Alquimia Digital
Programação de Baixo Nível

Antônio Pasquali Coelho[1]

[1]Curso de Ciência da Computação - Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil

1. Introdução

A alquimia é uma prática antiga que abrange uma ampla gama de disciplinas conforme apresentado no enunciado do trabalho. Embora seus principais objetivos, como a transmutação de metais e a criação de um elixir da vida tenham sido desacreditados, este trabalho foi baseado nesse conceito e é onde exploraremos a ideia de transmutação de imagens, ou seja, iremos transformar uma imagem em outra por meio de um algoritmo que compara os pixels e os reorganiza para assim tornar-se outra imagem.

2. Desenvolvimento

Durante a execução desse trabalho, forama pensadas diversas maneiras para a solução do problema apresentado, primeiramente foi criada uma struct chamada *PaletaCores*, porém o algoritmo pensado acabou não sendo muito útil. Logo após essa tentativa, na segunda vez, foi encontrado um código que calculava a distância entre as cores e retornava a distância percentual entre duas cores representada por dois pixels da estrutura *RGBpixel*.

2.1. Construção do algoritmo

Para a construção do algoritmo foram feitos os seguintes passos:

2.1.1. Definir funções

O código começa definindo duas funções, *distanciaCor* e *trocaPixels*. A primeira calcula a distância Euclidiana das cores entre dois pixels e a segunda faz troca de dois pixels em uma imagem.

2.1.2. Copiar imagem de origem para imagem de saída

O código então copia a imagem de origem para a imagem de saída. Isso é feito percorrendo cada pixel na imagem de origem e copiando seus valores de cor (vermelho, verde e azul) para o pixel correspondente na imagem de saída.

2.1.3. Encontrar o pixel correspondente mais próximo

O código então percorre cada pixel na imagem desejada. Para cada pixel, ele calcula a distância da cor para cada pixel na imagem de saída e rastreia o pixel correspondente mais próximo.

2.1.4. Trocar pixels

Depois que o pixel correspondente mais próximo na imagem de saída for encontrado, o código troca esse pixel pelo pixel correspondente na imagem desejada. Isto é feito utilizando a função *trocaPixels* definida anteriormente.

2.2. Validação de cada passo

Para a validação de cada etapa do algoritmo, podemos pensar da seguinte maneira:

2.2.1. Definir funções

Para validar essa etapa, primeiro verificamos se as funções estão definidas corretamente e se não produzem erros do compilador. Também podemos escrever testes de unidade para essas funções para garantir que funcionem conforme o esperado.

2.2.2. Copiar imagem de origem para imagem de saída

Podemos validar esta etapa comparando a imagem de saída com a imagem de origem após esta etapa. Eles deveriam ser idênticos.

2.2.3. Encontrar o pixel correspondente mais próximo

A validação do seguinte passo acontece verificando se o código identifica corretamente o pixel correspondente mais próximo para alguns casos de teste. Deve-se calcular manualmente as distâncias das cores e comparando-as com os resultados do código.

2.2.4. Trocar pixels

Verificamos se os pixels foram trocados corretamente na imagem de saída. Validamos a operação ao compararmos a imagem de saída com a imagem desejada após esta etapa. Os pixels trocados devem ter os mesmos valores de cores em ambas as imagens.

2.3. Pseudocódigo

A seguir a descrição do algoritmo criado em pseudocódigo:

Função *distanciaCor*(pixel1, pixel2):
rDiff = diferença de valores de vermelho entre pixel1 e pixel2
gDiff = diferença de valores de verde entre pixel1 e pixel2
bDiff = diferença de valores de azul entre pixel1 e pixel2
Retorna a raiz quadrada de (0.299 * rDiff * rDiff + 0.587 * gDiff * gDiff + 0.114 * bDiff * bDiff)

Função *trocaPixels*(imagem, index1, index2):
Armazena o pixel no index1 da imagem em uma variável temporária aux
Define o pixel no index1 da imagem como o pixel no index2

Define o pixel no index2 da imagem como aux

Para cada pixel i na imagem:
Copia o pixel de ORIGEM para SAIDA

Para cada pixel i na imagem DESEJ:
Inicializa indProx como 0
Inicializa distProx como a distância de cor entre o pixel i na DESEJ e o primeiro pixel em SAIDA

Para cada pixel j em SAIDA, incrementando de 50 em 50:
Calcula a distância de cor entre o pixel i na DESEJ e o pixel j em SAIDA
Se essa distância for menor que distProx:
Define indProx como j
Define distProx como essa distância

Troca o pixel i em SAIDA pelo pixel em indProx

3. Conclusão

Em suma, este trabalho se mostrou um grande desafio para mim, visto que nunca havia utilizado nenhum conceito de cores em algoritmos, porém acredito que tenha sido uma etapa extremamente importante no meu aprendizado. Minha principal dificuldade foi encontrar um jeito de calcular e quantificar o valor de cada pixel para achar o mais próximo e acredito que uma potencial melhoria para o meu código seria torná-lo mais otimizado para reduzir o tempo de processamento ou aprimorando a métrica de comparação entre cores para obter resultados ainda mais precisos, visto que a seu processamento pode demorar um pouco e se as imagens forem muito diferentes, não teremos um resultado muito agradável.

4. Referências e sites utilizados

https://pt.stackoverflow.com/questions/193427/como-comparar-dois-tons-de-cores-e-dar-a-porcentagem-de-similaridade-java
http://satyarth.me/articles/pixel-sorting/
https://www.resizepixel.com/
https://stackoverflow.com/questions/66462697/how-to-compare-two-image-with-c
https://stackoverflow.com/questions/189943/how-can-i-quantify-difference-between-two-images