UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

ELDREY SEOLIN GALINDO

APLICAÇÃO WEB PARA AUXÍLIO A DALTÔNICOS COM DEUTERANOPIA ATRAVÉS DA ANÁLISE DE CONTRASTE E RECOLORAÇÃO DE IMAGENS

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2015

ELDREY SEOLIN GALINDO

APLICAÇÃO WEB PARA AUXÍLIO A DALTÔNICOS COM DEUTERANOPIA ATRAVÉS DA ANÁLISE DE CONTRASTE E RECOLORAÇÃO DE IMAGENS

Trabalho de Conclusão de Curso de Bacharelado em Sistemas de Informação apresentado ao Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Bogdan Tomoyuki Nassu

CURITIBA

2015

"Uma imagem vale mais que mil palavras"

Confúcio

RESUMO

GALINDO, Eldrey S.. APLICAÇÃO WEB PARA AUXÍLIO A DALTÔNICOS COM DEUTERANOPIA ATRAVÉS DA ANÁLISE DE CONTRASTE E RECOLORAÇÃO DE IMAGENS. 53 f. Trabalho de Conclusão de Curso – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

O daltonismo atinge cerca de 8% da população masculina e 1 a cada 200 mulheres no mundo. Na *Web*, pessoas com daltonismo encontram diversas dificuldades para a compreensão de conteúdos em imagens, visto que elas frequentemente são produzidas sem levar em consideração tal característica. Neste projeto, foi desenvolvida uma aplicação que permite uma melhor percepção do conteúdo de imagens em páginas *Web* por pessoas com daltonismo. Para isso, foi desenvolvida uma extensão para o navegador Firefox e um servidor de aplicação, que permitem o uso de uma versão otimizada de um algoritmo para recoloração de imagens. A aplicação visa ser de fácil utilização, mantendo todo o processamento das imagens invisível para o usuário.

Palavras-chave: Daltonismo. Processamento de imagens. Web. Deuteranopia.

ABSTRACT

GALINDO, Eldrey S.. WEB APPLICATION TO ASSIST THE COLORBLIND WITH DEUTERANOPIA BY CONTRAST ANALYSIS AND IMAGE RECOLORING . 53 f. Trabalho de Conclusão de Curso – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

Color blindness affects about 8% of men and 1 in 200 women in the world. In the Web, people with color blindness has several difficulties in understanding content in images, since they are often produced without taking into account such feature. In this project, it was developed a program that allows a better understanding of the content of images in Web pages by people with color blindness. For this, it was developed an extension for Firefox browser and an application server, which allows the use of an optimized version of an algorithm to image recoloring. The application aims to be user-friendly, while maintaining all the processing of the images invisible to the user.

Keywords: Color blindness. Image processing. Web. Deuteranopia.

AGRADECIMENTOS

Agradeço à Universidade Tecnológica Federal do Paraná, Campus de Curitiba, especialmente ao meu orientador, professor Bogdan Tomoyuki Nassu, pela orientação e juntamente com o professor Leonelo Dell Anhol Almeida, por terem proposto o tema desse trabalho. Gostaria de agradecer também a todos os amigos e familiares, em especial aos meus colegas de curso, Allan Vinicius Ribas Wandratsch, Henrique Monteiro Rogich Jasinski e Jeam Martins Alves, por terem me apoiado durante o desenvolvimento do trabalho.

LISTA DE FIGURAS

LISTA DE SIGLAS

nm	nanômetros
RGB	R - *Red*, G - *Green*, B - *Blue* - Vermelho, Verde, Azul
CIE	*Commission Internationale de l'Eclairage* Comissão Internacional de Iluminação
REST	*Representational State Transfer*, Transferência de Estado Representacional
Json	*JavaScript Object Notation*, Notação de Objeto JavaScript
W3C	World Wide Web Consortium
XUL	*XML User Interface Language* - Linguagem de interface do usuário XML

SUMÁRIO

1 INTRODUÇÃO

O daltonismo é a falta de sensibilidade para distinguir total ou parcialmente algumas cores. Esta condição atinge cerca de 8% da população masculina do planeta e algo em torno de uma a cada duzentas mulheres [8].

As pessoas que possuem daltonismo enfrentam várias dificuldades em executar tarefas que são consideradas simples pelas pessoas não daltônicas como, por exemplo, combinar a cor da camisa com a calça ou gravata, identificar se uma criança está com insolação, distinguir entre as cores do semáforo ou identificar as linhas de ônibus de uma cidade [8]. Na *Internet* estas dificuldades são normalmente ressaltadas, uma vez que, visando facilitar o entendimento de determinados conteúdos, as páginas *Web* costumam apresentar diversas imagens e gráficos, e em raros casos estas páginas são criadas com o cuidado de serem legíveis para os daltônicos [9]. Um exemplo desta dificuldade pode ser visto na Figura 1, que ilustra como um daltônico com deuteranopia, incapacidade de discernir o vermelho do verde, enxerga um mapa.

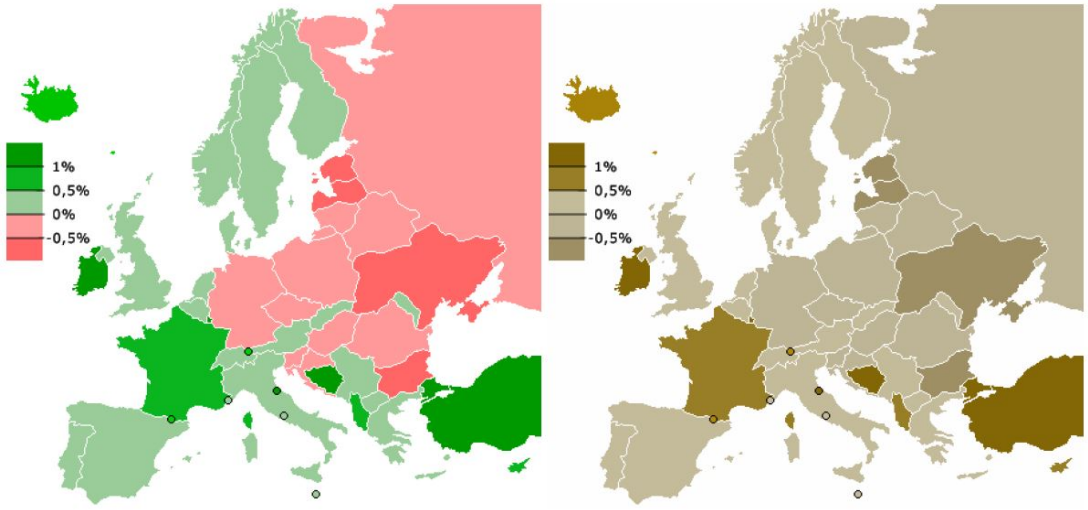


Figura 1: Visão de um mapa por uma pessoa com deuteranopia [1].

Existem basicamente três tipos de daltonismo: (1) Tricromatismo anômalo: quando

se percebe a cor branca através de proporções anômalas das três cores (vermelho, verde ou azul), resultando na confusão do vermelho com o preto, do vermelho com o verde ou a perda de sensibilidade do azul; (2) Dicromatismo: percebe-se a cor branca a partir do estímulo de apenas duas cores, gerando a dificuldade de se perceber uma das três cores (vermelho, verde ou azul) e (3) Monocromatismo: percebe-se a cor branca com qualquer estímulo, gerando uma percepção de apenas escalas de cinza ou em casos mais graves apenas a percepção do preto e do branco [10].

Apesar da quantidade de pessoas afetadas por algum tipo de daltonismo ser relativamente alta, os tipos de daltonismo afetam esta população de forma diferente. A Tabela 1 mostra a porcentagem de cada tipo de daltonismo na população mundial.

Tabela 1: Incidência de diferentes tipos de daltonismo na população mundial [1].

Classificação	**Porcentagem (%)**	
	Homens	**Mulheres**
Tricromatismo anômalo	**5.9**	**0.37**
Protanomalia (falha nos cones vermelhos)	1.0	0.02
Deuteranomalia (falha nos cones verdes)	4.9	0.35
Tritanomalia (falha nos cones azuis)	0.0001	0.0001
Dicromatismo	**2.1**	**0.03**
Protanopia (cones vermelhos ausentes)	1.0	0.02
Deuteranopia (cones verdes ausentes)	1.1	0.01
Tritanopia (cones azuis ausentes)	0.001	0.001
Monocromatismo	**0.003**	**0.00001**

Com o intuito de auxiliar a compreensão das imagens e gráficos nas páginas *Web* por usuário daltônicos, foram criados vários aplicativos e extensões [11] [12] [13]. A maioria destas aplicações utiliza filtros para alterar as imagens. Em alguns casos, a extensão possui uma configuração fixa para o filtro, e em outros casos a configuração é feita pelo usuário usando uma imagem como base. Entretanto, em ambos os casos, o mesmo filtro é aplicado a todas as imagens, alterando igualmente imagens distintas - às vezes até sem necessidade.

O objetivo deste trabalho é auxiliar as pessoas com daltonismo, facilitando a compreensão das imagens e gráficos nas páginas *Web*. Para tanto, foi desenvolvida uma aplicação *Web* com uma extensão para o navegador Firefox que altera as imagens de forma individual, minimizando a perda de detalhes. O projeto foi desenvolvido visando as pessoas com deuteranopia, uma vez que a mesma pode ser considerada um caso mais extremo da deuteranomalia, que por sua vez é o tipo de daltonismo mais comum. Entretanto, os conceitos aqui utilizados poderão ser estendidos para os outros tipos de daltonismo. Este trabalho se torna relevante para a comunidade, uma vez que não foi encontrada até o

momento nenhuma extensão capaz de alterar as imagens de uma pagina *Web* de forma individual.

O desenvolvimento do projeto teve como base o trabalho publicado por Machado e Oliveira em 2010 [1][14], onde é descrito um algoritmo de processamento de imagens que, com base na rotação do espaço de cor La*b*, altera as cores de cada imagem permitindo assim a compreensão da mesma pelos daltônicos. O presente projeto consiste não apenas na replicação do algoritmo descrito, mas também na otimização[1] e incorporação em uma extensão para navegadores *Web*. Para isso, foram utilizadas técnicas de otimização de código e uma modificação do algoritmo, que consiste no processamento em uma versão reduzida da imagem, que é depois recombinada com os dados de luminância da imagem original. A Figura 2 mostra um exemplo de imagem processada, sendo (a) a imagem original, (b) uma simulação da imagem original vista por um daltônico, (c) o resultado do algoritmo implementado e (d) a simulação da imagem resultante do algoritmo vista por uma pessoa daltônica.

Nos experimentos realizados, a aplicação *Web* foi capaz de modificar uma imagem de 1632x1224 em aproximadamente 0,062 minutos, enquanto a versão original necessitou de 272,131 minutos, 99,98% mais rápido do que a versão não otimizada. Sendo que todos os testes foram realizados em um notebook com processador Intel Core I5, 8Gb de memória RAM e placa de vídeo ATI Radeon HD4650 1gb GDDR3. Comparações feitas com uma métrica objetiva e uma base de dados de 120 imagens, mostraram que os resultados obtidos pela versão otimizada do algoritmo são próximos dos obtidos pela versão original. Em uma escala de -127 á 127, as comparações apresentaram uma média de 2,7 e um desvio padrão de 2,45.

[1]É considerado uma otimização do algoritmo, a melhora do algoritmo que gere um tempo de processamento menor, desde que a imagem resultante seja visualmente semelhante a imagem resultante do algoritmo não otimizado.

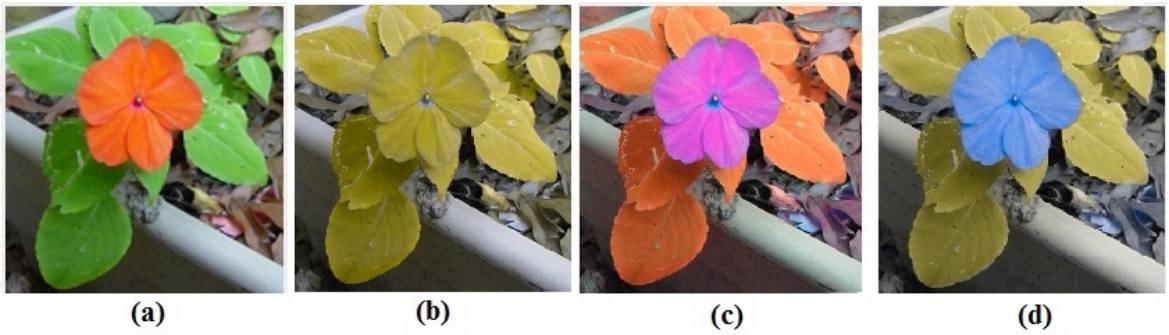


Figura 2: (a) Uma imagem, (b) o resultado da simulação da imagem original visto por um daltônico, (c) o resultado original do algoritmo implementado e (d) a simulação da imagem resultante do algoritmo visto por uma pessoa daltônica. Fonte: Autoria própria

O restante do trabalho se divide da seguinte forma. No Capítulo 2, é feita uma breve descrição da percepção das cores pelos seres humanos e os tipos existentes de daltonismo. No Capítulo 3, são descritos os trabalhos relacionados. O Capítulo 4 consiste nas definições necessárias para a compreensão deste trabalho. O Capítulo 5 é composto pela descrição das fases do projeto, a criação do simulador de dicromatismo (vermelho e verde), a aplicação para *desktop*, e a extensão para o navegador Firefox, apresentando também os resultados obtidos. Por fim o Capítulo 6 apresenta a conclusão do trabalho.

2 O DALTONISMO

De uma forma breve, será descrito neste capítulo como os seres humanos percebem as cores, enfatizando o funcionamento do olho. Serão abordados também os tipos de daltonismo e como essa dificuldade na percepção das cores afeta as pessoas na utilização da *Web* e de sistemas computacionais.

2.1 PERCEPÇÃO DAS CORES

Um dos órgãos responsáveis pela visão humana é o olho. Ele tem a função de converter a luz refletida pelos objetos em pulsos elétricos que são transmitidos ao cérebro. A luz emitida pelos objetos é percebida através de dois tipos de células, os cones que se encontram no centro da retina, e os bastonetes que se encontram na região periférica da retina [15]. Os bastonetes são responsáveis por perceber a alteração de luminância dos objetos, ou seja o quão claros ou escuros os objetos estão, já os cones são responsáveis por perceber as cores dos objetos.

Os cones são subdivididos em três tipos de acordo com a frequência a que são mais sensíveis, os *Long*-cones ou L-cones, são mais sensíveis a frequências longas, os *Medium*-cones ou M-cones, têm uma sensibilidade maior às frequências medianas e os *Short*-cones ou S-cones são mais sensíveis à frequências curtas. A Figura 3 demonstra essa divisão e associa as frequências em nanômetros (nm) com as cores que elas representam [2].

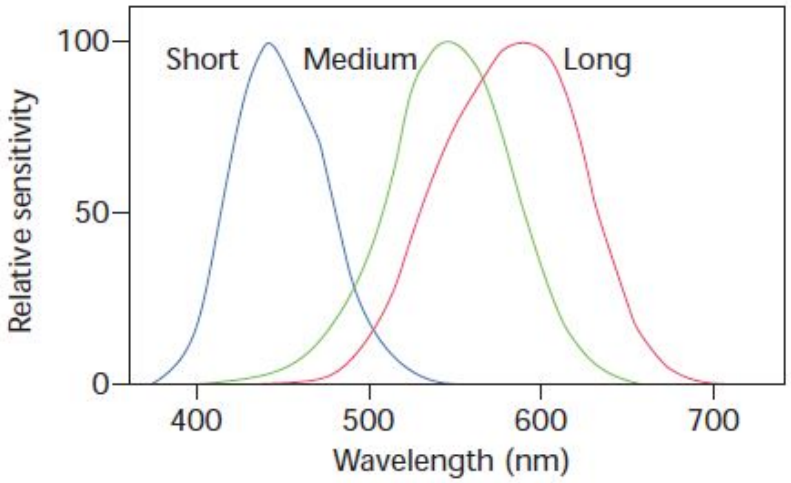


Figura 3: Sensibilidade dos cones (L,M e S) aos comprimentos de onda, em nm [2].

2.2 DALTONISMO

O daltonismo é a dificuldade na percepção de uma ou mais cores. Existem basicamente três tipos de daltonismo [10]:

- Tricromatismo anômalo: quando se percebe a cor branca através de proporções anômalas das três cores (vermelho, verde e azul). O tricromatismo anômalo é subdividido em:

  Protanomalia: é o escurecimento das cores próximas às frequências mais baixas. Resulta na confusão entre o vermelho e o preto.

  Deuteranomalia: é a dificuldade em diferenciar a cor verde, normalmente resultando na confusão do vermelho com o verde.

  Tritanomalia: é a perda de sensibilidade da cor azul.

- Dicromatismo: percebe-se a cor branca a partir do estímulo de apenas duas cores. O Dicromatismo é subdividido em:

  Protanopia: é a dificuldade na percepção da cor vermelha;

  Deuteranopia: é a dificuldade de percepção da cor verde;

  Tritanopia: é a dificuldade na percepção da cor azul ou amarela.

- Monocromatismo: percebe-se a cor branca com qualquer estímulo. Nestes casos tem-se a dificuldade de perceber a intensidade das cores, sendo sensível apenas à

claridade dos objetos. Em casos mais raros a percepção dos objetos é feita apenas em tons de cinza ou em preto e branco.

A Figura 4 mostra como cada um dos tipos de daltonismo afeta a percepção das cores.

Figura 4: Simulação dos tipos de daltonismo [3].

A falta de sensibilidade na percepção das cores pode ser congênita, quando o feto adquire o daltonismo devido a problemas na sua formação ou de forma hereditária, ou adquirida, quando a dificuldade se origina da diminuição de cones e bastonetes, alterações na córnea ou na retina, e alterações no sistema de pós-receptores da imagem [10].

2.3 TESTE DE ISHIHARA

O teste de Ishirara foi criado em 1906 e é considerado o teste mais eficaz para a detecção de daltonismo. O teste consiste em uma imagem formada por círculos preenchidos com duas cores, que variam de acordo com o tipo de daltonismo que se quer validar. A Figura 5 mostra um exemplo do teste para detecção de deuteranopia (vermelho e verde).

Do lado esquerdo está a imagem original, já do lado direito está a simulação da imagem vista por um daltônico com dicromatismo (vermelho e verde).

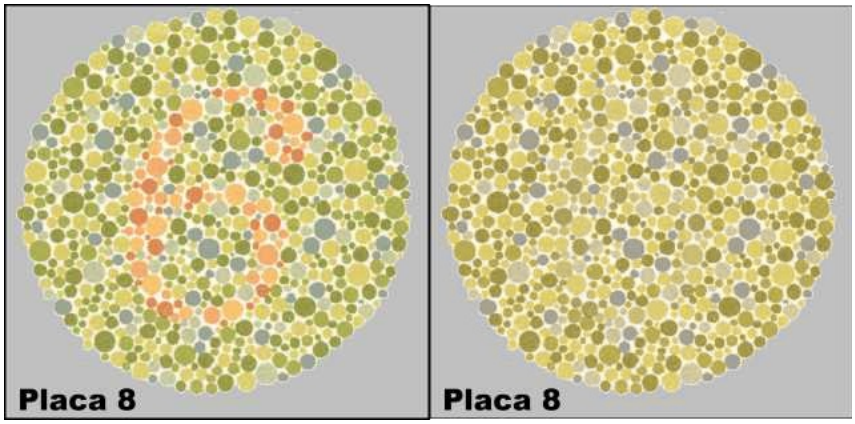


Figura 5: Exemplo de teste de Ishihara [4].

3 TRABALHOS RELACIONADOS

No decorrer deste trabalho foram encontrados alguns programas, aplicativos e extensões relacionados. A analise destas aplicações, realizada neste capítulo, faz parte do acompanhamento do estado da arte e auxilia no aperfeiçoamento deste trabalho.

3.1 *COLORBLINDING*

O *Colorblinding* é uma extensão criada em 2015 para o navegador Google Chrome. Ela tem como objetivo auxiliar os desenvolvedores de páginas e conteúdos *Web* a gerarem estes conteúdos de maneira acessível às pessoas com daltonismo [11].

Esta extensão, quando ativa no navegador, aplica filtros nas imagens que estão sendo exibidas. A alteração dessas cores visa simular como uma pessoa com um tipo determinado de daltonismo enxerga o conteúdo da página, como ilustra na Figura 6, onde a imagem original (a) e a alterada pela extensão (b).

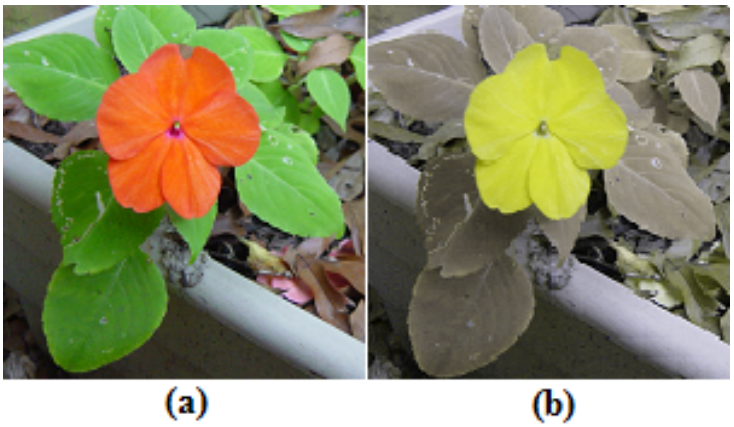


Figura 6: Resultado da simulação feita pela extensão *Colorblinding*. Fonte: Autoria própria

O código do *Colorblinding* é aberto e encontra-se disponível no GitHub [11].

3.2 COBLIS - COLOR BLINDNESS SIMULATOR

Diferente do Colorblinding, o Coblis é um simulador que se encontra em uma página *Web*. Seu objetivo deste simulador é simular vários tipos de daltonismo em uma imagem especifica. Para isso o Coblis necessita que seja feito o *upload* da imagem para o servidor onde página encontra-se hospedada. Após este *upload*, é mostrada no site a imagem simulada para o tipo de daltonismo escolhido, como é possível observar na Figura 7, onde (a) demonstra a imagem sendo submetida ao site e (b) a mesma imagem simulada pelo site para deuteranopia [12].

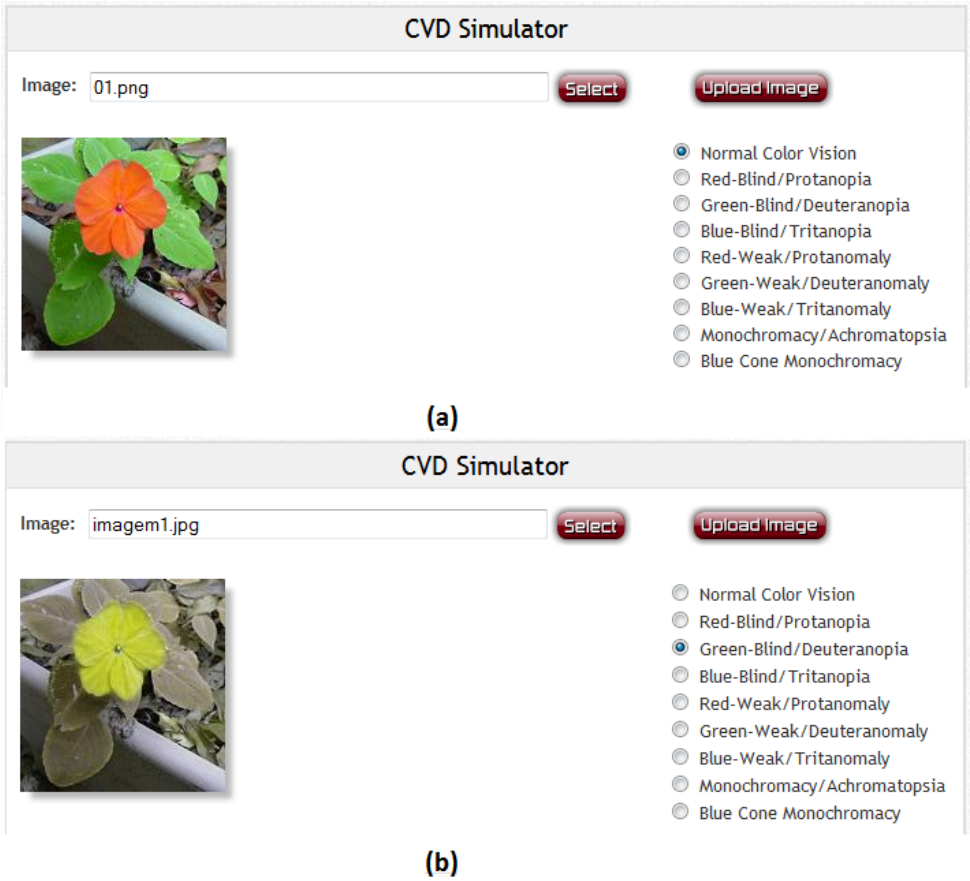


Figura 7: Resultado da simulação feita pelo Coblis Fonte: Autoria própria.

Este simulador encontra-se hospedado no link `http://www.color-blindness.com/coblis-color-blindness-simulator`

O Coblis auxilia não apenas na construção de páginas *Web*, mas também no desenvolvimento de qualquer conteúdo. Apesar deste auxilio, poucos conteúdos na *Web* são acessíveis para daltônicos.

3.3 COLORBLINDS TOOL

Este *plug-in* para o navegador Firefox cria um atalho no menu para a pagina Colorblinds.org. O objetivo deste *plug-in* é permitir uma melhor compreensão das imagens pelos daltônicos. Para isso, é possível fazer *upload* de uma imagem qualquer para a página que a ferramenta indica. Uma vez esse *upload* feito, a ferramenta altera a luminância da imagem, permitindo a visualização de detalhes da imagem antes não percebidos. A Figura 8 mostra o resultado de um teste feito com essa ferramenta, onde a imagem (a) é a imagem original, e a imagem (b) processada pelo Colorblinds.org [13].

Figura 8: Comparação da imagem original (a) com a imagem resultante da ferramenta Colorblinds.org (b) Fonte: Autoria própria.

Conforme o usuário passa o ponteiro do mouse sobre a imagem, o *plug-in* apresenta o nome e o código da cor em hexadecimal, como pode ser observado na Figura 9.

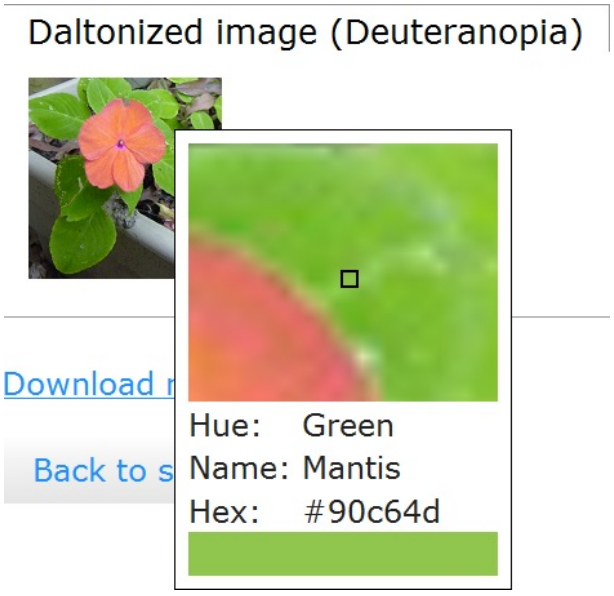


Figura 9: Ferramenta Colorblinds.org Fonte: Autoria própria.

O fato do *plug-in* mudar apenas a luminância das imagens restringe consideravelmente os tipos de daltonismo que esta ferramenta auxilia. Além disso o usuário precisa enviar imagens a um site externo, de forma que o *plug-in* aumenta consideravelmente as interações necessárias para se obter o resultado. Desta forma, o *plug-in* não cumpre com os objetivos propostos neste trabalho.

3.4 REAL-TIME TEMPORAL-COHERENT COLOR CONTRAST ENHANCEMENT FOR DICHROMATS

Este trabalho foi desenvolvido em 2010 por Machado e Oliveira [14], sendo parte da dissertação de mestrado de Machado [1], premiada no concurso de teses e dissertações da Sociedade Brasileira de Computação em 2011 [14]. Devido à alta qualidade e capacidade do algoritmo de permitir a compreensão das imagens por pessoas com daltonismo, esta dissertação é a principal base para o desenvolvimento do presente trabalho. Desta forma, este algoritmo será descrito em detalhes no Capítulo 5.

O algoritmo descrito na dissertação utiliza técnicas de processamento de imagens para alterar as cores da imagem com base no contraste, de forma a permitir a compreensão da imagem por pessoas com determinados tipos de daltonismo. O processamento de imagens utilizado tem um elevado custo computacional, por isso, o programa foi desenvolvido para rodar diretamente em GPUs (*Graphics Processing Unit* - Unidades de Processamento Gráfico).

A Figura 10 demonstra o resultado do processamento da imagem original que está à esquerda. A imagem ao centro é o resultado da simulação para dicromatismo (vermelho e verde) da imagem original, já a imagem a direita é o resultado final do processamento ponto de vista de uma pessoa com dicromatismo (vermelho e verde).

Figura 10: Resultado do processamento da imagem apresentado por Machado e Oliveira [1].

4 DEFINIÇÕES

Este capitulo apresenta algumas definições importantes para a compreensão do trabalho. Serão apresentadas definições para um pixel, uma imagem, e para os espaços de cor RGB e La*b*.

4.1 IMAGEM E PIXEL

Uma imagem pode ser definida como sendo uma matriz M_{mXn}, onde m representa a largura ou quantidade de colunas e n a altura ou quantidade de linhas.

A imagem é composta por pontos chamados de *pixels*. Um pixel, no contexto de imagens coloridas, é comumente representado como um vetor p tridimensional, relacionado com a posição da matriz M_{mXn} através de uma função bidimensional $f(x,y)$, sendo $0 \leq x < m$ e $0 \leq y < n$. A posição $f(0,0)$ representa o canto superior esquerdo, como mostra a equação 1 [16].

$$f(x,y) = \begin{bmatrix} f(0,0) & f(1,0) & \cdots & f(m-1,0) \\ f(0,1) & f(1,1) & \cdots & f(m-1,1) \\ \vdots & \vdots & \ddots & \vdots \\ f(0,n-1) & f(1,n-1) & \cdots & f(m-1,n-1) \end{bmatrix} \quad (1)$$

Cada pixel p^i representa um ponto no sistema de coordenadas de um dado espaço de cor. Desta forma um pixel $p^1(0,0,0)$ representaria um ponto nas coordenadas $(0,0,0)$ em um dado espaço de cor.

4.2 ESPAÇOS DE COR

Espaços de cor são definições matemáticas criadas para possibilitar a manipulação de imagens de maneira mais apropriada para cada necessidade. São fragmentos das frequências de cor enxergadas pelos seres humanos [16].

Neste trabalho são utilizados dois espaços de cor: o RGB (*Red*, *Green*, *Blue* - Vermelho, Verde, Azul) e o La*b*. O primeiro é o espaço de cor normalmente usado na representação, armazenamento e exibição de imagens em computadores, já o segundo é uma representação mais próxima da capacidade de percepção das cores feita pelo olho humano.

4.2.1 RGB

Os seres humanos percebem as cores através de três cores: vermelho (R-*red*), verde (G-*green*) e azul (B-*blue*). Estas cores são também consideradas as cores primárias, sendo possível através delas gerar todas as outras cores [17]. O espaço RGB tem como função mapear as cores de forma a garantir possibilidade da geração das mesmas, de forma que sejam visíveis para os seres humanos. Este espaço de cor é utilizado na exibição de imagens em monitores, televisões, projetores, entre outros equipamentos de projeção de imagens.

No espaço RGB, cada cor representa um ponto em um espaço tridimensional que varia de 0 até 255, ou de 0 até 1 em caso de um espaço normalizado. Desta forma o espaço RGB é representado por um cubo, cujos vértices são as cores primárias, vermelho, verde, azul, margenta, amarelo, ciano, branco e preto. Estes dois últimos são respectivamente a junção das três cores anteriores e a ausência destas [2].

A Figura 11 representa o espaço RGB e seus pontos extremos [2].

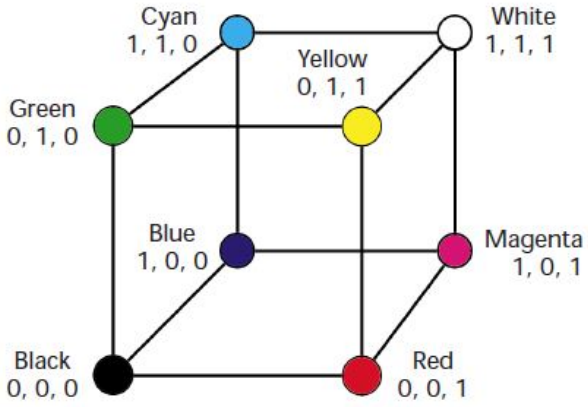


Figura 11: Representação gráfica do espaço RGB normalizado [2].

4.2.2 LA*B*

Apesar do espaço de cor RGB contemplar todas as cores que são possíveis de serem reproduzidas nos equipamentos de projeção de imagens, este espaço não é capaz de representar todas as cores visíveis aos seres humanos. Visando uma maior contemplação das cores visíveis, a CIE (*Commission Internationale de l'Eclairage* Comissão Internacional de Iluminação), desenvolveu o espaço de cor La*b* [18].

O espaço La*b* é um espaço tridimensional onde o L representa a luminância da imagem, o a* as coordenadas que variam do vermelho até o verde e o b* as coordenadas que variam do amarelo até o azul. Apesar do La*b* ser um espaço tridimensional, ele não é representado como um cubo, e sua representação pode ser encontrada de várias formas. Na Figura 12 são representado apenas as direções do espaço.

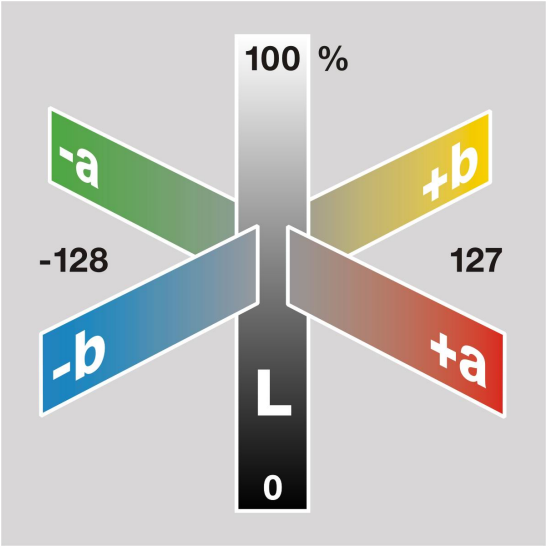


Figura 12: Direções do espaço La*b* [5]

É possível converter qualquer imagem que esteja no espaço RGB para o espaço La*b* e vice-versa. Esta conversão na teoria não implica em perda de informações, mas na prática, devido a arredondamentos, existem pequenas variações que podem ser consideradas imperceptíveis a olho nu.

O espaço La*b* é aproximadamente perceptualmente uniforme, ou seja, a distância entre as cores no espaço é aproximadamente proporcional à magnitude da diferença entre as cores percebidas por um ser humano. Por isso, a distância das cores no espaço La*b* será usada como uma medida de contraste, e o daltonismo será modelado como uma perda de contraste neste espaço [19] [18].

5 IMPLEMENTAÇÃO E RESULTADOS

Para este projeto, foram implementados diferentes módulos, em três fases principais:

- Fase 1 - trata-se do desenvolvimento de um simulador de daltonismo para deuteranopia. Este simulador foi de suma importância para o projeto como um todo, pois além de permitir uma melhor compreensão sobre o daltonismo, o simulador é usado como parte do algoritmo de processamento de imagens implementado neste projeto. O desenvolvimento deste simulador teve como base a descrição feita por Kuhn [6].
- Fase 2 - esta fase, consistiu na implementação do algoritmo descrito na dissertação de mestrado de Machado e Oliveira [1]. Esta implementação foi feita em linguagem Java.
- Fase 3 - nesta fase, foram criados um servidor de aplicação e uma extensão para o navegador Firefox, que permitem o acesso através da *Web* a uma versão otimizada do algoritmo implementado na Fase 2.

Para auxiliar no desenvolvimento dos algoritmos, foi utilizada a biblioteca OpenCV [20], uma biblioteca é de código aberto e contém operações básicas de processamento de imagens e visão computacional.

Nas próximas seções, as fases do projeto são descritas em detalhes, assim como os resultados obtidos.

5.1 FASE 1 - SIMULADOR DEUTERANOPIA

O simulador de deuteranopia foi desenvolvido com um duplo objetivo, o primeiro consiste no entendimento e na visualização das dificuldades encontradas pelos daltônicos ao navegar em páginas Web, o segundo é o fato da imagem simulada ser a base para o cálculo de rotação no espaço La*b*. Nas próximas subseções é descrita a implementação

do simulador de deuteranopia e feita a comparação dos resultados deste algoritmo com os apresentados no artigo de Kuhn [21].

5.1.1 IMPLEMENTAÇÃO

O simulador de deuteranopia consiste de três passos: conversão da imagem do espaço RGB para o espaço La*b*; projeção das cores da imagem em um plano no espaço La*b* e conversão da imagem novamente para o espaço RGB.

Para as conversões RGB $\Leftrightarrow$ La*b* foi utilizada uma função da biblioteca de processamento de imagens OpenCV. Esta função converte os pixels da imagem do espaço RGB, sendo $0 \leq R, G, B \leq 255$ para pixels no espaço La*b*, onde $0 \leq L \leq 100$, $-128 \leq a^*, b^* \leq 127$, e vice-versa [20].

Após a conversão da imagem de RGB para La*b*, é feita a projeção das cores sobre um plano s. Como mostrado na Figura 13, este plano é alinhado ao eixo L, forma um ângulo θ com o eixo b*. Desta forma, a luminância das cores é preservada durante o processamento. A projeção de uma determinada cor é feita ignorando a coordenada L e modificando apenas as coordenadas a* e b*. Ou seja, a projeção da cor sobre o plano é equivalente à projeção de um ponto no espaço a* b* sobre uma reta r com ângulo θ que passa pela origem, assim como mostra a Figura 14.

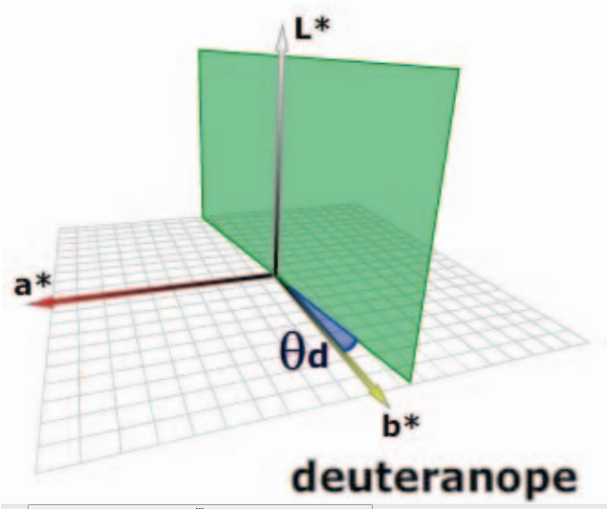


Figura 13: Representação do plano s no espaço La*b* [6].

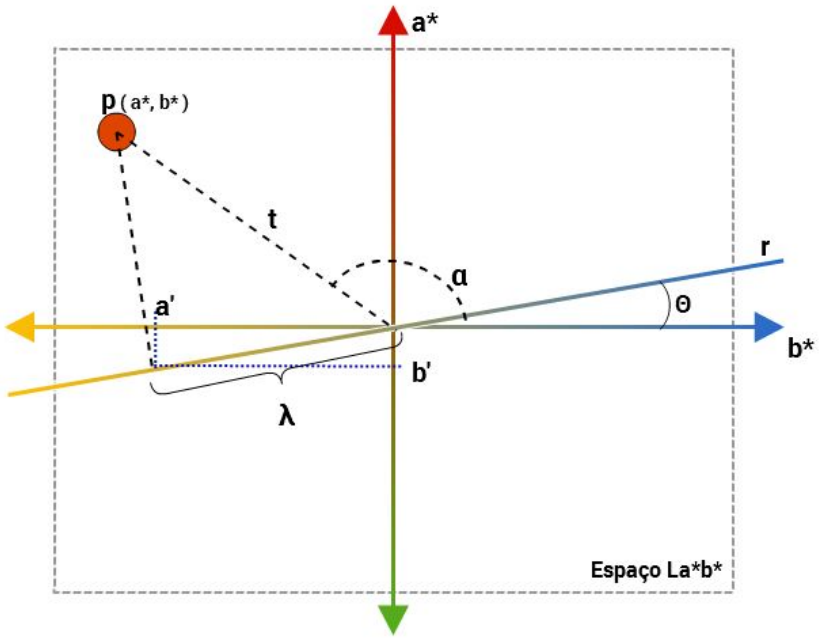


Figura 14: Projeção do pixel p sobre a reta r no espaço La*b*.

Para realizar a simulação de deuteranopia sobre a imagem, para cada pixel p no espaço La*b* é feita a sua projeção sobre a reta r. Para isso, para cada pixel $p_{(a^*,b^*)}$ são obtidos suas coordenadas polares $p_{(t,\alpha)}$, através das Equações 2 e 3, respectivamente.

$$t = \sqrt{a^{*2} + b^{*2}} \tag{2}$$

$$\alpha = arctan\left(\frac{a^*}{b^*}\right) \tag{3}$$

A partir das coordenadas polares, é calculado para cada pixel o valor λ, que representa a distância do pixel projetado sobre a reta r até a origem. Com esta informação é possível encontrar os valores de a' e b', que representam as novas coordenadas do pixel no espaço La*b*, como demonstram as Equações 4 e 5.

$$\lambda = t \cdot \cos\left(\left|\left(-8.11^\circ \cdot \frac{\pi}{180}\right) - \alpha\right|\right) \tag{4}$$

$$\begin{aligned} a' &= \lambda \cdot \sin\left(-8.11^\circ \cdot \tfrac{\pi}{180}\right) \\ b' &= \lambda \cdot \cos\left(-8.11^\circ \cdot \tfrac{\pi}{180}\right) \end{aligned} \tag{5}$$

Segundo Kuhn [21], o ângulo θ que permite a simulação da deuteranopia é -8.11°.

Apesar do foco do presente trabalho ser apenas um tipo de daltonismo (deuteranopia), este simulador é capaz de simular outros tipos de dicromatismo. Para isto, basta alterar o ângulo θ do plano s para um ângulo que corresponda ao tipo de daltonismo desejado.

O código do simulador em Java encontra-se no APÊNDICE A.

5.1.2 RESULTADOS DO SIMULADOR

A validação do simulador consistiu na comparação visual entre todos os resultados obtidos da sua execução com aqueles apresentados por Kuhn [6]. A Figura 15 mostra algumas comparações entre os resultados apresentados por Kuhn e os obtidos neste trabalho. Sendo as imagens (a) as originais, (b) as respectivas imagens apresentadas por Kuhn [6] e (c) as respectivas imagens processadas com o simulador implementado neste trabalho. É possível observar uma leve diferença na tonalidade das cores. Contudo esta diferença pode existir devido à origem das imagens usadas e a fatores de compressão.

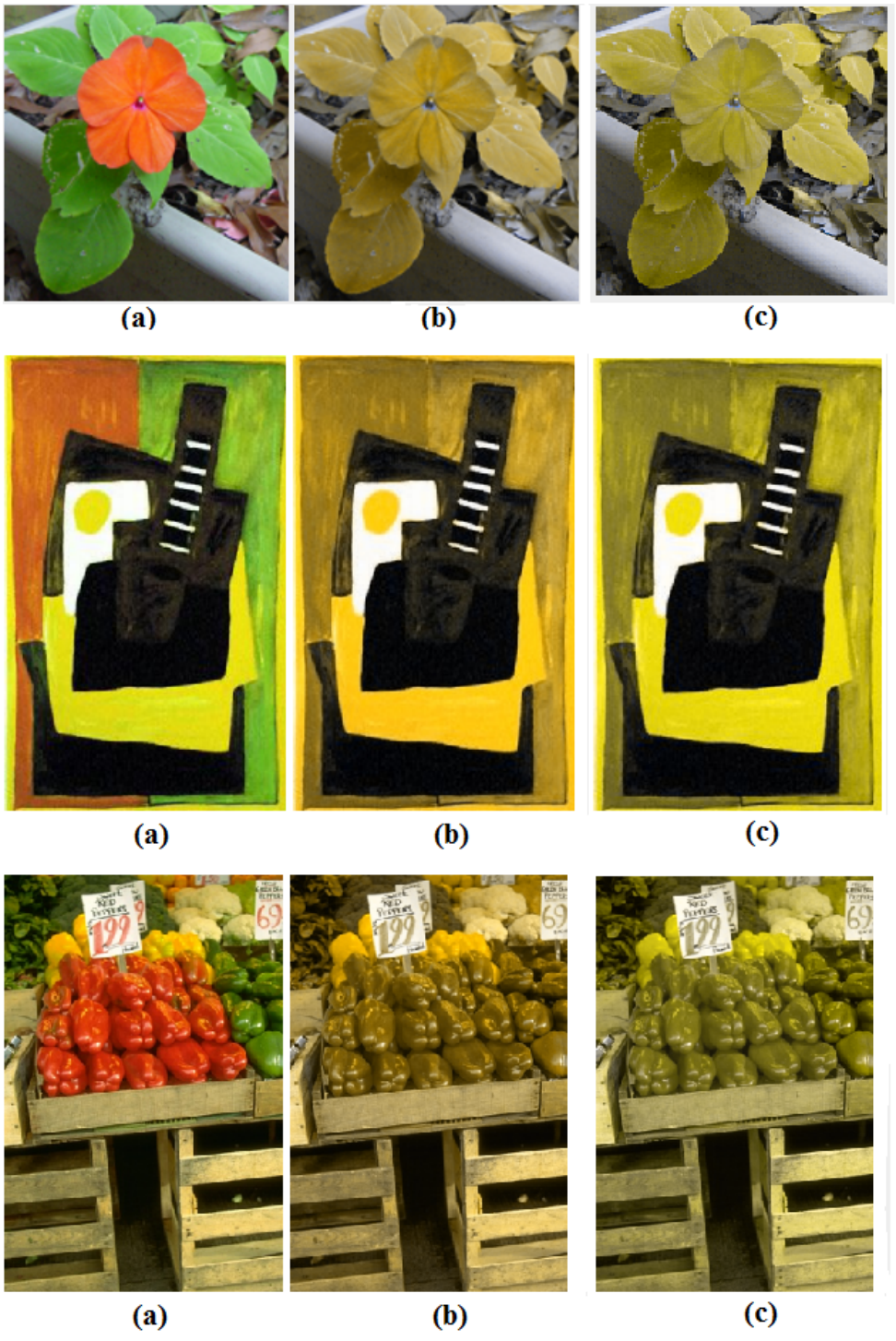


Figura 15: Comparação entre os resultados do simulador com os resultados apresentados por Kuhn [6].

Outras imagens interessantes de serem observadas são os testes de Ishihara, uma vez que eles são usados para o diagnóstico do daltonismo. A Figura 16 apresenta alguns destes testes, enquanto a Figura 17 apresenta a simulação destes testes para deuteranopia. É possível observar que a Figura 1, apesar de ter suas cores alteradas, tem seu conteúdo

claro, isso é devido ao fato do conteúdo ser invisível para outros tipos de daltonismo, como a protanopia e a tritanopia.

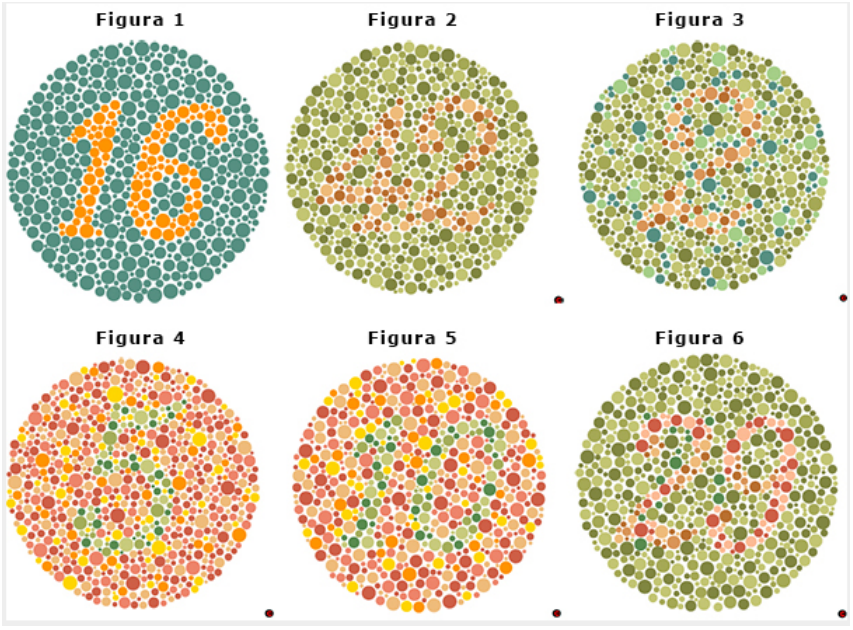


Figura 16: Testes de Ishihara [7].

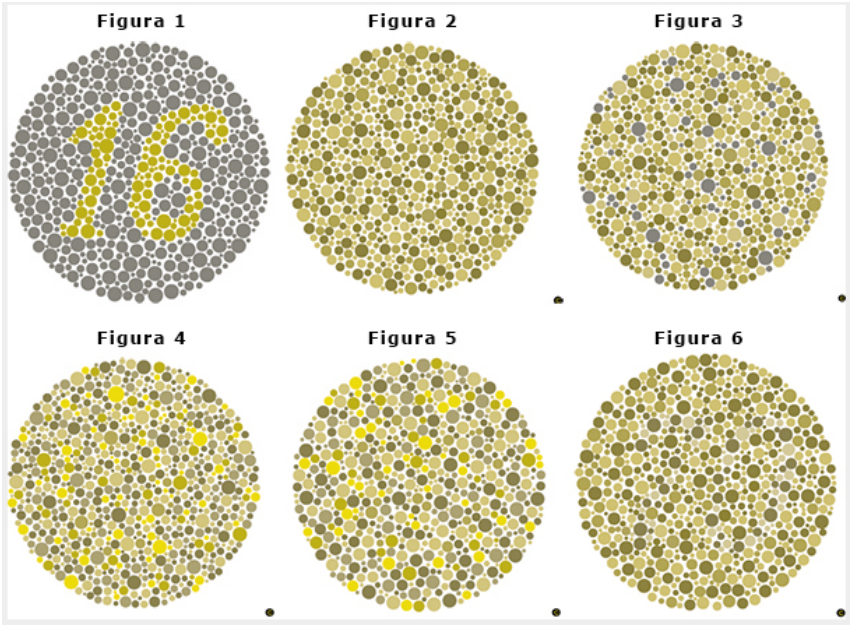


Figura 17: Simulação do teste de Ishihara com o algoritmo desenvolvido neste trabaho Fonte: Autoria própria.

5.2 FASE 2 - APLICAÇÃO DESKTOP

A segunda fase deste projeto consistiu no desenvolvimento de uma aplicação para *desktop* que implementa o algoritmo descrito na dissertação de Machado e Oliveira [1]. Esta aplicação foi desenvolvida em linguagem Java e reutiliza a implementação do simulador, descrito na Seção 5.1. Nas próximas subseções, é descrita a implementação da aplicação e feita a comparação dos resultados deste algoritmo com os apresentados por Machado e Oliveira [1].

5.2.1 IMPLEMENTAÇÃO

O primeiro passo da aplicação consiste em, a partir de uma imagem de entrada O_{mXn}, gerar uma segunda imagem S_{mXn}. Esta é obtida a partir do simulador de deuteranopia descrito na Seção 5.1.

Em seguida, para cada pixel $p^1 \in O_{mXn}$ é escolhido aleatoriamente um pixel vizinho $p^2 \in O_{mXn}$, e os respectivos pixels p'^1 e $p'^2 \in S_{mXn}$. A escolha aleatória segue uma distribuição Gaussiana, com média 0 e variância $\left(\frac{2}{\pi} * \sqrt{2 * MIN(m,n)}\right)$, sendo $MIN(m,n)$ uma função que retorna o menor valor entre a altura n e a largura m da imagem. A utilização dessa distribuição garante uma probabilidade maior do pixel vizinho estar localizado em um posição próxima na imagem.

A diferença entre dois pixels no espaço La*b* é dada pela distância Euclidiana entre estes pixels. Desta forma, são calculadas as distâncias D_1 e D_2 entre os pixels p^1 e p^2 e entre os pixels p'^1 e p'^2. Este cálculo é feito levando em conta apenas as coordenadas a* e b* dos pixels, o que resulta nas Formulas 6 e 7.

$$D_1 = \sqrt{(p^1_{a^*} - p^2_{a^*})^2 + (p^1_{b^*} - p^2_{b^*})^2} \tag{6}$$

$$D_2 = \sqrt{(p'^1_{a^*} - p'^2_{a^*})^2 + (p'^1_{b^*} - p'^2_{b^*})^2} \tag{7}$$

Após obter D_1 e D_2, é determinada a medida da perda de contraste d, dada pela Equação 8.

$$d = \frac{D_1 - D_2}{D_1} \tag{8}$$

É criada então uma matriz M_{kX2} de perda de contraste, contendo todos os vetores de perda de contraste para as coordenadas a* e b* da imagem, sendo $k = m * n$ (m e n são respectivamente as dimensões da imagem O_{mXn}). O vetor W para um pixel é computado multiplicando-se d pela direção da perda de contraste, como mostra a Equação 9:

$$(W^{a^*}, W^{b^*}) = d \cdot (p^1_{a^*} - p^2_{a^*}, p^1_{b^*} - p^2_{b^*}) \tag{9}$$

O próximo passo do algoritmo se dá por encontrar o vetor $\vec{v}$ no plano a*b*, que representa a menor perda de contraste. Para isso é gerada uma matriz Hessiana M'_{2X2}, que é obtida através da multiplicação da matriz M_{kX2} pela sua transposta M^T_{2Xk}, em seguida é calculado o vetor $\vec{v}$ com o maior autovalor.

Sendo Θ o ângulo do vetor $\vec{v}$, todas as cores da imagem são rotacionadas nos eixos a* e b*. Desta forma, cada pixel da imagem final no espaço La*b* terá o valor de L da imagem original e novos valores para a* e b*, gerando uma imagem que permita uma melhor percepção dos daltônicos.

O código da aplicação em Java encontra-se na APÊNDICE B.

5.2.2 RESULTADOS DA APLICAÇÃO

Para a validação dos resultados, foi feita uma comparação visual entre os resultados obtidos pela aplicação e todos os resultados apresentados por Machado e Oliveira [1]. Em seguida, foi feito o processamento nos testes de Ishihara, uma vez que estes testes são considerados os mais eficazes para o diagnostico do daltonismo.

A Figura 18 mostra as imagens originais (a), as imagens de resultado apresentadas por Machado e Oliveira [1] (b) e as imagens obtidas pela aplicação (c). É possível observar uma pequena diferença entre as imagens obtidas pela aplicação e as imagens apresentadas por Machado e Oliveira, mas esta diferença se dá pela forma como as imagens foram obtidas e pelo tipo de compressão da imagem.

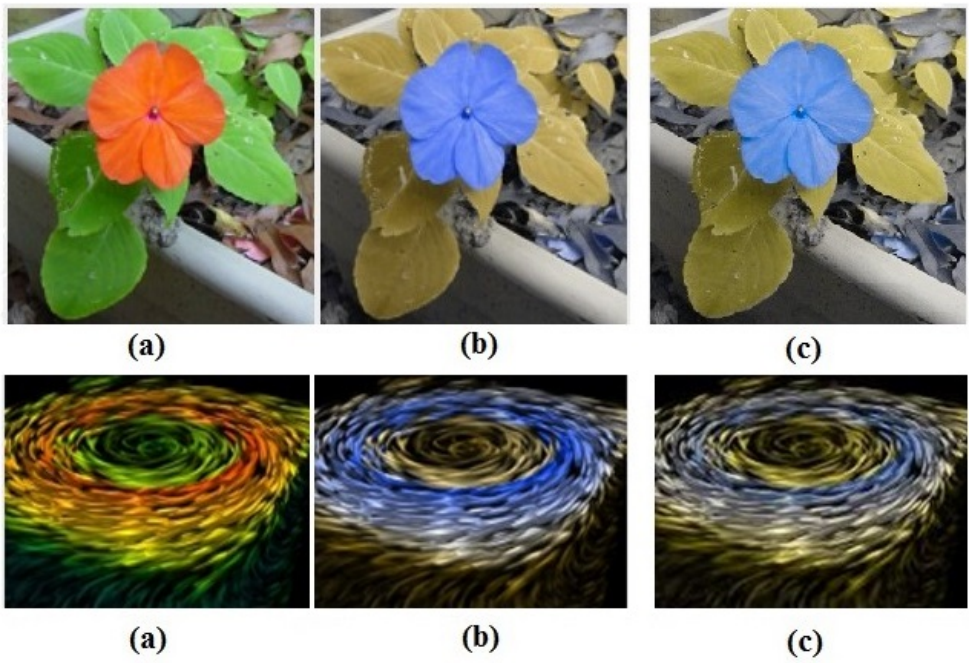


Figura 18: Comparação entre os resultados obtidos pela aplicação e os resultados apresentados por Machado e Oliveira [1].

É importante salientar que as imagens finais apresentadas por Machado e Oliveira são equivalentes às imagens percebidas por daltônicos. Ou seja, após o processamento da imagem, esta passou novamente pelo simulador. Este fato pode ser observado na Figura 19, onde a imagem (a) é a imagem original, a imagem (b) é a imagem após o processamento do algoritmo desenvolvido neste trabalho e a imagem (c) é a imagem (b) do ponto de vista de um daltônico. Assim, uma pessoa com deuteranopia vê as imagens (b) e (c) de forma semelhante.

Figura 19: Simulação da imagem após processamento.

As Figuras 20, 21, 22 e 23 demonstram a validação do algoritmo nos testes de Ishihara, sendo a Figura 21 a simulação da Figura 20, a Figura 22 o resultado do algoritmo

e a Figura 23 a simulação da Figura 22. Um daltônico com deuteranopia percebe as imagens da Figura 20 semelhante às imagens da Figura 21 e as imagens da Figura 22 semelhante as imagens da Figura 23.

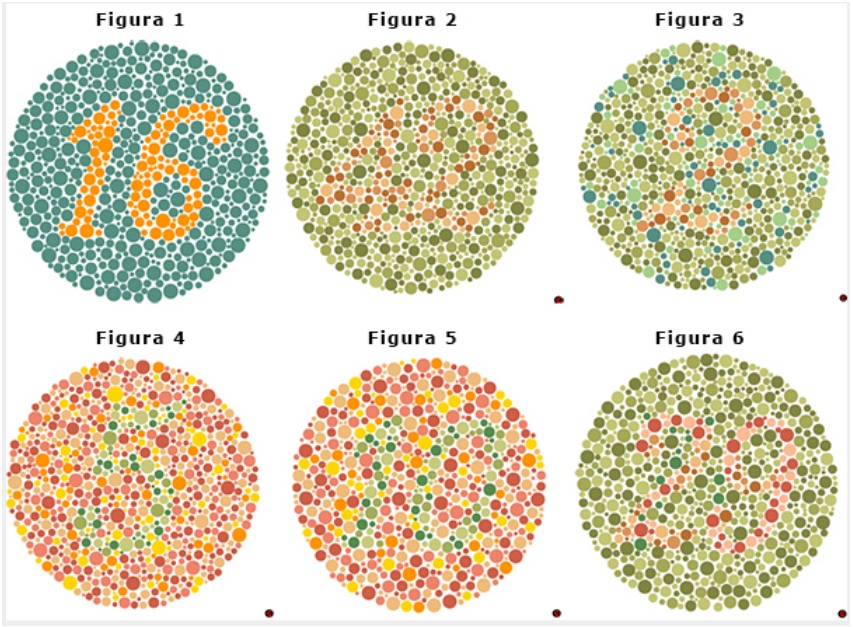


Figura 20: Testes de Ishihara [7].

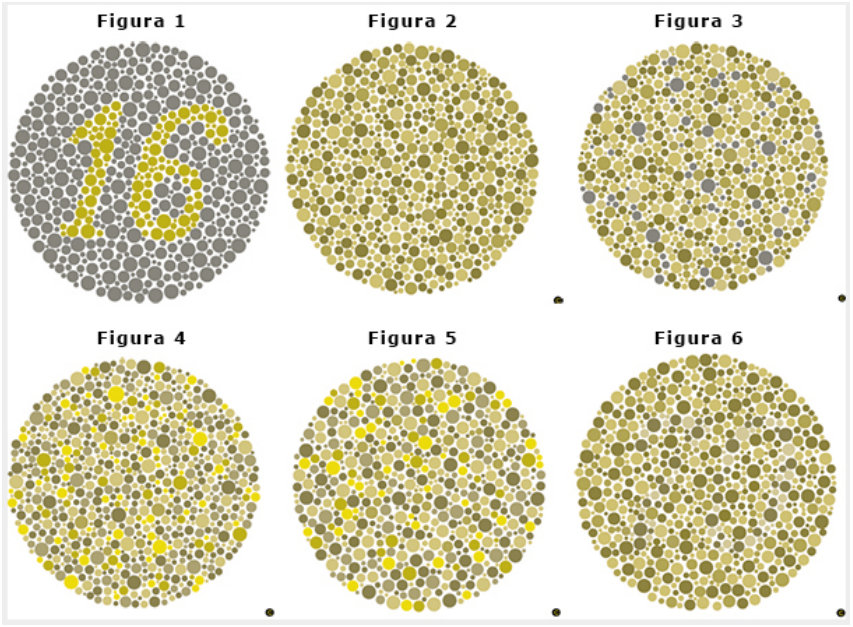


Figura 21: Simulação do teste de Ishihara.

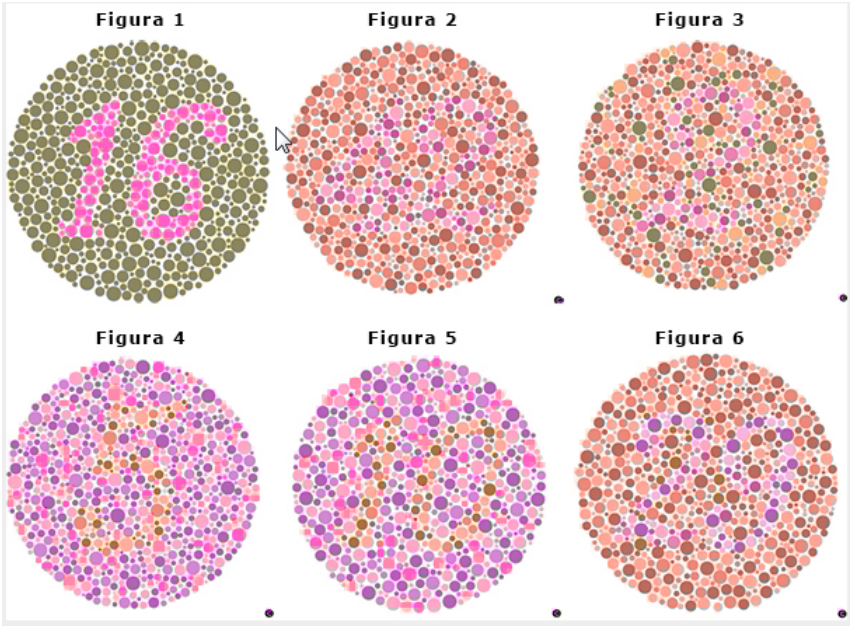


Figura 22: Testes de Ishihara após o processamento.

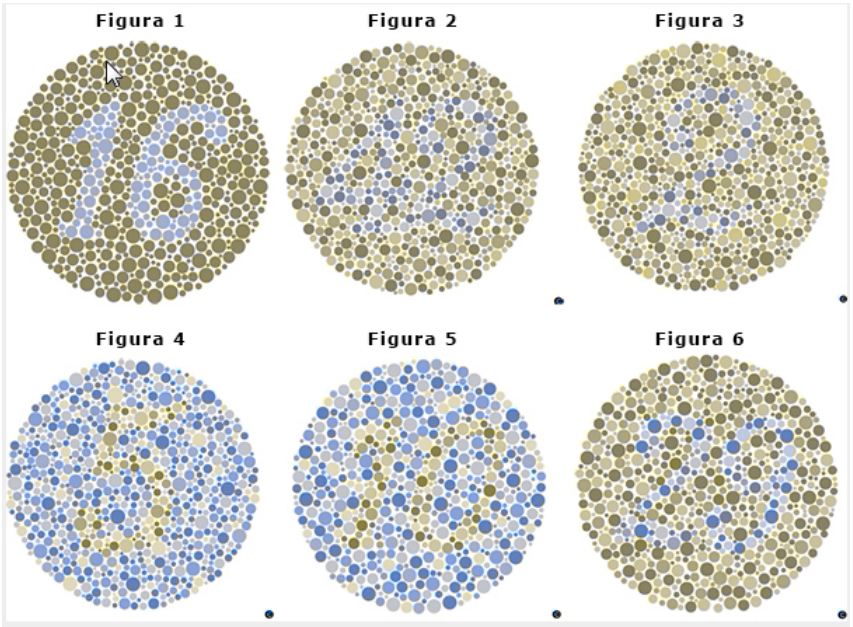


Figura 23: Simulação do teste de Ishihara após o processamento.

Como o algoritmo percorre as imagens de entrada pixel a pixel, o tempo de execução depende diretamente da quantidade de pixels existentes na imagem. Ou seja, quanto maior a imagem, maior será o tempo de execução. A Figura 24 mostra os tempos, em minuto, para o processamento de 8 imagens de diferentes tamanhos. É possível observar que a maior imagem, com 1632 de largura e 1224 de altura (1.997.568 pixels), levou um tempo de 272,13 minutos, o que equivale a aproximadamente 5 horas.

Estes testes foram executados em um notebook com processador Intel Core I5, 8Gb de memória RAM e placa de vídeo ATI Radeon HD4650 1gb GDDR3.

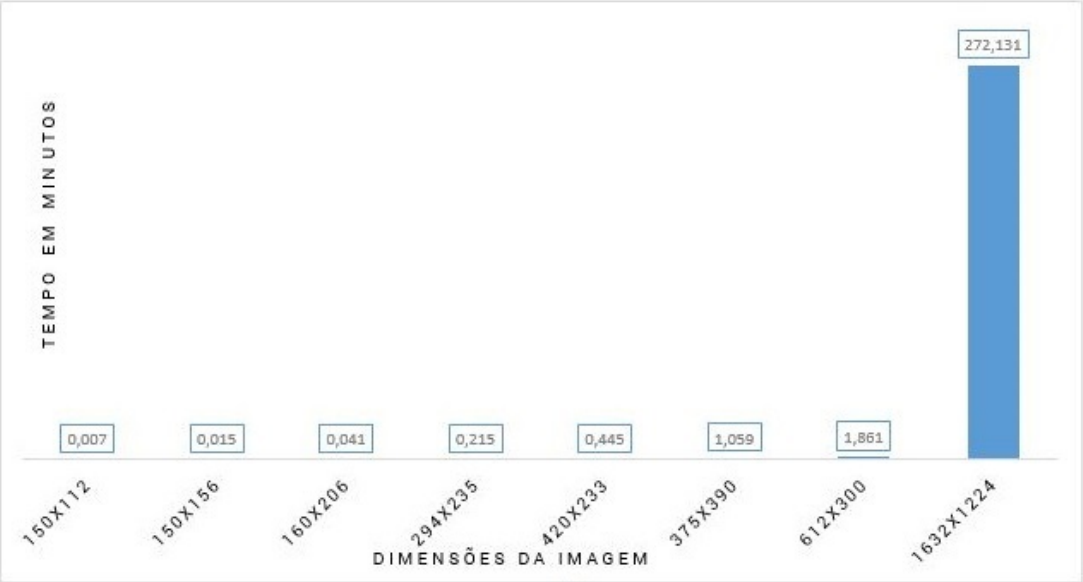


Figura 24: Relação do tamanho das imagens com o tempo de processamento.

5.3 FASE 3 - APLICAÇÃO *WEB* - ACTIOCOLOR

Um dos objetivos deste trabalho consiste em criar uma aplicação *Web* que permita aos daltônicos uma melhor compreensão das imagens. Devido a limitações e restrições de segurança dos navegadores, optou-se por desenvolver uma aplicação *Web* em duas partes. A primeira consiste em um servidor REST (*Representational State Transfer* - Transferência de Estado Representacional) que irá receber uma requisição no formato Json (*JavaScript Object Notation* - Notação de Objeto JavaScript), e retornar a imagem processada. A segunda parte trata-se de uma extensão para o navegador Firefox que captura a imagem selecionada pelo usuário e a envia ao servidor, uma vez que o servidor responda, a extensão irá substituir a imagem na página *Web*.

Nas próximas seções, serão descritos a implementação e as otimizações feitas no servidor, Assim como a implementação da extensão para o navegador Firefox. Em seguida, os resultados obtidos são e apresentados.

5.3.1 IMPLEMENTAÇÃO E OTIMIZAÇÃO DO SERVIDOR *WEB*

O servidor consiste em uma aplicação Java hospedada em um servidor GlassFish [22]. Esta aplicação se comunica através de requisições REST em formato Json. A apli-

cação espera que a requisição contenha o nome da imagem, o formato de compressão, e a imagem codificada em *Base64*, (forma de descrever arquivos binários como texto).

O primeiro passo do servidor ao receber a requisição consiste em decodificar a imagem em formato *Base64* para uma imagem. Após essa decodificação, a imagem é reduzida por um fator. Este fator é selecionado de acordo com a quantidade de pixels da imagem original. A Tabela 2 mostra os fatores de redução utilizados no algoritmo. Estes fatores foram determinados de forma empírica.

Tabela 2: Fatores de divisão para a redução da imagem.

Número de pixels na imagem (n)	**Fator de redução** d	**Exemplo de imagens**	
		Dimensões	**Quantidade de pixels**
n $\leq$ 786.432	4	1024x768	786.432
n $\leq$ 5.038.848	6	2048x1536	3.145.728
n $\leq$ 9.291.264	10	3000x2048	6.144.000
n $\leq$ 13.543.680	12	3240x3240	10.497.600
n $>$ 13.543.680	1.000	4320x3240	13.996.800

A Figura 25 mostra a imagem original $O_{700x400}$ e a imagem reduzida $R_{175x100}$ com um fator de redução $d = 4$.

Figura 25: Resultado da redução de uma imagem.

O algoritmo descrito na Seção 5.2 é aplicado à imagem reduzida. Uma otimização feita no algoritmo foi a criação de uma função *hash* que retorna valores aproximados de seno e cosseno pré-computados. Essa função foi criada pois a utilização de uma função *hash* se torna mais rápida do que o uso da biblioteca *Math* do Java. Esta função consiste

em dois vetores com 360 valores aproximados de senos e cossenos em radianos. Desta forma, dado um ângulo qualquer α em radianos, basta dividi-lo por 0,017453292 para obter a posição nos vetores, que correspondem aos valores aproximados de seno e cosseno do ângulo α.

Uma vez a imagem processada, esta será redimensionada através de uma interpolação bilinear. Para cada pixel $p_{(L,a^*,b^*)} \in F$, sendo F a imagem final, é calculada uma média ponderada dos quatro pixels $\in R$, no plano a* e b*, mais próximos da posição do pixel $p_{\left(\frac{a^*}{d},\frac{b*}{d}\right)}$ em R. A interpolação é feita apenas nos eixos a* e b*, sendo que o eixo L é obtido da imagem original. Ao processar uma versão reduzida da imagem, é obtido um custo computacional reduzido, mas gerando perdas de detalhes. Usando o canal L da imagem original, essas perdas são reduzidas consideravelmente.

Em seguida, a imagem ampliada é codificada para Base64 e colocada em um Json, por fim a mensagem é empacotada com o protocolo de segurança *CROSS-Domain* e enviada ao cliente que efetuou a requisição. O empacotamento da mensagem no protocolo é essencial, uma vez que o *CROSS-Domain* é definido pela W3C (*World Wide Web Consortium*) e implementado nos navegadores *Web* visando garantir a segurança dos usuários [23].

5.3.2 IMPLEMENTAÇÃO DA EXTENSÃO

A extensão foi desenvolvida em JavaScript para o navegador Firefox, e consiste em uma interface simples que faz a requisição do processamento de uma imagem selecionada pelo usuário. Esta interface foi criada utilizando a linguagem de configuração do Firefox XUL (Linguagem de interface do usuário XML). A Figura 26 mostra o menu de contexto ao clicar em uma imagem.

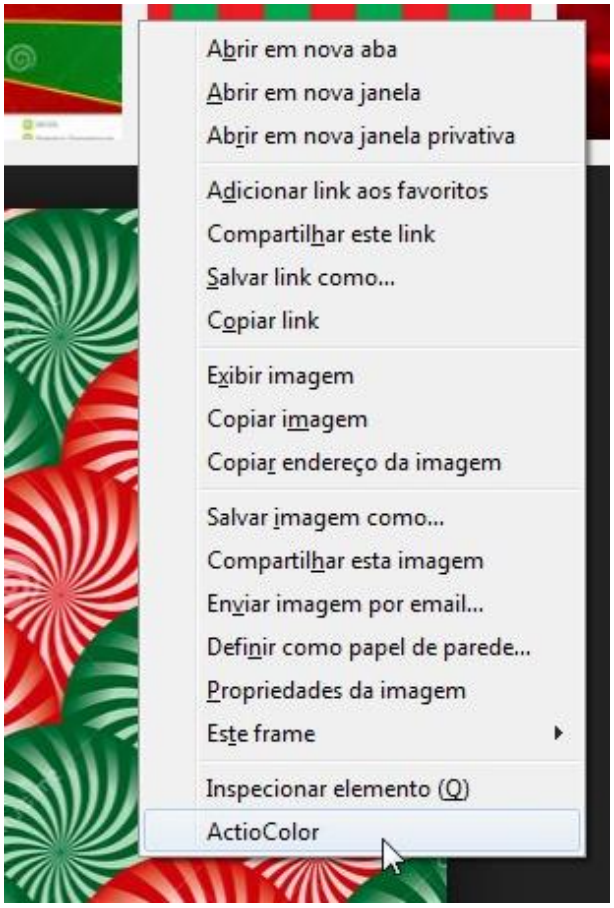


Figura 26: Menu de contexto em uma imagem após a instalação da extensão.

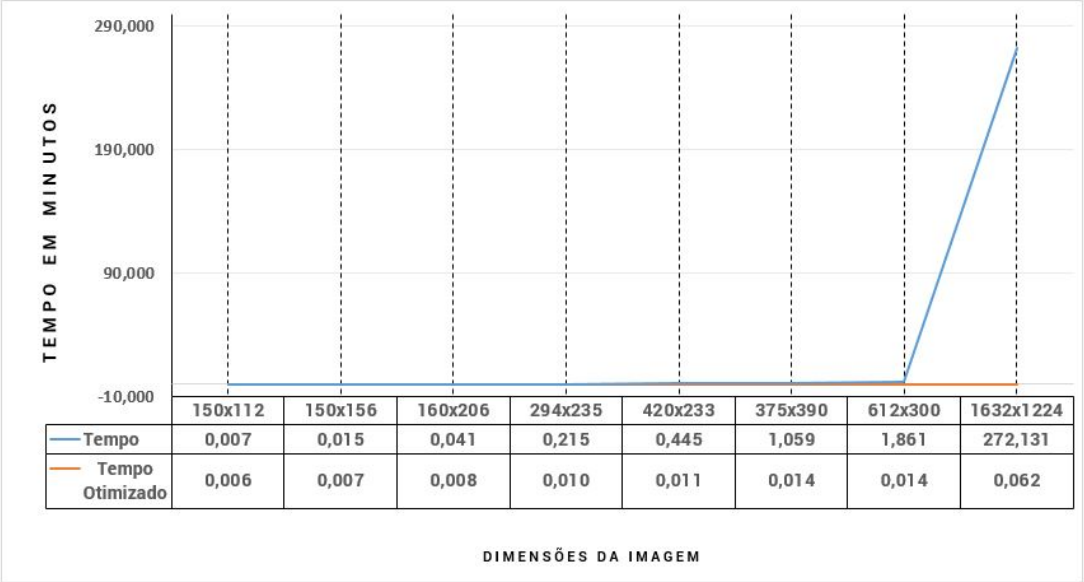
Após a extensão ser instalada no navegador, através da biblioteca de aplicações do Firefox ou diretamente, é inserida uma opção no menu de contexto (menu do botão direito do *mouse*). Esta opção só está disponível quando o usuário clicar com o botão direito do *mouse* em uma imagem.

A extensão codifica a imagem que o usuário selecionou em *Base64*, e codifica uma *string* Json, com o nome e formato da imagem e a codificação da imagem, e envia ao servidor a *string* requisitando o processamento da imagem. A extensão aguarda a resposta do servidor sem travar o uso do navegador, desta forma o usuário continua utilizando e visualizando a página normalmente.

5.3.3 RESULTADOS

A implementação das otimizações descritas na Subseção 5.3.1 resultou em uma melhora significativa no tempo de processamento das imagens. A Figura 27 mostra os gráficos do tempo em minutos necessário para o processamento de 8 imagens aleatórias.

É possível observar que a imagem com a maior dimensão (1632x1224) necessitou de um tempo maior de processamento para ambos os algoritmos, contudo o tempo no algoritmo otimizado é 272,069 minutos menor do que o necessário no algoritmo não otimizado, resultando em uma diminuição de 99,98% do tempo de processamento. Este resultado se dá ao fato da imagem ter sido reduzida com um fator de redução $d = 6$, o que resultou no processamento de uma imagem de dimensões 272x204.



	150x112	150x156	160x206	294x235	420x233	375x390	612x300	1632x1224
Tempo	0,007	0,015	0,041	0,215	0,445	1,059	1,861	272,131
Tempo Otimizado	0,006	0,007	0,008	0,010	0,011	0,014	0,014	0,062

Figura 27: Com paração do tempo de processamento de 8 imagens pelo algoritmo otimizado e pelo algoritmo não otimizado.

Apesar da otimização ter resultado em um tempo de processamento menor, ela também resultou em uma perda de qualidade para determinadas imagens. Nota-se que imagens contendo cores sólidas e mudanças bruscas de contraste, como desenhos, como mostrado na Figura 28, onde a imagem (a) é a imagem original e a figura (b) é a imagem após a ampliação. Contudo, este efeito não é percebido em imagens onde as mudanças de contraste são mais suaves, como fotografias.

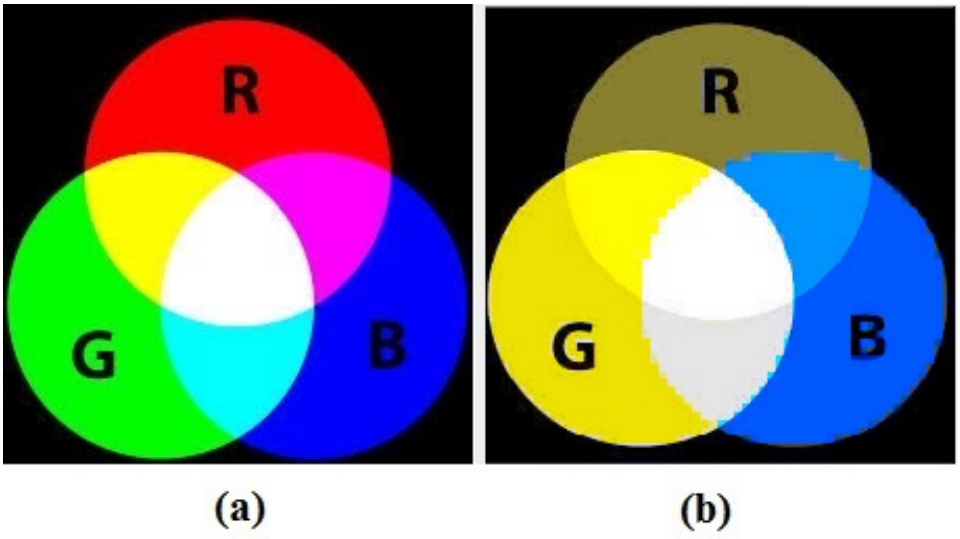


Figura 28: Exemplo de perda de qualidade devido a otimização Fonte: Autoria própria.

Para a validação objetiva desta otimização, foi feito o processamento de 120 imagens pelos dois algoritmos (otimizado e não otimizado), em seguida é calculado a soma das distancias dos pixels das imagens resultantes dos processamentos, gerando uma média de 2,7, um desvio padrão de 2,45 e um máximo de 18,68. Sendo a maior distancia possível da imagem processada pelo algoritmo não otimizado, para a processada pelo algoritmo otimizado o tamanho do espaço La*b* utilizado (-127 até 127).

A Figura 29 mostra o resultado do processamento em páginas *Web*, sendo as imagens (a) as páginas sem nenhuma alteração, as imagens (b) as páginas vista por um daltônico com deuteranopia e as imagens (c) as páginas vistas por um daltônico e com figuras alteradas pela extensão.

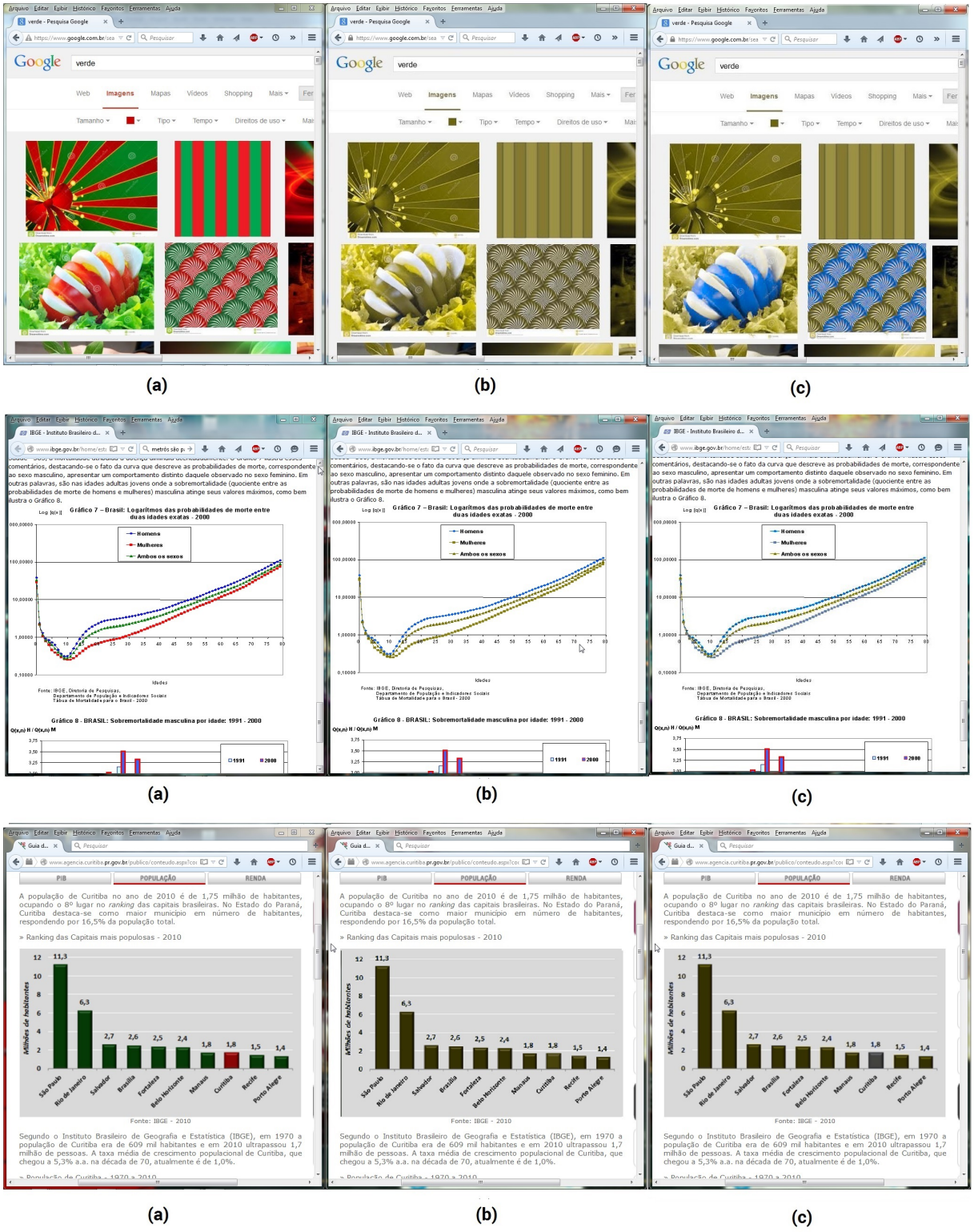


Figura 29: As imagens (a) mostram o navegador Firefox com as páginas *Web* originais, as imagens (b) ilustram essas páginas vistas por daltônicos com deuteranopia e as imagens (c) exemplificam os resultados da extensão nestas páginas Fonte: Autoria própria.

6 CONCLUSÃO

Neste trabalho de conclusão de curso, foi apresentado uma aplicação *Web* que auxilia na compreensão de imagens em páginas *Web* para daltônicos.

Com base nos resultados apresentados neste trabalho, comprovamos a eficácia do algoritmo de simulação de deuteranopia apresentado por Kuhn [21] e do algoritmo de recoloração de imagens apresentado por Machado e Oliveira [1]. Comprovamos também a possibilidade de reutilização dos algoritmos para auxiliar daltônicos na navegação de páginas *Web*.

A partir dos algoritmos apresentados por Kuhn, Machado e Oliveira, foi possível desenvolver uma aplicação *Web* (ActioColor), com o objetivo de permitir a utilização destas técnicas de processamento de imagens em páginas da *Web*.

As contribuições deixadas por este trabalho, consistem na otimização do algoritmo e na criação de uma aplicação *Web*, composta por um servidor e uma extensão para o Firefox, ambas permitem uma melhor percepção das imagens nas páginas *Web*.

Apesar de existir melhorias a serem feitas na extensão, o uso da mesma poderá permitir um nível de compreensão melhor para pessoas daltônicas, possibilitando uma maior acessibilidade aos conteúdos expostos na Internet.

6.1 TRABALHOS FUTUROS

Apesar da aplicação apresentada neste trabalho ser funcional, esta é apenas um protótipo. Desta forma algumas melhorias podem ser realizadas para aumentar a qualidade e a velocidade da aplicação.

- A primeira delas seria a configuração de um servidor global, permitindo assim o acesso da extensão a partir de qualquer computador.
- Otimizar o servidor de aplicação, com a utilização de *threads* ou processamento em

GPU.

- Visando auxiliar outros tipos de daltonismo como tritanopia e protanopia, se fazem necessárias pequenas alterações no servidor e na extensão.
- Melhorar a acessibilidade da extensão, permitindo ao usuário desfazer o processamento de uma determinada imagem.

APÊNDICE A – CÓDIGO FONTE - SIMULADOR

```
public class SimuladorDeuteranopia {

    private static final float anguloDeuterotopia = (float) ((Math.PI /
    180) * -8.11);

    public static void main(String[] args) {
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
        Mat imagem8U = Highgui.imread("C:\imagem1.png");

        imagemTela(convertMatToBuffer(imagem8U), "ORIGINAL");

        Mat imagem = new Mat(imagem8U.size(), CvType.CV_32FC3);
        imagem8U.convertTo(imagem, CvType.CV_32F, 1.0 / 255.0);

        Mat labImagem = new Mat(imagem.size(), CvType.CV_32FC3);
        Imgproc.cvtColor(imagem, labImagem, Imgproc.COLOR_BGR2Lab);

        Mat deutImagem = new Mat(imagem.size(), CvType.CV_8UC3);
        Imgproc.cvtColor(projetorDeuteranotopia(labImagem), imagem, Imgproc
    .COLOR_Lab2BGR);
        imagem.convertTo(deutImagem, CvType.CV_8U, 255);
    }

    private static Mat projetorDeuteranotopia(Mat imagem) {
        int imgH = (int) imagem.size().height;
        int imgW = (int) imagem.size().width;
        Mat imagDeut = new Mat(imagem.size(), CvType.CV_32FC3);
        double[] pixel = new double[3];
        int L = 0, a = 1, b = 2;
        float alpha = 0, t = 0, lambda = 0;

        for (int i = 0; i < imgH; i++) {
            for (int j = 0; j < imgW; j++) {
```

```
33                  pixel = imagem.get(i, j);
                    alpha = (float) Math.atan2(pixel[a], pixel[b]);
35                  t = (float) Math.sqrt(Math.pow(pixel[a], 2) + Math.pow(
    pixel[b], 2));

37                  lambda = (float) (t * Math.cos(Math.abs(anguloDeuterotopia
    - alpha)));

39                  pixel[a] = lambda * Math.sin(anguloDeuterotopia);
                    pixel[b] = lambda * Math.cos(anguloDeuterotopia);
41
                    imagDeut.put(i, j, pixel);
43              }
            }
45          return imagDeut;
        }
47
}
```

Simulador.java

APÊNDICE B – CÓDIGO FONTE - APLICAÇÃO

```
class Processamento {

    private static final Matematica mate = new Matematica();

    private static final float anguloDeuterotopia = (float) ((Math.PI /
    180) * -8.11);
    private static final int L = 0;
    private static final int A = 1;
    private static final int B = 2;
    private static int largura;
    private static int altura;

    private static float[][] matrizPerdaContraste;
    private static int[] posicaoGaussiana;
    private static int contOperacao;
    private static float anguloRotacao;

    private static final float[] pixelOriginal = new float[3];
    private static final float[] pixelVizinho = new float[3];
    private static final float[] pixelOriginalDet = new float[3];
    private static final float[] pixelVizinhoDet = new float[3];

    private static float distOriginalVizinho;
    private static float distOriginalVizinhoDet;
    private static float direcaoPerdaContraste;
    private static float[][] matrizFinal;
    private static float[][] autovetor;
    private static float[] autovalor;
    private static float[] newPixel;
    private static float[] oldPixel;
    private static float senoAngRotacao;
    private static float cossenoAngRotacao;

    public Mat processamento(Mat labImagem_32FC3) {
```

```
34
        Mat labImagem_32FC3_Simulado = projetorDeuteranotopia(
    labImagem_32FC3);
36
        altura = (int) labImagem_32FC3.size().height;
38      largura = (int) labImagem_32FC3.size().width;

40      matrizPerdaContraste = new float[altura * largura][2];
        contOperacao = 0;
42
        for (int ROW = 0; ROW < altura; ROW++) {
44          for (int COL = 0; COL < largura; COL++) {

46              labImagem_32FC3.get(ROW, COL, pixelOriginal);
                posicaoGaussiana = mate.calcDistriGaussi(COL, ROW, largura,
     altura);
48
                labImagem_32FC3.get(posicaoGaussiana[1], posicaoGaussiana
    [0], pixelVizinho);
50
                labImagem_32FC3_Simulado.get(ROW, COL, pixelOriginalDet);
52              labImagem_32FC3_Simulado.get(posicaoGaussiana[1],
    posicaoGaussiana[0], pixelVizinhoDet);

54              distOriginalVizinho = (float) (Math.sqrt((pixelOriginal[A]
    - pixelVizinho[A]) * (pixelOriginal[A] - pixelVizinho[A])
                        + (pixelOriginal[B] - pixelVizinho[B]) * (
    pixelOriginal[B] - pixelVizinho[B])));
56
                distOriginalVizinhoDet = (float) (Math.sqrt((
    pixelOriginalDet[A] - pixelVizinhoDet[A]) * (pixelOriginalDet[A] -
    pixelVizinhoDet[A])
58                      + (pixelOriginalDet[B] - pixelVizinhoDet[B]) * (
    pixelOriginalDet[B] - pixelVizinhoDet[B])));

60              if (distOriginalVizinho == 0) {
                    direcaoPerdaContraste = 0;
62              } else {
                    direcaoPerdaContraste = ((distOriginalVizinho -
    distOriginalVizinhoDet) / distOriginalVizinho);
64              }
```

```
66                  matrizPerdaContraste[contOperacao][0] = (pixelOriginal[A] -
     pixelVizinho[A]) * direcaoPerdaContraste;
                    matrizPerdaContraste[contOperacao][1] = (pixelOriginal[B] -
     pixelVizinho[B]) * direcaoPerdaContraste;
68                  contOperacao++;

70              }
            }
72
            matrizFinal = mate.multiplicaMatriz(matrizPerdaContraste, mate.
    matrizTransposta(matrizPerdaContraste));
74          autovalor = mate.autovalor(matrizFinal);
            autovetor = mate.autovetor(matrizFinal, autovalor);
76
            if (autovetor[0][0] > autovetor[1][0]) {
78              anguloRotacao = (float) Math.atan2(autovetor[0][0], autovetor
    [0][1]);
            } else {
80              anguloRotacao = (float) Math.atan2(autovetor[1][0], autovetor
    [1][1]);
            }
82
            return rotacionaImagem(labImagem_32FC3, anguloRotacao);
84      }

86      private static Mat rotacionaImagem(Mat imagemLab, float angRotacao) {

88          Mat imagemFinal = new Mat(imagemLab.size(), CvType.CV_32FC3);

90          senoAngRotacao = (float) mate.getSeno(angRotacao);
            cossenoAngRotacao = (float) mate.getCosseno(angRotacao);
92
            oldPixel = new float[3];
94          newPixel = new float[3];
            for (int ROW = 0; ROW < altura; ROW++) {
96              for (int COL = 0; COL < largura; COL++) {

98                  imagemLab.get(ROW, COL, oldPixel);

100                 newPixel[L] = oldPixel[L];
                    newPixel[A] = oldPixel[B] * senoAngRotacao + oldPixel[A] *
    cossenoAngRotacao;
```

```
102                     newPixel[B] = oldPixel[B] * cossenoAngRotacao - oldPixel[A]
        * senoAngRotacao;

104                     imagemFinal.put(ROW, COL, newPixel);
                    }
106             }
                return imagemFinal;
108         }
    }
```

Processamento.java

REFERÊNCIAS

[1] MACHADO, G. M. *A Model for Simulation of Color Vision De diency and A Color Constrast Enhancement Techinique for Dichromats*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, 2010.

[2] MACDONALD, L. Using color effectively in computer graphics. *Computer Graphics and Applications, IEEE*, v. 19, n. 4, p. 20–35, Jul 1999. ISSN 0272-1716.

[3] FRANCISCO, E. *Guia sobre Cores Cores e Acessibilidade*. Abril 2014. Disponível em: <http://chiefofdesign.com.br/cores-e-acessibilidade>. Acesso em: 2015/05/31.

[4] VILAR, M. *Testes cores de Ishihara*. Disponível em: <http://marcelovilar.com.br/teste-de-cores-de-ishihara-2/>.

[5] STAUDEK, T. *We work with a color computer*. agosto 2004. Disponível em: <http://digiarena.e15.cz/pracujeme-s-barvou-na-pocitaci_4/ch-47505>. Acesso em: 2015/05/07.

[6] KUHN, G. R. *Image Recoloring for Color-Vision Deficients*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, Abr 2009.

[7] PORET, S.; DONY, R.; GREGORI, S. Image processing for colour blindness correction. In: *Science and Technology for Humanity (TIC-STH), 2009 IEEE Toronto International Conference*. [S.l.: s.n.], 2009. p. 539–544.

[8] AWARENESS, C. B. *Colour Blind Awareness*. 2006. Disponível em: <http://www.ryobi-sol.co.jp/visolve/en/>. Acesso em: 06/06/2014.

[9] WANG, M. et al. In-image accessibility indication. *Multimedia, IEEE Transactions on*, v. 12, n. 4, p. 330–336, June 2010. ISSN 1520-9210.

[10] BRUNI, L. F.; CRUZ, A. A. V. e. Sentido cromático: tipo de defeitos e testes de avaliação clínica. *Arquivo Braileiro de Oftalmoogia.*, v. 69, n. 5, p. 766–775, 2006. Acesso em: 2014/06/09.

[11] CARDOSO, L. *Colorbldingin*. abril 2015. Disponível em: <https://goo.gl/d4nWNG>. Acesso em: 07/05/2015.

[12] CORP, R. S. *Diretrizes: Caso de Teste*. 2003. Disponível em: <http://www.funpar.ufpr.br:8080/rup/process/modguide/md_tstcs.htm>. Acesso em: 2014/06/10.

[13] DESVIGNES, S. *Colorblind Design*. 2012. Disponível em: <http://www.color-blindness.com/coblis-color-blindness-simulator>. Acesso em: 2015/05/31.

[14] MACHADO, G. M.; OLIVEIRA, M. M. Real-time temporal-coherent color contrast enhancement for dichromats. *Computer Graphics Forum*, v. 29, n. 3, p. 933–942, June 2010. Proceedings of EuroVis.

[15] FILHO, O. M.; NETO, H. V. *Processamento Digital de Imagens*. 1999.

[16] PEDRINI, H.; SCHWATZ, W. R. *Análise de Imagens Digitais*. 2008.

[17] CHENG, H. et al. Color image segmentation: advances and prospects. *Pergamon*, 2000. Acesso em: 2014/06/09.

[18] CIE 1976 L*a*b* Colour space. *ISO 11664-4*. Abril 2008. Disponível em: <http://www.iso.org/iso/catalogue_detail.htm?csnumber=52497>.

[19] ALBUZ, E.; KOCALAR, E.; KHOKHAR, A. Quantized cielab* space and encoded spatial structure for scalable indexing of large color image archives. v. 6, p. 1995–1998 vol.4, 2000. ISSN 1520-6149.

[20] BRADSKI, G. Open source computer vision. *Dr. Dobb's Journal of Software Tools*, 2000. Disponível em: <http://opencv.org/>. Acesso em: 2015/06/24.

[21] KUHN, G.; OLIVEIRA, M.; FERNANDES, L. A. An efficient naturalness-preserving image-recoloring method for dichromats. *Visualization and Computer Graphics, IEEE Transactions on*, v. 14, n. 6, p. 1747–1754, Nov 2008. ISSN 1077-2626.

[22] ORACLE. *GlassFish - World's first Java EE 7 Application Server*. Out 2014. Disponível em: <https://glassfish.java.net/>. Acesso em: 2015/07/03.

[23] W3C. *Cross-Origin Resource Sharing*. Anne van kesteren. [S.l.]. Disponível em: <http://www.w3.org/TR/2014/REC-cors-20140116/#acknowledgments>. Acesso em: 06/05/2015.