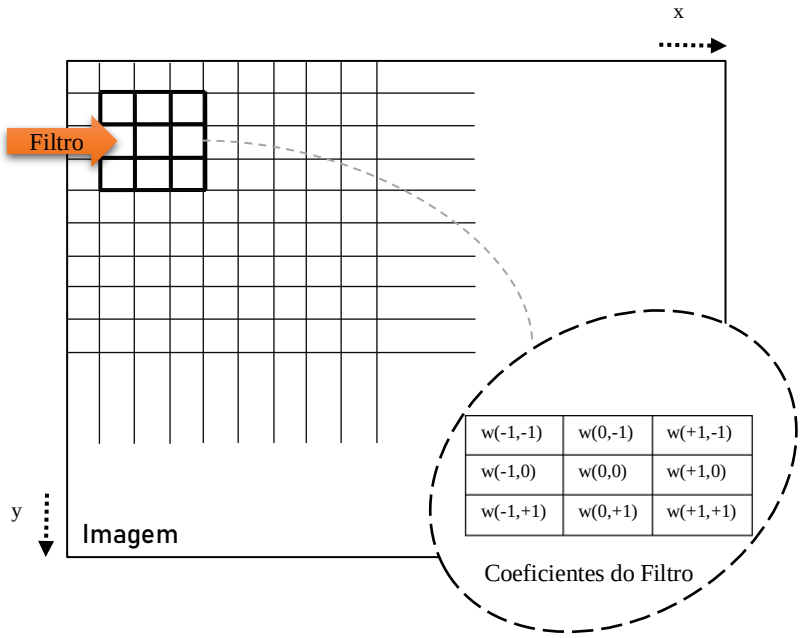
O domínio espacial corresponde a matriz de *pixels* de uma imagem, no qual ocorre o manuseio direto de seus *pixels* através das técnicas de processamento de imagens. Assim, quando se trabalha com imagens no domínio espacial todas as operações são realizadas diretamente nos *pixels* da imagem original (FARIA, 2010).

De acordo com Gonzalez e Woods (2006) o termo filtro representa a vizinhança de *pixels* entorno de um *pixel* central. Essa amostra de *pixels* é utilizada para realizar operações de filtragem que irão produzir um *pixel* final como produto. A operação de filtragem cria um novo *pixel* na mesma coordenada central do filtro selecionado. Ao percorrer todo o espaço da matriz é gerado uma nova matriz com os resultados obtidos das operações de filtragem.

Na Figura 8 pode-se visualizar o funcionamento da filtragem espacial com uma vizinhança 3 x 3. A filtragem espacial de modo geral em uma imagem de dimensões W x H

com um filtro de dimensões w x n é dada pela equação (2.1), no qual x e y variam de forma que cada *pixel* em w percorre todos os *pixels* em f (GONZALEZ, WOODS, 2006).

Figura 8 - Operação de filtragem espacial com vizinhança 3x3.



Fonte: Autoria própria.

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \qquad (2.1)$$

O w corresponde aos *pixels* da máscara utilizada na operação com o filtro selecionado. O f corresponde aos *pixels* da imagem a ser processada. Os limites de a e b em um filtro 3 x 3 vão variar de -1 a 1 para percorrer os coeficientes do filtro selecionado.

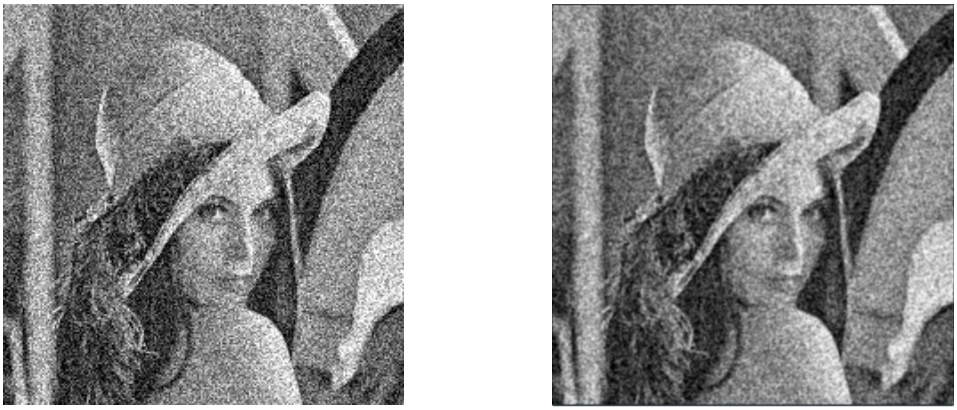
2.1.4.1. Filtros de Suavização

Filtros espaciais de suavização são utilizados de forma a reduzir ruídos que estão presentes nas imagens. Os filtros de suavização tendem a borrar a imagem original com o intuito de minimizar a variação de intensidade da vizinhança filtrada. A realização dessa operação utiliza os chamados filtros passa-baixa. Os filtros espaciais passa-baixa lineares de suavização condizem simplesmente com a média dos *pixels* da vizinhança da máscara de filtragem (PRATT, 2007).

Na prática, em uma situação mais simples se tem o filtro de média como descrito na equação (2.2). Ao somar os nove valores de intensidade (zi) de vizinhança 3 x 3 com centro (x, y) se obtêm o valor médio (R) para substituir o valor inicial utilizado (GONZALEZ, WOODS, 2006). Como pode ser visualizado na Figura 9.

$$R = \frac{1}{9}\sum_{i=1}^{9} z_i \qquad (2.2)$$

Figura 9 - Uma aplicação de passa-baixa de média. Em (a) uma imagem de lena com efeito sal e pimenta. Em (b) uma imagem de lena suavizada com máscara da media 3x3.



(a) (b)

Fonte: Autoria própria.

Na Figura 9(a) foi aplicado um ruído para variar as intensidades da vizinhança dos pixels para que a máscara de passa-baixa pudesse atuar. Essa forma possibilita uma melhor visualização do processo de borramento proposto pela operação de passa-baixa. A Figura 9(b) ilustra o resultado obtido com o processamento da passa-baixa descrita na equação (2.2). É possível perceber uma redução dos ruídos que estão presentes na imagem original.

2.1.4.2. Filtros de Aguçamento

Diferentemente dos filtros de suavização, temos os filtros de aguçamento ou também chamados de passa-alta, que operam pela diferenciação das intensidades encontradas no domínio espacial (FILHO E NETO, 1999). Dessa forma, os filtros passa-alta realçam bordas, outras descontinuidades como o ruído ao mesmo tempo que atenuam as áreas de intensidades com pouca variação e áreas homogêneas da imagem. Os *pixels* da borda são os *pixels* que possuem as intensidades da função imagem que mudam abruptamente. Diferentes filtros

(Figura 10) foram propostos para realizar a detecção dessas intensidades abruptas (GONZALEZ; WOODS, 2006).

Figura 10 - Filtros para detecção de intensidades abruptas com graus variados.

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal

2	-1	-1
-1	2	-1
-1	-1	2

+45º

-1	2	-1
-1	2	-1
-1	2	-1

Vertical

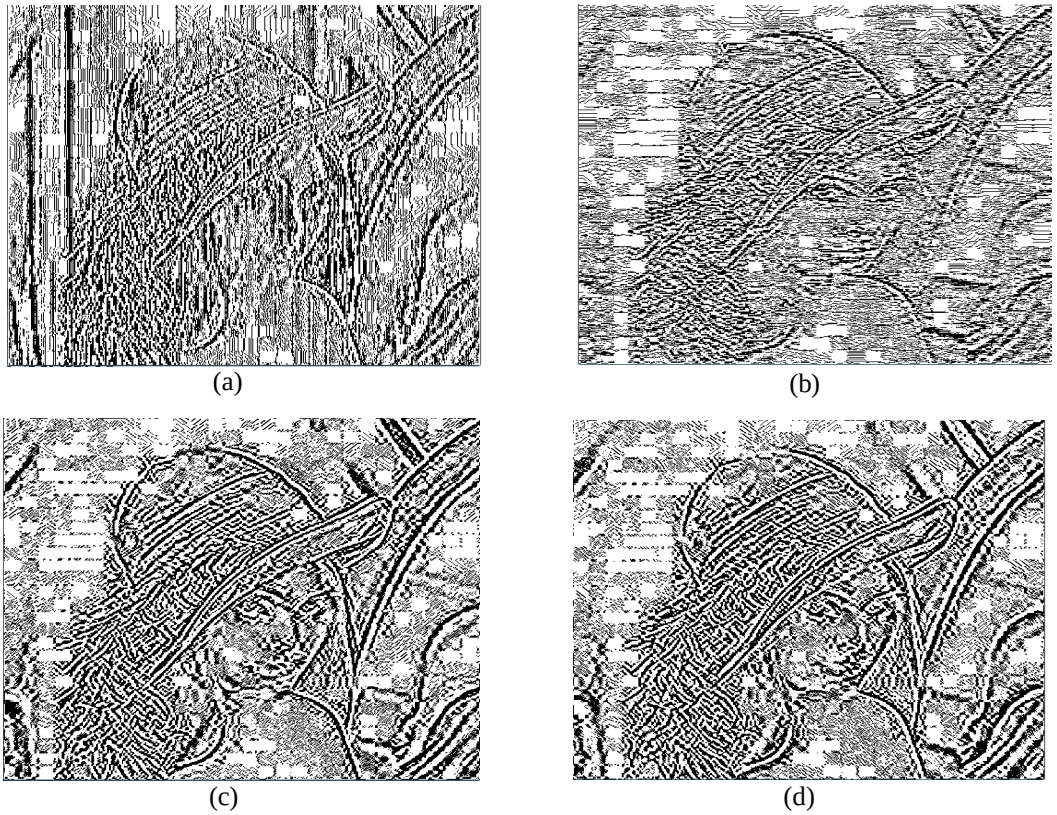
-1	-1	2
-1	2	-1
2	-1	-1

-45º

Fonte: Adaptado de Gonzalez e Woods (2006).

O filtro de detecção horizontal irá detectar as variações abruptas que estão com orientação em 0º. Diferentemente, o filtro vertical irá realçar as bordas que estão com angulação de rotação em 90º. Os filtros +45 e -45 graus irão detectar as bordas que possuam ângulo de projeção correspondente. Na Figura 11 pode ser visualizado alguns resultados obtidos da aplicação desses filtros.

Figura 11 - Aplicação de filtros para detecção de intensidades abruptas com graus. Em (a) 90º. Em (b) 0º. Em (c) +45º. Em (d) -45º.



(a) (b)

(c) (d)

Fonte: Autoria própria.

No entanto, é perceptível que os filtros de detecção de intensidades abruptas detectam as bordas e também os ruídos com a determinada angulação designada. Entretanto, não é possível identificar intensidades abruptas com variações de graus que não possuem máscaras para reconhecer tal angulação, como: 15º, 30º, 70º, entre outras. Além de ocorrer a detecção de ruídos que não são interessantes para a análise das bordas.

2.1.5. Transformação Geométrica

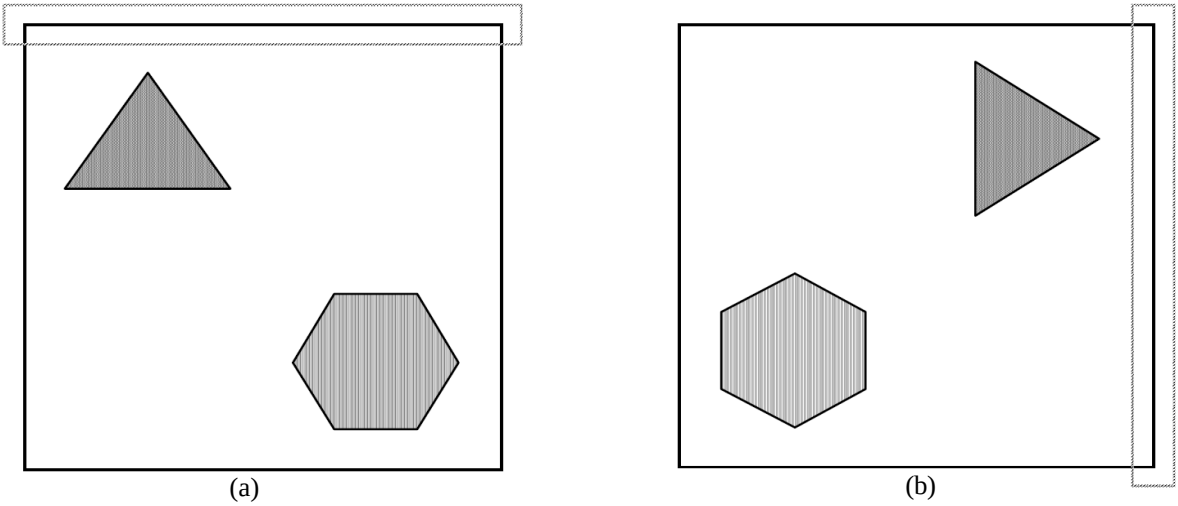
Uma vez que os dados da imagem foram amostrados e quantificados, alterações na imagem podem ser realizadas com o auxílio das transformações geométricas para diferentes finalidades, sejam elas: no aumento e diminuição de escala, na reflexão, na translação, no cisalhamento e na rotação da imagem. As operações geométricas alteram as relações espaciais entre os *pixels* da imagem e podem ser usados para corrigir distorções, mudanças de escala e rotações (EASTON, 2010).

2.1.5.1. Rotação

Uma imagem pode ser rotacionada de um ângulo qualquer, em sentido horário ou sentido anti-horário. Matematicamente, a rotação de cada ponto (x, y) de uma imagem por um ângulo variado Θ mapeará o resultado desta operação na localidade de coordenadas (x', y'), sendo calculado pela equação (2.3) (JANKE, 2015).

$$\begin{aligned} x' &= x.\cos(\theta) - y.\mathrm{sen}(\theta) \\ y' &= y.\cos(\theta) + x.\mathrm{sen}(\theta) \end{aligned} \tag{2.3}$$

Figura 12 - Rotação de objetos com ângulo de 90º. Em (a) objetos orientados em 0º. Em (b) objetos rotacionados em 90º.



Fonte: Adaptado de Marques Filho e Vieira Neto (1999).

A Figura 12(b) representa o resultado da operação de rotação aplicada nos objetos da imagem Figura 12(a). Esta rotação foi realizada com o ângulo de 90º.

2.2. Computação Gráfica

Na computação gráfica, a ciência e a arte se comunicam visualmente por meio de um computador e seus dispositivos gráficos. Diferentes algoritmos estão sendo utilizados para melhorar a qualidade das cenas apresentadas para o usuário de forma a utilizar o máximo da capacidade do dispositivo. A demanda do usuário por gráficos mais atraentes, aumentou a necessidade de estudos na área da computação gráfica. Novos algoritmos são propostos por diferentes autores com o intuito de melhorar as cenas gráficas em variados dispositivos, aproveitando o que cada um pode oferecer de capacidade visual ou processamento (CONCI; AZEVEDO, 2008).

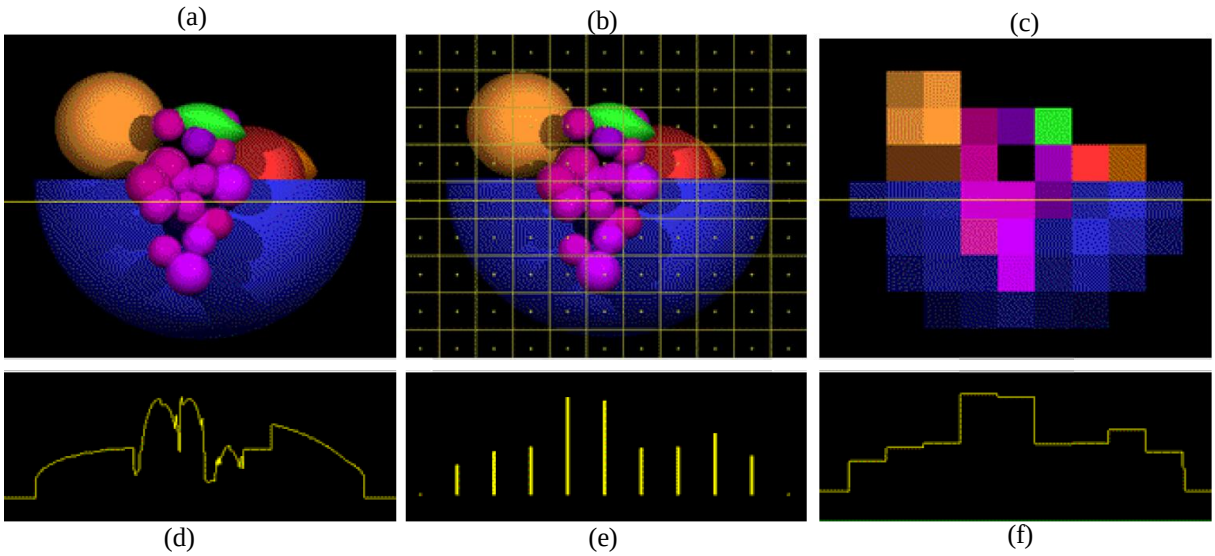
2.2.1. *Aliasing*

Durante o processo de amostragem da cena original (Figura 13) em alguns casos, se obtêm uma amostragem inferior a desejada, e ao ser visualizada ou reestruturada é perceptível a redução de qualidade gerada pela frequência inferior em relação a frequência contínua

original. Correspondente a uma subamostragem das cores projetadas, caracterizada pela perda de detalhes da amostragem da projeção original (FOLEY *et al*. 1996; CONCI; AZEVEDO, 2008).

Ao observar as cenas projetadas em dispositivos gráficos, sejam linhas, pontos ou imagens complexas, estas, na verdade, são um conjunto de *pixels* – unidades quadradas, indivisíveis, organizadas de forma matricial (grade).

Figura 13 - Processo de amostragem da imagem original na imagem final visualizada. Em (a) imagem original. Em (b) a grade de *pixels.* Em (c) imagem resultante. Em (d) amostragem original. Em (e) o sinal amostrado. Em (f) amostragem inferior.



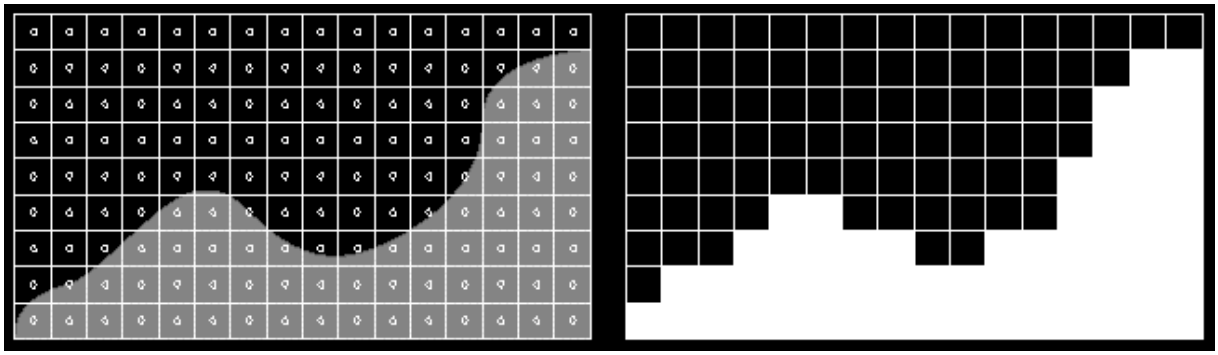
Fonte: Adaptado de Raposo (2007).

A Figura 13 possui em sua composição três imagens em fases distintas. A Figura 13(a) representa imagem original a ser amostrada e os sinais de frequência originais. A Figura 13(b) está representando o sinal amostrado da imagem original. E por fim, a Figura 13(c) é amostragem obtida da função contínua original, neste caso, o resultado foi inferior por causa da frequência de amostragem ser menor do que o esperado para conseguir amostrar todos os pontos corretamente.

Na tentativa de inserir formas circulares, ou até mesmo retas com angulações diferentes de 0º , 90º , 180º ou 270º em relação aos eixos dessa matriz de *pixels*, devido a quantidade disponível de *pixels*, as técnicas para desenho dessas formas não são eficazes para mascarar um aspecto de escadinha (Figura 14) devido a distribuição dos itens que formam o objeto não poderem ser representadas de forma lisa. O processo de *aliasing* se caracteriza pelo serrilhamento das bordas nas projeções gráficas, ocasionando uma redução na qualidade da

projeção apresentada, em consequência, o serrilhamento reduz a qualidade final da cena produzindo uma baixa atratividade visual.

Figura 14 - Representação do efeito *aliasing* – Em (a) o objeto real a ser projetado. Em (b) o objeto projetado.



(a) (b)

Fonte: Adaptado de Lang (2016).

Na Figura 14(a) tem-se um objeto ondulado real a ser projetado na cena. Figura 14(b) o mesmo objeto projetado com o efeito *aliasing*.

De modo geral, o *aliasing* é projetado pela tentativa de armazenar muitos valores em poucos pixels. Ao tentar arredondar a cobertura real em frações de 0 ou 1 criam-se imprecisões que geralmente aparecem na forma de imagens em blocos. Os métodos *antialiasing* buscam melhorar aproximação da cobertura para reduzir o efeito (HUGHES *et al.* 2014).

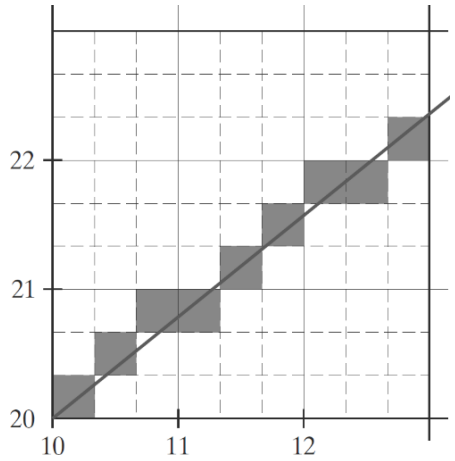
2.2.2. Métodos de *Antialiasing*

Não existe uma solução ou método que irá resolver por completo a deformidade ocasionada na representação gráfica das cenas. No entanto, a busca está destinada em reduzir o efeito de modo a retirar a percepção visual que se gera ao projetar tais gráficos. Basicamente o efeito de escadinha é tratado ao se trabalhar os *pixels* anexos à zona afetada de forma a suavizar diferentes intensidade de cores. Na literatura é possível encontrar métodos para trabalhar com essas regiões irregulares.

2.2.2.1. Superamostragem por Divisão de *Pixel*

A superamostragem por divisão de *pixel* é um método de *antialiasing* que divide cada *pixel* tracejado em vários *subpixels* e em seguida realiza o cálculo dos *pixels* que estão sobre à reta a ser projetada, conforme ilustra a Figura 15. Cada *pixel* subdividido recebe um valor proporcional de intensidade com base na quantidade de *subpixels* que estão no trajeto da reta preterida. O método pode usar diferentes quantidades de divisões para o *pixel* com o intuito de aumentar a amostragem de *pixels* subdivididos, e com isso obter níveis de intensidade de cor maiores. Subdividindo em 9 partes, se obtêm três níveis de intensidade. Dividindo em 16 partes é possível obter 4 níveis de intensidade de cor e sucessivamente (FOLEY *et* al, 1996).

Figura 15 - Funcionamento do método de *antialiasing* com divisão do *pixel* em *subpixels.*



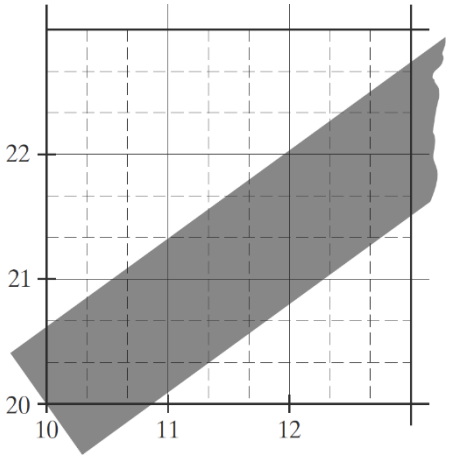
Fonte: Adaptado de Hearn, Baker e Carithers (2014).

No entanto, o aumento da necessidade de memória e processamento dos *subpixels* está diretamente proporcional à quantidade de subdivisões. Portanto, quanto mais dividir o *pixel*, uma melhor representação da intensidade do *pixel* é obtida, mas em compensação necessita de mais espaço e operações. Todavia, ao utilizar o método de divisão em *subpixels* se obtêm uma sobrecarga na memória e processamento do dispositivo, chegando em alguns casos a não execução completa do método por falta de espaço em memória.

2.2.2.2. Superamostragem em Área

O presente método de antialiasing lida com os *pixels* tracejando uma linha reta em área, com a linha de largura finita (Figura 16) atribuindo um valor em porcentagem de intensidade ao *pixel* com base na ocupação desta linha na área do *pixel (*HUGHES *et* al, 2014).

Figura 16 - Traçado da linha reta na área do *pixel* e *subpixels*.



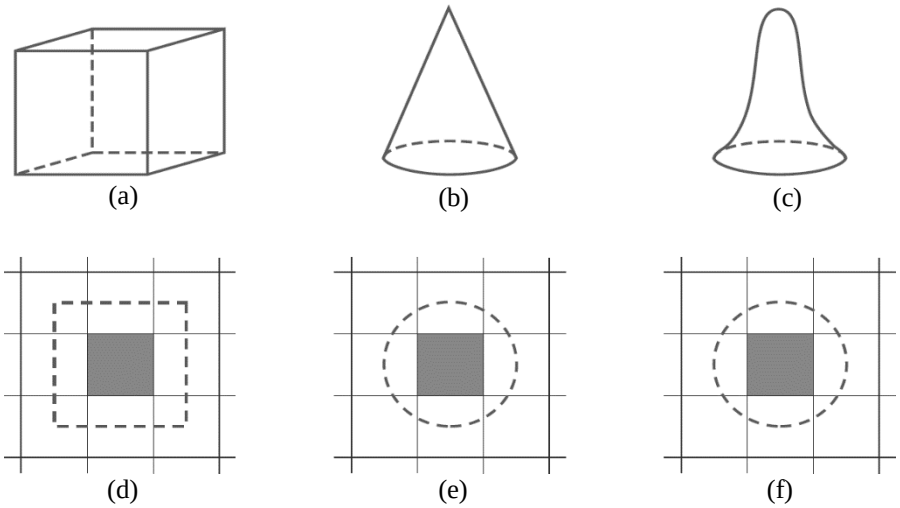
Fonte: Adaptado de Hearn, Baker e Carithers (2014).

A divisão do *pixel* continua sendo elaborada, no entanto, se faz necessário a verificação da quantidade de *subpixels* que estão sendo tracejados pela reta para determinar a porcentagem de intensidade que o *pixel* respectivo irá receber. Na Figura 16, o pixel (11, 20) está sendo coberto por praticamente 50% da reta, por isso sua intensidade máxima estaria atrelada a essa porcentagem de ocupação. A execução desse método também exige ocupação de memória adicional, consequentemente necessitando de processamento destas informações tornando-o um método custoso.

2.2.2.3. Superfícies Ponderadas

Considerada uma das técnicas mais precisas para realizar redução do *aliasing*, temos as superfícies ponderadas. O método funciona atribuindo uma superfície sobre o *pixel* para obter a média de intensidade ponderada com base na ocupação da área (HEARN; BAKER; CARITHERS, 2014), como ilustrado na Figura 17.

Figura 17 - Exemplos de superfícies ponderadas. Em (a) um cubo. Em (b) um cone. Em (c) a forma gaussiana. Em (d) função retangular. Em (e) função cônica. Em (f) função gaussiana.



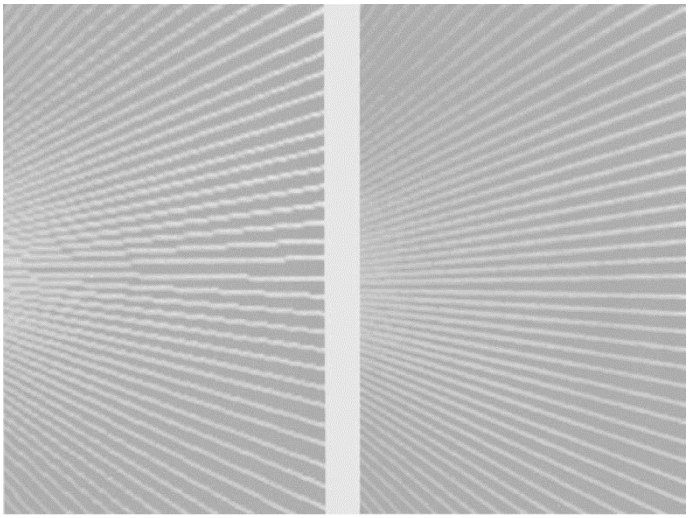
Fonte: Adaptado de Hearn, Baker e Carithers (2014).

O *pixel* ocupado pela área utilizada no cálculo terá o seu valor de intensidade final atribuído com base em porcentagem de valores que estão no centro da área e próximos a ele. No entanto, a porcentagem irá reduzir ao mesmo tempo que se distancia do centro da área.

2.2.2.4. Pixel Phasing

Este método cria um sistema de varredura para endereçar as posições de *subpixel* dentro da grade da tela. A suavização ocorre movimentando as posições do *pixel*. O sistema incorpora fases de *pixel* projetadas para que o feixe de elétrons possa ser deslocado por uma fração de diâmetro de *pixel*. O feixe elétrons é deslocado para traçar pontos mais perto do caminho verdadeiro da borda do objeto (Figura 18). Também permitem o ajuste de *pixels* individuais para um meio adicional ocorrendo uma distribuição de intensidade (HEARN; BAKER; CARITHERS, 2014).

Figura 18 – O aumento dos pontos endereçáveis pelo método *pixels phasing.* A imagem foi de 768 por 576 a 3072 por 2304. Em (a) a imagem com efeito *aliasing.* Em (b) a imagem suavizada com o método *pixels phasing.*



(a) (b)

Fonte: Adaptado de Hearn, Baker e Carithers (2014).

A Figura 18(b) apresenta o resultado da aplicação do método *pixel phasing*. O método *pixel phasing* aumenta a resolução da imagem original (Figura 18(a)) com efeito *aliasing* com objetivo de encontrar as melhores posições dos *pixels* atribuídos. Este procedimento reduz a percepção do efeito escadinha ao retornar uma imagem com melhor qualidade visual.

2.3. Ferramentas de Desenvolvimento

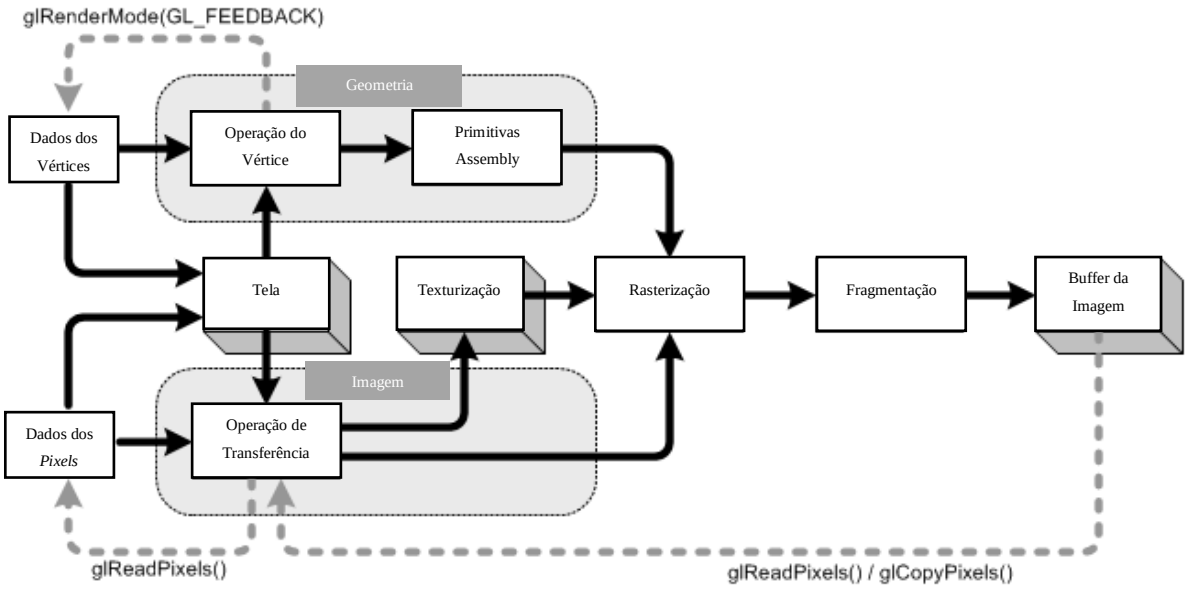
Nesta seção estão descritas as ferramentas utilizadas no auxílio da criação dos cenários projetados e na geração das imagens resultantes do processamento do algoritmo proposto. São empregadas duas API's no desenvolvimento, são elas: OpenGL e a OpenCV.

A OpenGL é uma API madura e multiplataforma que trabalha com a geração de gráficos 2D e 3D. Sua utilização se tornou completamente difundida em diferentes linguagens de programação, como: C++, Java, Python e entre outros. E também entre diferentes tipos de dispositivos, sejam: *smartphones* e grandes computadores (BORESKOV; SHIKIN, 2014).

A OpenGL simplifica o desenvolvimento de softwares gráficos, acelerando o tempo de comercialização dos produtos. As rotinas do OpenGL são responsáveis por todas as etapas necessárias para criação de cenas gráficas (Figura 19) - desde da renderização de simples

polígonos pontilhados, linear ou até a criação de superfícies curvas mais complexas iluminadas e texturizadas (KHRONOS, 1997).

Figura 19 - Representação do pipeline de execução do OpenGL.



Fonte: Adaptado de KHRONOS (1997).

Neste trabalho a OpenGL está sendo utilizada para a criação das cenas gráficas. Essas cenas geradas possuem diferentes objetos e formas. O objetivo é criar o máximo de objetos que possuam diferentes angulações de projeção. Outro papel importante da API é a obtenção dos *pixels* da cena projetada com a função *glReadPixels*. Esta função obtêm os canais RGB de todos os *pixels* que foram desenhados na tela para que assim o algoritmo proposto possa funcionar. Essa informação é essencial no tratamento do efeito *aliasing*. Sem os dados brutos dos *pixels* não é possível realizar o tratamento do efeito.

Já a OpenCV é uma biblioteca de software que possui diferentes tipos de fins, mas tem como base a manipulação de imagens para obter algum resultado. Inclui milhares de algoritmos otimizados para trabalharem com imagens, até mesmo em tempo real. Dentre as ações possíveis de realizar com a OpenCV, destaca-se: identificar objetos, classificar ações humanas em vídeo, unir imagens para produzir uma alta resolução de imagem, encontrar imagens semelhantes em um banco de dados de imagens ou também na simples definição ou criação de imagens. A biblioteca está desenvolvida em diferentes linguagens de programação, como: C++, Python, Java e MATLAB com suporte para Linux, Windows, Android e Mac OS (OPENCV, 2018).

Neste trabalho, a OpenCV cria as imagens resultantes do processamento do algoritmo de antialiasing proposto. Ela recebe os dados dos *pixels* e salva em arquivos com extensão *png*.

3 *Antialiasing* Baseado em Filtragem Espacial de Imagens

O presente capítulo descreve o algoritmo de *antialiasing* baseado em filtragem espacial de imagens. Nos próximos parágrafos pode ser observado uma descrição geral do funcionamento do algoritmo e, logo em seguida, será detalhada cada parte nas seções subsequentes do capítulo.

Foi demonstrado que os algoritmos propostos para reduzir o efeito *aliasing* necessitam bastante de memória, e consequentemente, um alto custo de processamento. Outro fator complicador no uso de tais algoritmos é que geralmente não conseguem ou não foram desenvolvidos para atuarem com alguns tipos de formas geométricas de projeção.

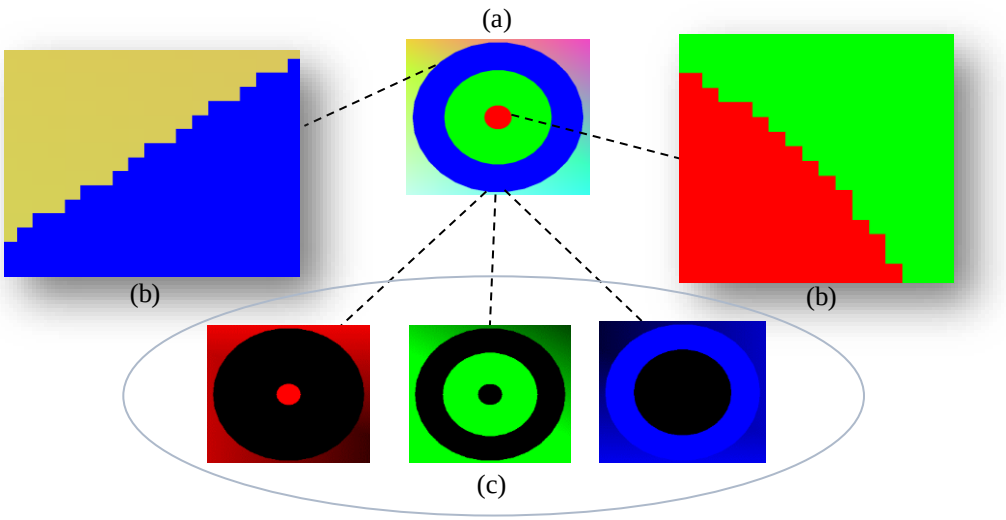
O primeiro problema é divisão dos *pixels* da projeção original em *subpixels* para ser realizado os cálculos de *antialiasing*. Este procedimento custa bastante memória e processamento. Neste caso, o algoritmo proposto apenas trata os *pixels* na sua forma original sem nenhuma divisão, conseguindo uma eficiência em memória e processamento.

O segundo problema está ligado a suavização de diferentes formas geométricas. Os métodos de *antialiasing* geralmente não conseguem detectar diferentes tipos de objetos. Neste sentido, o algoritmo proposto rotaciona as posições utilizadas na detecção de bordas para obter o melhor ângulo de projeção.

Neste caso, para que apenas um algoritmo seja capaz de obter sucesso em diferentes cenários de projeção, é necessário que uma parte do código esteja preparada para encontrar diferentes formas geométricas dentro da cena projetada. Pensando nisso, o algoritmo proposto utiliza uma forma de rotacionar as posições da vizinhança do *pixel* com o objetivo de encontrar diferentes formas geométricas de projeção das bordas dento da cena. Como resultado, todos os tipos de objetos podem ser detectados com esse procedimento.

Para o correto funcionamento do algoritmo proposto são necessárias algumas informações durante a chamada do método de *antialiasing*. Primeiramente, faz-se necessário que o método receba a imagem original representada por um modelo de cores, como mencionado na seção 2.1.3, o modelo RGB é o mais indicado e por isso o método proposto trabalha com cenas representadas nesse modelo.

Figura 20 - Projeção com o efeito *aliasing* e os canais de cores RGB. Em (a) a projeção original. Em (b) um *zoom* aplicado nas regiões de borda. Em (c) os canais de cores RGB.



Fonte: Autoria própria.

A Figura 20(a) é uma cena que possui círculos com diferentes diâmetros de projeção. Zoom foi aplicado nos locais de bordas (Figura 20(b)) para uma melhor visualização do efeito *aliasing* acometido nesta projeção. A Figura 20(c) representa os canais de cores do modelo RGB obtidos desta cena com a API OpenGL.

A Figura 21 apresenta o fluxograma para o método de *antialiasing* desenvolvido durante esse trabalho. Além das componentes R, G e B da cena, os parâmetros utilizados são o valor de variação do ângulo de rotação, o limiar de detecção de borda e a matriz de suavização. Sobre o funcionamento do método, na prática, percorre-se a matriz da imagem original e em cada *pixel* é realizado o cálculo de rotação das posições da vizinhança. Este procedimento tem como retorno as novas posições para serem utilizadas na operação de detecção de bordas (filtro passa-alta) e na suavização (filtro passa-baixa).

Utilizando as posições obtidas no processo de rotação obtêm-se as intensidades do *pixel* e de sua vizinhança para realizar o cálculo de detecção de bordas. O valor do módulo retornado deste cálculo é comparado com o maior módulo já obtido por rotações com ângulos diferentes realizadas anteriormente, no *pixel* e em sua vizinhança.

Figura 21 - Fluxograma do algoritmo de *antialiasing* baseado em filtragem especial de imagens.

Fonte: Autoria própria.

Ao final dessas interações, o maior módulo obtido com as rotações é comparado com limiar fornecido na chamada do método. Este limiar é valor mínimo para detecção das bordas da cena. Quanto maior for o limiar, maior será a restrição de detecção das bordas. Se o módulo for igual ou maior do que o limiar, então será realizada a suavização do pixel com base nas posições detectadas com a rotação que gerou o maior módulo. O resultado final é o pixel suavizado na região de borda com as melhores posições de rotação possíveis.

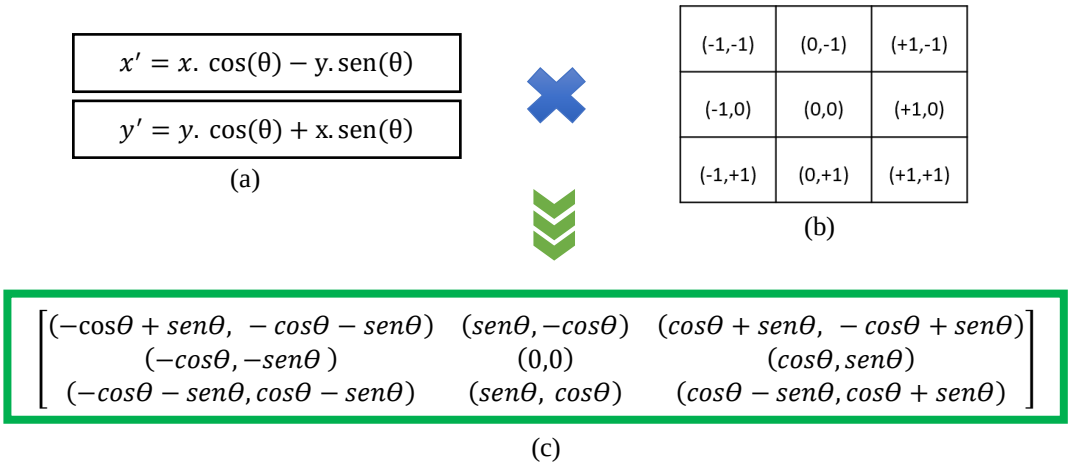
A implementação do algoritmo proposto está descrita no Apêndice A. O exemplo foi desenvolvido utilizando a linguagem de programação C++. As próximas seções deste capítulo irão explicar detalhadamente cada operação realizada.

3.1. A etapa de Rotação do Filtro Selecionado

Em cenários com diversas formas de objetos, um único método de *antialiasing* não seria o suficiente para tratar o *aliasing*. Por isso, a operação de rotação realizada no algoritmo

proposto é fundamental para torná-lo genérico. Neste sentido, a substituição do x e do y na equação de rotação (eq. (2.3)) pelas coordenadas do coeficiente do filtro (Figura 8) retornam as fórmulas de rotação para encontrar a nova coordenada de cada *pixel* na vizinhança, como ilustra a Figura 22. A intenção do algoritmo não é rotacionar as máscaras de detecção de bordas e suavização, mas sim utilizar as posições remanejadas da vizinhança do *pixel* nos cálculos.

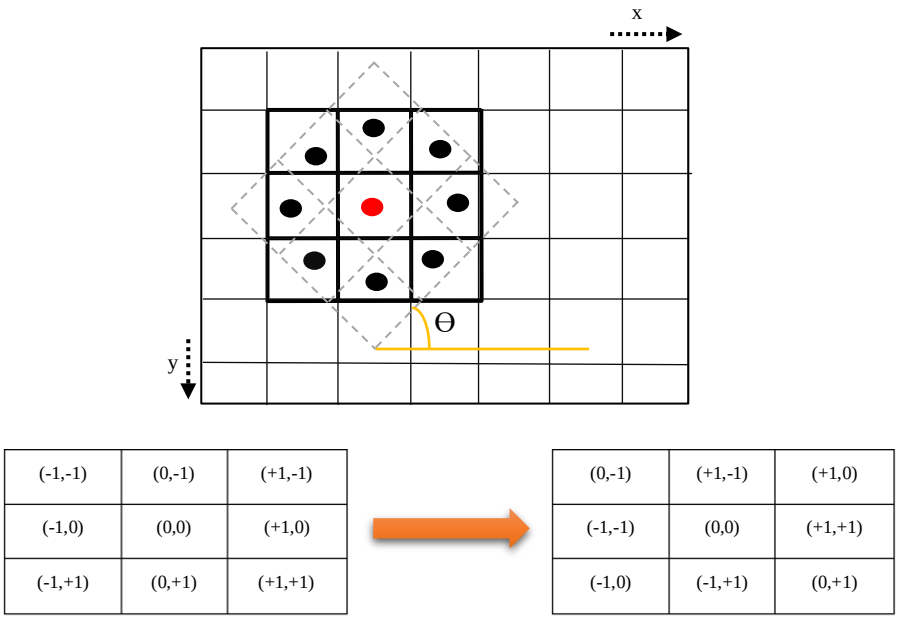
Figura 22 – Demonstrativo da obtenção da fórmula de rotação. Em (a) a equação de rotação. Em (b) os coeficientes do filtro. Em (c) a fórmula de rotação para cada coordenada do filtro.



Fonte: Autoria própria.

Na Figura 23 pode ser visualizado um exemplo de operação da matriz de rotação. Neste cenário a matriz do filtro selecionado é rotacionado em 45º. O resultado é uma rotação de uma casa no sentido anti-horário das posições da vizinhança do *pixel* selecionado.

Figura 23 - Visualização do filtro rotacionado com ângulo de 45º.



Fonte: Autoria própria.

O ângulo de rotação irá variar de 0° até um ângulo menor do que 180°. Neste caso, ângulo igual a 180° ou superior são reflexos de rotações já realizadas. O realismo de detecção está diretamente ligado com o ângulo de variação. Por isso, quanto menor o valor do ângulo de variação, mais realista a detecção de bordas e a suavização. Entretanto, o tempo de processamento aumentará devido a necessidade de processar mais interações.

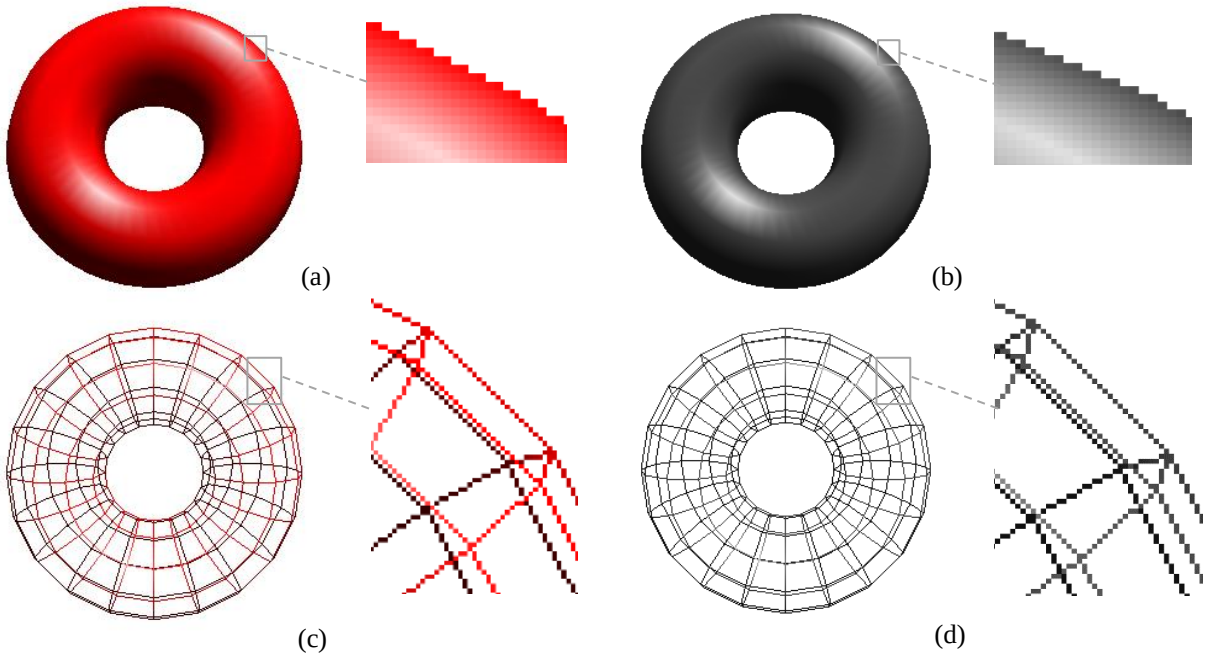
3.2. A etapa de obtenção da Intensidade dos *Pixels*

Para reduzir o tempo de processamento do método de passa-alta e passa-baixa, são obtidas as intensidades de cada pixel do filtro selecionado. Neste caso, a redução de tempo no processamento do algoritmo é mais de duas vezes a operação de passa alta e passa baixa devido a eliminação das operações em cada canal RGB para apenas o canal em escala de cinza. Para obter a intensidade de cada pixel do filtro, está sendo utilizada a fórmula de luminância (Y) do sistema de cor YIQ (eq. (3.1)). Esta operação transforma o modelo de cor RGB em escala de cinza.

$$Y = 0.299R + 0.587G + 0.144B \tag{3.1}$$

Na Figura 24 pode ser visualizado uma ilustração referente a este processo, no entanto, a operação é apenas realizada no filtro selecionado.

Figura 24 - O objeto Torus projetado. Em (a) Torus colorido. Em (b) Torus em escala de cinza. Em (c) linhas de projeção do Torus colorido. Em (d) linhas de projeção do Torus em escala de cinza.



Fonte: Autoria própria.

O Torus é um objeto disponibilizado pela biblioteca OpenGL. Este objeto está representando em colorido na Figura 24(a) e o mesmo objeto em escala de cinza Figura 24(b) obtido com a equação (3.1). Outro objeto disponibilizado pela API é a projeção dos contornos do Torus (Figura 24(c)). A mesma equação foi utilizada neste contorno do objeto para obter a imagem em escala de cinza (Figura 24(d)). Vale ressaltar que ainda não foi realizada nenhuma operação de suavização na projeção apresentada. A operação de suavização só ocorre na última etapa.

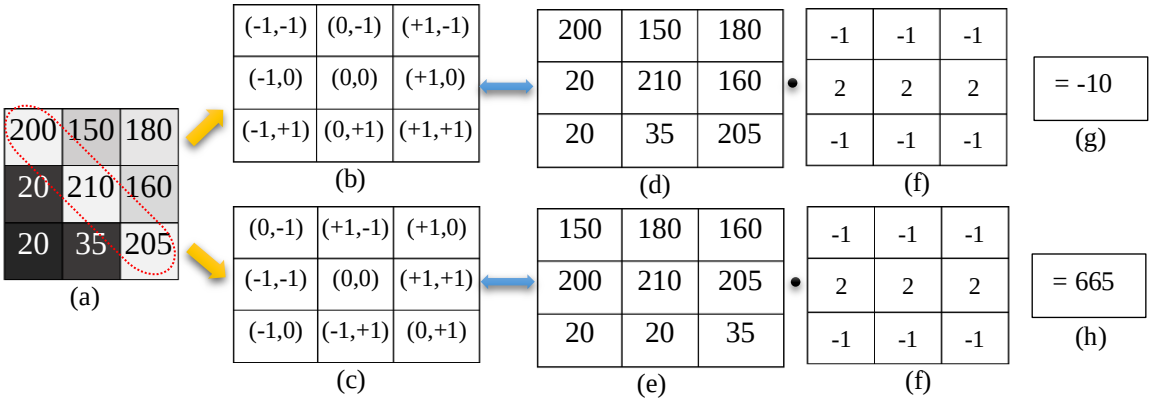
3.3. A etapa de Detecção de Bordas

O algoritmo obtém as posições de rotação para serem utilizadas no cálculo de detecção de bordas. Neste sentido, apenas uma máscara passa-alta horizontal é necessária para efetuar o processo. Esta máscara foi descrita na seção 2.1.4.2.

Na Figura 25 pode ser visualizada na prática a importância da rotação realizada nos coeficientes da vizinhança do *pixel* selecionado. Inicialmente, existe uma região posterior à

borda que necessita ser detectada pelo algoritmo (Figura 25(a)). A operação de convolução realizada entre o filtro selecionado na horizontal (Figura 25(b)) com a passa-alta horizontal (Figura 25(f)) retorna um módulo negativo (Figura 25(g)), isso significa que a projeção da borda não se encontra na horizontal ou aquela região não possui uma variação abrupta de intensidade.

Figura 25 - Exemplo de funcionamento da passa-alta com o filtro rotacionado. Em (a) a matriz intensidade da imagem original. Em (b) filtro normal. Em (c) filtro rotacionado em 45º. Em (d) intensidade dos *pixels* com posições na horizontal. Em (e) intensidade dos *pixels* com posições rotacionadas em 45º. Em (f) máscara de passa-alta horizontal. Em (g) módulo obtido com a operação de passa-alta com filtro na horizontal. Em (h) módulo obtido com a operação de passa-alta com filtro rotacionado em 45º.

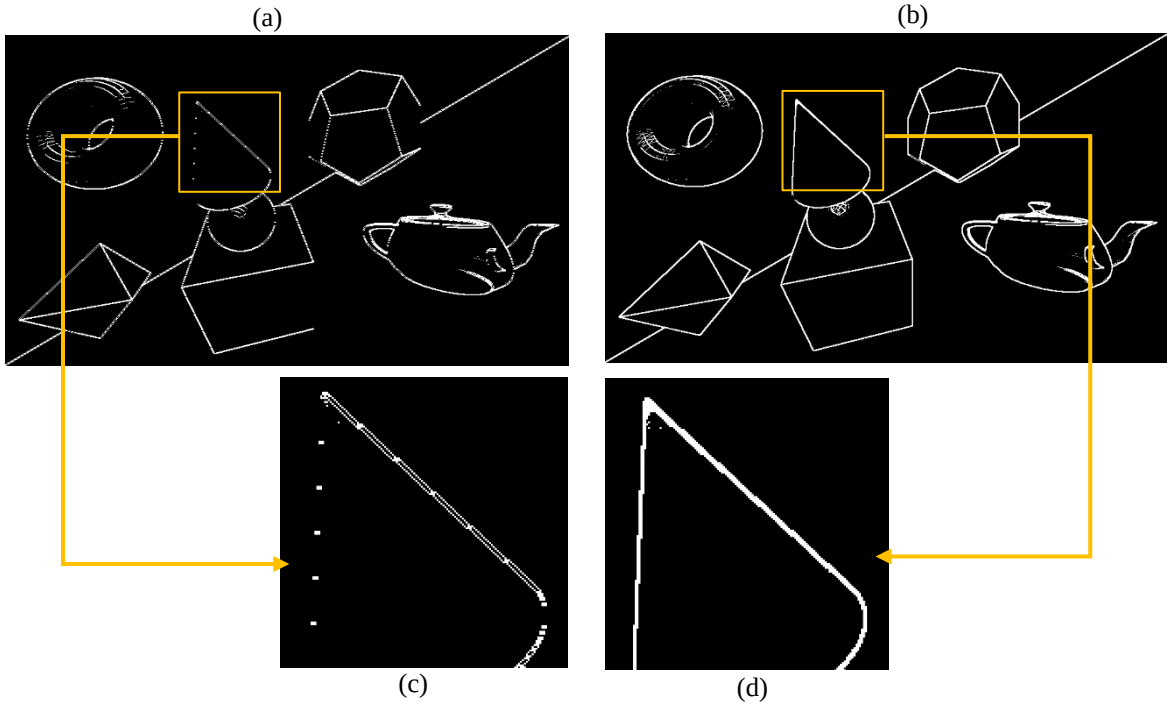


Fonte: Autoria própria.

Neste caso, apenas utilizando a máscara de passa-alta horizontal não é o suficiente para detectar a diagonal que faz divisa com a borda. Esta diagonal necessita ser suavizada para que a percepção da variação de intensidades de cores não seja visível, dessa forma reduzindo o efeito *aliasing*.

Apenas uma máscara de passa-alta criada para detectar projeções com rotação em 45º seria capaz de detectar essa região de variação abruptas. No entanto, o algoritmo proposto rotaciona o filtro selecionado em qualquer grau de rotação. No caso em exemplo, têm-se um filtro rotacionado em 45º (Figura 25(c)). A mesma operação realizada anteriormente, agora com o filtro rotacionado em 45º com a mesma passa-alta horizontal (Figura 25(f)) retorna um módulo positivo e muito superior obtido anteriormente (Figura 25(h)). Neste contexto, a utilização da máscara horizontal sem rotacionar as posições do filtro selecionado não é o suficiente para detectar a borda com ângulo de projeção diferente de 0º. Como pode ser visualizado na Figura 26.

Figura 26 – Aplicação de detecção de bordas com variação de ângulo diversos. Em (a) detecção de bordas apenas na horizontal (0º). Em (b) detecção de bordas com valor de variação de ângulo igual a 10º. Em (c) *zoom* aplicado em região com falha na detecção. Em (d) *zoom* aplicado em região com todas as bordas detectadas.

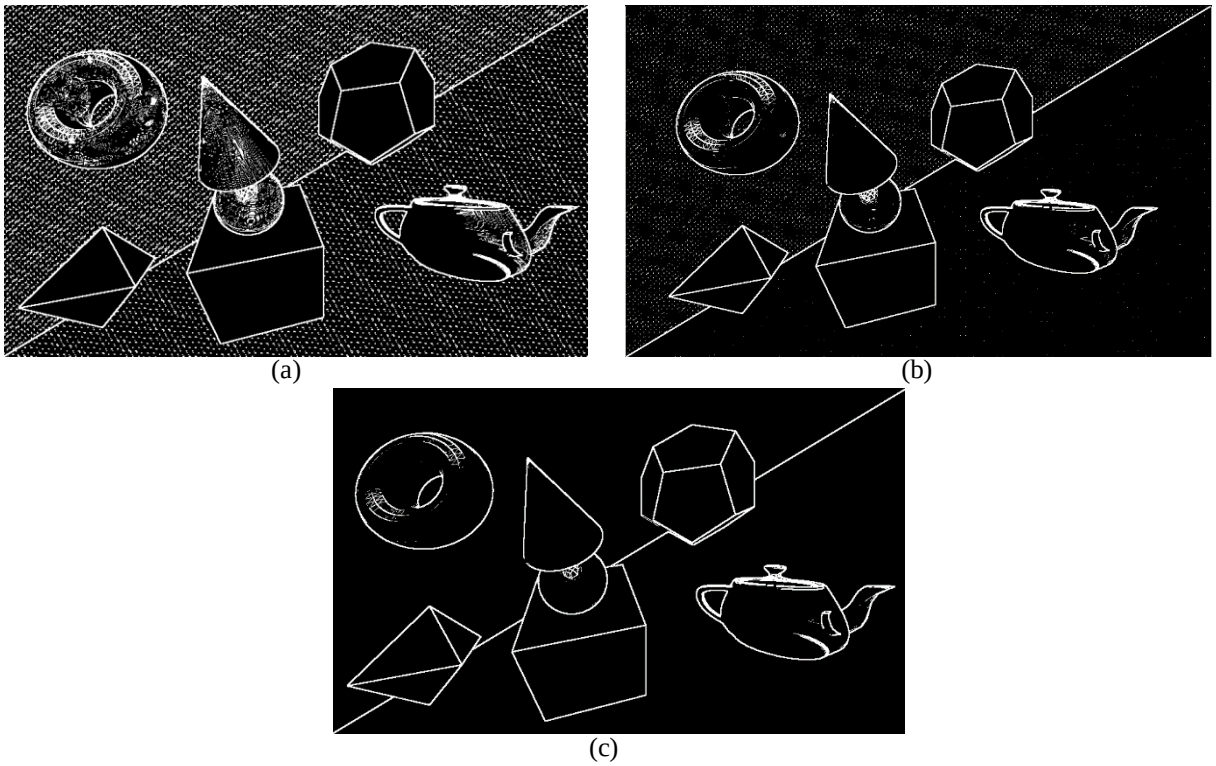


Fonte: Autoria Própria.

É necessário que o ocorra uma variação do ângulo de rotação para detectar todos graus de rotações possíveis. Na Figura 26(a) foi apenas utilizado o filtro na horizontal com a máscara de passa-alta horizontal. Neste sentido, não foi possível a detecção de todas as bordas da cena projetada. No entanto, com a mesma cena e agora com variação de rotação do ângulo em 10º foi possível detectar todas regiões de bordas da cena gráfica projetada, este resultado pode ser visualizado na Figura 26(b).

A borda não será sempre detectada quando o filtro for rotacionado. Isso dependerá de dois fatores: se ela é realmente uma borda, e se somente se, satisfaz a condição do limiar. Na Figura 27 foi aplicado três tipos de limiar diferentes. No primeiro caso foi aplicado um limiar 3 e o resultado foi a detecção de vários ruídos pela cena em geral. No segundo caso foi aplicado um limiar igual a 5 e o resultado ainda continua retornando uma certa quantidade de ruído. No entanto, quando foi aplicado um limiar igual a 8 o resultado retornou apenas as bordas da cena projetada.

Figura 27 - Aplicação de detecção de bordas com limiares diversos. Em (a) um limiar igual a 3. Em (b) um limiar igual a 5. Em (c) um limiar igual a 8.



(a) (b)

(c)

Fonte: Autoria própria.

Portanto, o módulo obtido deverá ser igual ou superior ao limiar informado. Quanto maior for o limiar, menor será o poder de detecção. Em contraponto, um fator positivo com o aumento do limiar é a redução da detecção de ruídos. Neste contexto, é ideal encontrar um limiar para cada cena projetada.

3.4. A etapa de Suavização do *Pixel*

A operação de suavização é última ser realizada pelo algoritmo proposto. Este processamento é o mais simples de todos. É aplicada a máscara de passa-baixa com o filtro obtido na detecção de borda (obtidas com as melhores posições de rotação) em cada canal de cor RGB. Na seção 2.1.4.1 foi descrita a passa-baixa utilizada neste processo. A equação (2.2) simplesmente obtém a média do *pixel* selecionado e dos *pixels* na vizinhança em cada canal de cor RGB. O valor obtido com operação em cada canal é transformado na nova cor do *pixel*. Este cálculo está demonstrado na equação (3.2).

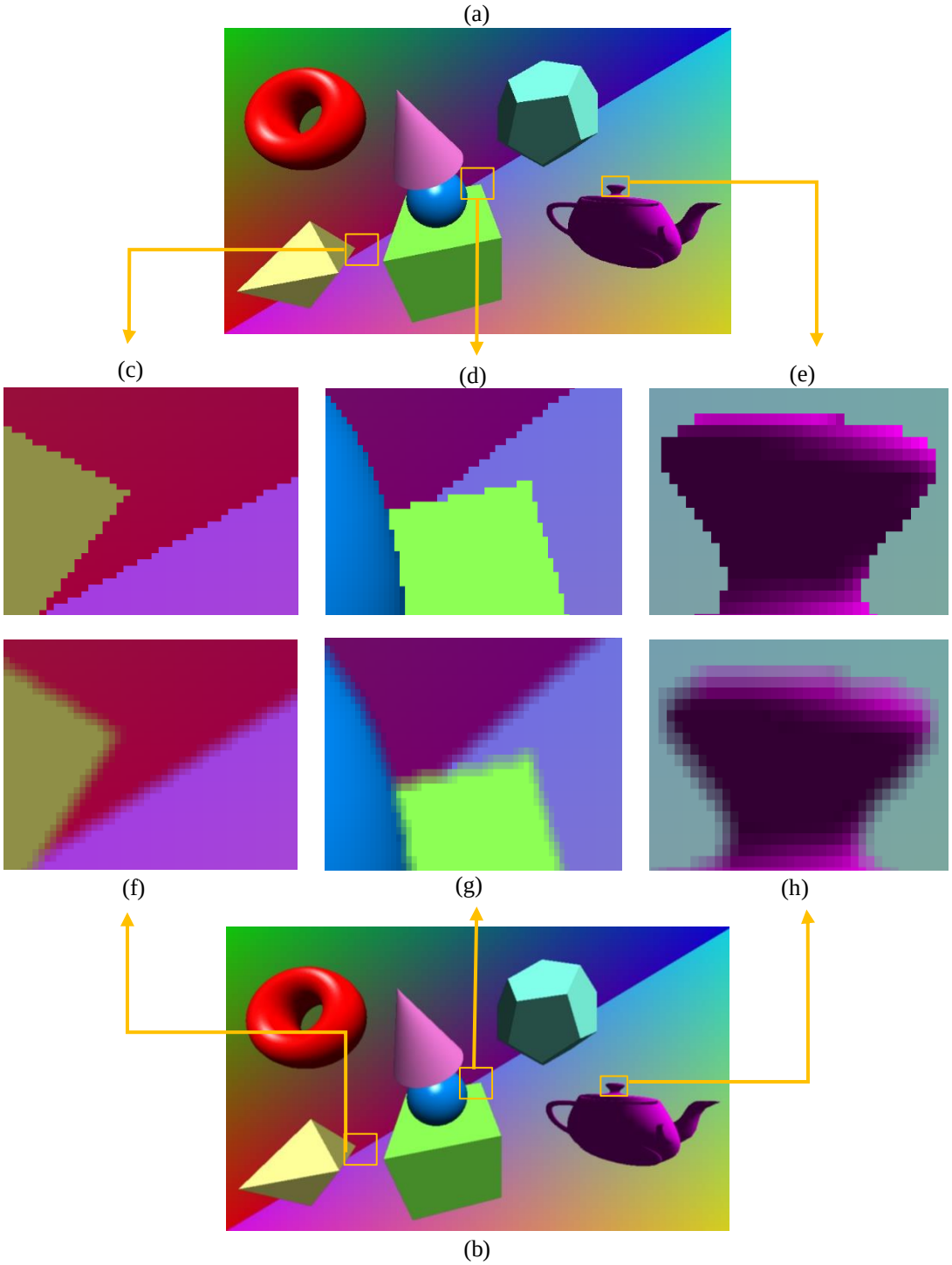
$$novo\ pixel = \left(\sum_{l=-a}^{a} \sum_{c=-b}^{b} R[(a,b)]p[(a,b)], \sum_{l=-a}^{a} \sum_{c=-b}^{b} G[(a,b)]p[(a,b)], \sum_{l=-a}^{a} \sum_{c=-b}^{b} B[(a,b)]p[(a,b)] \right) \quad (3.2)$$

Esta operação obtêm os novos valores de cada canal de cor RGB para formar o novo *pixel* da projeção no local selecionado. As variáveis a e b vão variar de acordo com o tamanho do filtro selecionado. Para um filtro com tamanho 3 x 3 as variáveis vão variar de -1 a 1 percorrendo todos os coeficientes do filtro selecionado. A variável p corresponde a passa-baixa utilizada. Uma operação de convolução irá ocorrer entre essas duas matrizes em cada canal de cor para formar o novo *pixel* da projeção.

O resultado final do processamento da cena em todos os *pixels* poder ser visualizado na Figura 28. Nesta figura está ocorrendo um confrontamento da cena com efeito *aliasing* (Figura 28(a)) e a mesma cena com a aplicação do método de *antialiasing* (Figura 28(b)). Foram aplicados *zooms* em alguns locais da cena para melhorar visualização do efeito *aliasing* e pela aplicação do método de *antialiasing* proposto. Esses locais foram escolhidos porque possuem mudanças abruptas de intensidade de cores.

O efeito *aliasing* deteriora por completo as regiões de borda das cenas projetadas (como pode ser visualizado nos *zooms* Figura 28(c)(d)(e)). Essa redução da qualidade das bordas da cena torna a projeção final muito inferior em qualidade visual. No entanto, com aplicação do método de *antialiasing* proposto é possível enganar os olhos humanos para obter uma melhor atratividade visual (ilustrados nos *zooms* Figura 28(f)(g)(h)). Esses locais são os mesmos do efeito *aliasing*, entretanto, nessas regiões ocorre uma redução da variação abruptas de intensidade.

Figura 28 - Resultado da aplicação do algoritmo de *antialiasing* proposto. Em (a) imagem original. Em (b) imagem com *antialiasing*. Em (c), (d) e (e) regiões selecionadas que possuem o efeito *aliasing*. Em (f), (g) e (h) as mesmas regiões tratadas com o método de *antialiasing*.



Fonte: Autoria própria.

A variação de uma cor para outra tornou-se gradativa reduzindo a percepção abrupta de cor. Com isso, diminuindo o efeito *aliasing* projetado na cena. A aplicação do método de

antialiasing mostrou-se eficaz na suavização do efeito. Essa redução apresentada melhora a qualidade visual final observada.

4 Conclusão e Trabalhos Futuros

Este trabalho propôs um algoritmo genérico de *antialiasing* capaz de suavizar o efeito *aliasing* gerado pela subamostragem da função contínua original. O algoritmo utiliza os métodos de filtragem espacial de imagens para amenizar dois problemas que estão presentes em alguns métodos de antialiasing, o consumo de memória e a detecção de diferentes formas geométricas dos objetos.

Todos os objetivos informados foram atingidos. O algoritmo consegue detectar todas as formas geométricas dos objetos que são geradas pelas cenas gráficas complexas apresentadas. Além de reduzir o consumo da memória e processamento utilizando os métodos de filtragem espacial de imagens nas operações do método.

O realismo de detecção de bordas e de suavização está diretamente ligados a dois fatores: o limiar de detecção e o ângulo de variação. Neste caso, é necessário que se encontre os melhores fatores para cada cena projetada. Como foi apresentado, qualquer variação desses dois fatores mudam os resultados obtidos.

Como trabalhos futuros é possível propor a incorporação de paralelismo no algoritmo proposto a fim de melhorar o seu desempenho computacional. Além disso, detectar quais etapas podem ser paralelizadas e que obtêm um maior rendimento. Outro fator importante é a comparação de execução com outros métodos propostos na literatura, verificar e comparar o tempo de execução, avaliar e comparar a utilização de memória e aplicar em diferentes hardwares.

Outro ponto a propor é automatização do limiar informado durante a chamada do método. Esse limiar também pode ser aplicado nos três canais de cores RGB com o intuito de verificar um melhor resultado da detecção de bordas. A utilização de diferentes tipos de passa-alta e passa-baixa nas cenas gráficas pode ser sugerido para a identificação dos melhores resultados de detecção das bordas e de suavização, além do aumento da matriz utilizada nas operações. Por fim, é proposto a utilização de alguma equação de estimação para determinar a cor do pixel real sem a ocorrência de arredondamentos.

Referências

AZEVEDO, E. CONCI, A. **Computação Gráfica: Geração de Imagens**, Rio de Janeiro, Elsevier, 2003.

BORESKOV, A. SHIKIN, E. **Computer Graphics – From Pixels to Programmable Graphics Hardware**, New Work, CRC Press, 2014.

CONCI, A. AZEVEDO, E. LETA, F. R. **Computação Gráfica: Teoria e Prática**, Rio de Janeiro, Elsevier, 2008.

EASTON, R. **Fundamentals of Digital Image Processing.** NY, USA, Rochester Institute of Technology, 2010.

FARIA, D. **Análise e Processamento de Imagem.** 2010. Disponível em: <https://web.fe.up.pt/~tavares/downloads/publications/relatorios/MEB_Diogo_Faria_TrabPraticos.pdf>. Acesso em: 28 jun. 2018.

FOLEY, J. D. DAM, A. V. FEINER, S. K. HUGHES, J. F. **Computer Graphics: Principles and Practice**, Crawfordville, Addison-Wesley, 1996.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing – Third Edition**. Upper Saddle River, NJ, USA: Prentice-Hall, 2006.

HEARN, D. BAKER, P. CARITHERS W. **Computer Graphics with OpenGL** – **Fourth Edition,** London, Pearson Education, 2014.

HUGHES, F. J. DAM, V. A. MCGUIRE, M. SKLAR, F. D. FOLEY, D. J. FEINER, K. S. AKELEY, K. **Computer Graphics: Principles and Practice - Third Edition,** London, Pearson Education, 2014.

HUNT, R. **The Reproduction of Colour - Sixth Edition**. Hoboken, NJ, USA, John Wiley, Sons, Ltd, 2004.

JANKE, S. J. **Mathematical Structures for Computer Graphics,** Hoboken, New Jersey, John Wiley & Sons, Inc. 2015.

KHRONOS, G. **OpenGL Overview.** 1997. Disponível em: <https://www.opengl.org/about/>. Acesso em: 16 jul. 2018.

LANG, B. **Oculus CTO Shares VR Dev Tip: The Formula for Avoiding Aliasing.** 2016. Disponível em: <https://www.roadtovr.com/avoiding-anti-aliasing-virtual-reality-gear-vr-oculus-rift-john-carmack/> Acesso em 11 jul. 2018.

MARQUES FILHO, O.; VIEIRA NETO, H. **Processamento Digital de Imagens**, Rio de Janeiro, Brasport, 1999.

OPENCV, E. **About**. 2018. Disponível em: <https://opencv.org/about.html>. Acesso em: 16 jul. 2018.

PRATT, K. W. **Digital Image Processing – Fourth Edition,** Los Altos, California, John Wiley & Sons, Inc. 2007.

RAPOSO, B. A. **Antialiasing – Eliminação de Superfícies Escondidas.** 2007. Disponível em: <http://webserver2.tecgraf.puc-rio.br/~abraposo/INF1366/> Acesso em: 07 Ago. 2018.

Apêndice A – Implementação do Algoritmo de Antialiasing

```
void antiAliasing(
    int width,
    int height,
    int a,
    int l,
    unsigned char * R,
    unsigned char * G,
    unsigned char * B,
    PIXEL * *result,
    char * passaBaixa
){

  //tamanho total do vetor matriz da imagem original
  int size = width * height;

  //tamanho dos filtros de agucamento e suavizacao
  unsigned char tamMatriz = 9;

  //passa alta horizontal (default)
  char passaAlta[] = {-1, -1, -1, 2, 2, 2, -1, -1, -1};

  //somatorio da matriz passa baixa
  unsigned char somatorioMatriz = somarMatriz(passaBaixa, tamMatriz);

  //variaveis temporarias
  double x, y;
  int pos, modulo;
  int dataModulo;
  int dataPosition[tamMatriz];
  int guardarPosicao[tamMatriz];
  unsigned char guardarY[tamMatriz];
  PIXEL * rgb;

  // percorrendo a matriz da imagem original
  for (int i = 0; i < size; i++){
    dataModulo = 0;
    // processo de rotacao
    for (int angulo = 0, radiano; angulo < 180; angulo += a){
      modulo = 0;
      radiano = angulo * PI / 180.0;

      //-1,1
      x = -cos(radiano) - sin(radiano);
      y = -sin(radiano) + cos(radiano);
      pos = buscarPosicao(i, width, height, round(x), round(y));
      guardarPosicao[0] = pos;
      guardarY[0] = obterIntensidade(R, G, B, pos);
      //0,1
      x = -sin(radiano);
      y = cos(radiano);
      pos = buscarPosicao(i, width, height, round(x), round(y));
      guardarPosicao[1] = pos;
```

```
53        guardarY[1] = obterIntensidade(R, G, B, pos);
54        //1,1
55        x = cos(radiano) - sin(radiano);
56        y = sin(radiano) + cos(radiano);
57        pos = buscarPosicao(i, width, height, round(x), round(y));
58        guardarPosicao[2] = pos;
59        guardarY[2] = obterIntensidade(R, G, B, pos);
60        //-1,0
61        x = -cos(radiano);
62        y = -sin(radiano);
63        pos = buscarPosicao(i, width, height, round(x), round(y));
64        guardarPosicao[3] = pos;
65        guardarY[3] = obterIntensidade(R, G, B, pos);
66        //0,0
67        guardarPosicao[4] = i;
68        guardarY[4] = obterIntensidade(R, G, B, i);
69        //1,0
70        x = cos(radiano);
71        y = sin(radiano);
72        pos = buscarPosicao(i, width, height, round(x), round(y));
73        guardarPosicao[5] = pos;
74        guardarY[5] = obterIntensidade(R, G, B, pos);
75        //-1,-1
76        x = -cos(radiano) + sin(radiano);
77        y = -sin(radiano) - cos(radiano);
78        pos = buscarPosicao(i, width, height, round(x), round(y));
79        guardarPosicao[6] = pos;
80        guardarY[6] = obterIntensidade(R, G, B, pos);
81        //0,-1
82        x = sin(radiano);
83        y = -cos(radiano);
84        pos = buscarPosicao(i, width, height, round(x), round(y));
85        guardarPosicao[7] = pos;
86        guardarY[7] = obterIntensidade(R, G, B, pos);
87        //1,-1
88        x = cos(radiano) + sin(radiano);
89        y = sin(radiano) - cos(radiano);
90        pos = buscarPosicao(i, width, height, round(x), round(y));
91        guardarPosicao[8] = pos;
92        guardarY[8] = obterIntensidade(R, G, B, pos);
93
94        //obtendo o modulo de deteccao a partir do angulo respectivo.
95        for (int k = 0; k < tamMatriz; k++){
96          modulo += (guardarY[k] * passaAlta[k]);
97        }
98
99        //verificando o melhor modulo de deteccao e guardando as informacoes da
melhor rotacao com base no modulo obtido.
100       if (abs(modulo) > dataModulo){
101         dataModulo = abs(modulo);
102         for (int k = 0; k < tamMatriz; k++){
103           dataPosition[k] = guardarPosicao[k];
104         }
105       }
106     }
107
108     //informacoes do pixel original (para os pixels que nÃ£o estao em borda)
109     rgb = new PIXEL;
110     rgb->r = R[i];
```

```
111     rgb->g = G[i];
112     rgb->b = B[i];
113     rgb->l = 0;
114     rgb->x = i % width;
115     rgb->y = i / width;
116 
117     //operacao de suavizacao, se 'modulo' atender o criterio do limiar minimo de
deteccao.
118     if (dataModulo >= l){
119       rgb->r = 0;
120       rgb->g = 0;
121       rgb->b = 0;
122 
123       for (int k = 0; k < tamMatriz; k++){
124         pos = dataPosition[k];
125         // se a posicao for nula, pula a operacao.
126         if(pos != size){
127           rgb->r += round((R[pos] * passaBaixa[k]/somatorioMatriz));
128           rgb->g += round((G[pos] * passaBaixa[k]/somatorioMatriz));
129           rgb->b += round((B[pos] * passaBaixa[k]/somatorioMatriz));
130         }
131       }
132       rgb->l = 255;
133     }
134 
135     // salvando o pixel no vetor resultado
136     result[i] = rgb;
137   }
138 }
139 
140 // descobre a posicao do pixel no vetor (bidimensional para unidimensional)
141 int buscarPosicao(int i, int width, int height, int x, int y){
142   int v;
143 
144 
145   // detectando a posicao no vetor
146   if (x == -1 && y == 1){
147     v = i + width - 1;
148   }
149   if (x == 0 && y == 1){
150     v = i + width;
151   }
152   if (x == 1 && y == 1){
153    v = i + width + 1;
154   }
155   if (x == -1 && y == 0){
156     v = i - 1;
157   }
158   if (x == 0 && y == 0){
159     return i;
160   }
161   if (x == 1 && y == 0){
162     v = i + 1;
163   }
164   if (x == -1 && y == -1){
165     v = i - width - 1;
166   }
167   if (x == 0 && y == -1){
168     v = i - width;
```

```
169   }
170   if (x == 1 && y == -1){
171     v = i - width + 1;
172   }
173 
174   int xP = v % width;
175   int yP = v / width;
176 
177   //verificacao dos limites da imagem
178   if(xP < 0 || xP >= width || yP < 0 || yP >= height){
179     // retorno com valor direcionado para uma posicao nula (0).
180     return height*width;
181   }else return v;
182 }
183 
184 // obter a luminancia (Y) do pixel qualquer
185 int obterIntensidade(
186     unsigned char *dataRed,
187     unsigned char *dataGreen,
188     unsigned char *dataBlue,
189     int pos){
190   return round(0.299 * dataRed[pos] + 0.587 * dataGreen[pos] + 0.114 *
    dataBlue[pos]);
191 }
192 
193 // obter o somatorio dos valores dos filtros passa baixa
194 unsigned char somarMatriz(char * matriz, unsigned char tam){
195   unsigned char valor = 0;
196   for(int i = 0; i < tam; i++){
197     valor += matriz[i];
198   }
199 
200   return valor;
201 }
```