Desenvolvimento de um plugin de delineação para o ambiente ImageJ

Development of a delineation filter plugin for ImageJ

Nancy Baygorrea
Bolsista PCI, Matemática, D.Sc.

Otávio da Fonseca Martins Gomes
Supervisor, Eng. Químico, D. Sc.

Resumo

Este trabalho apresenta-se a implementação de um *plugin*, em linguagens Java e Python, para ser inserido no ambiente ImageJ, para a delineação de imagens de microscopia eletrônica de varredura (MEV) de minérios. Resultados experimentais mostram que o *plugin* Delin_vips proposto é efetivo e eficiente no ambiente ImageJ para imagens típicas da ordem de 1 MPixel até imagens com GPixels. Assim, além da aplicação original que motivou seu desenvolvimento, ele pode ser considerado como um *plugin* de uso geral para processos de delineação.

Palavras chave: filtro de delineação; microscopia de minérios; MEV; Libvips; ImageJ.

Abstract

In this paper, we introduce a built-in Java class of a delineation filter algorithm for ImageJ. We perform experiments on Java class and Python script using Libvips library. The experimental results show that the delineation filter *plugin* is effective and efficient in the ImageJ environment for typical images on the order of 1 MPixel to images with GPixels. Thus, in addition to the original application that motivated its development, it can be considered as a general-purpose plugin for delineation filtering.

Key words: delineation filter; ore microscopy; SEM; Libvips; ImageJ.

1. Introdução

O Processamento Digital de Imagens (PDI) consiste em um conjunto de técnicas que utilizam operações matemáticas para realçar imagens e/ou extrair dados quantitativos de imagens (GONZALEZ e WOODS, 2002). No campo da caracterização tecnológica de minérios, PDI é usado em conjunto com técnicas de microscopia para determinação de características importantes do minério, como associações minerais, textura e liberação.

Um procedimento típico de PDI compreende as etapas de aquisição de imagem, pré-processamento, segmentação, pós-processamento, extração de características e classificação. O pré-processamento é a etapa que intenciona melhorar a imagem, corrigindo defeitos oriundos da aquisição e realçando detalhes de interesse, de modo a facilitar sua visualização ou segmentação (GOMES, 2007). Isto é geralmente realizado através de aritmética de imagens e operações de filtragem (GONZALEZ e WOODS, 2002).

Em PDI, considera-se dois domínios de filtragem, o espacial e o de frequência. Os métodos de filtragem que trabalham no domínio espacial, aqueles que operam diretamente sobre os pixels de uma imagem, utilizam normalmente operações de convolução com máscaras chamadas de *kernel* ou peso, o qual é definido pela relação: $g(x,y) = T(f(x,y))$ onde, $f(x,y)$ é a imagem de entrada a ser filtrada, $g(x,y)$ é a imagem na saída e T é um operador sobre f definido em alguma vizinhança do pixel de posição (x,y). Assim, se usamos uma máscara $n \times m$, o esquema do processo de convolução por filtragem espacial linear é dado pela relação:

$$g(x,y) = \sum_{i=-m}^{m} \sum_{j=-n}^{n} f\,(x-i, y-j) * h(i,j), \qquad (1)$$

onde $h(i,j)$ é denotado como a matriz filtro com $i \in \{-m, \ldots, m\}$ e $j \in \{-n, \ldots, n\}$, como os índices de coluna e linha, respectivamente.

Por outro lado, qualquer filtro que não seja uma soma ponderada de suas entradas será um filtro não linear, isto é, a imagem de saída não é uma função linear das suas entradas. Nesse caso, os somatórios da Equação 1 são substituídos por algum tipo de operador não linear:

$$g(x,y) = O_{i,j}[f(x-i, y-j) * h(i,j)] \qquad (2)$$

Para muitos filtros não lineares, os elementos de $h(i,j)$ são todos 1. Denotemos $O_{i,j,1}$ quando $h(i,j) = 1$. Alguns exemplos de filtros não lineares são: os filtros de máximo e mínimo, qual o filtro de máximo localiza os pontos mais claros na vizinhança e o filtro de mínimo determina os pontos mais escuros, com $O_{i,j,1} = max_{i,j}$ e $O_{i,j,1} = min_{i,j}$, respectivamente; o filtro de ponto médio, que calcula o ponto médio entre os valores máximo e mínimo na vizinhança, com $O_{i,j,1} = \frac{1}{2}(max_{i,j} + min_{i,j})$; e o filtro de mediana, baseado no ranqueamento dos valores dos pixels, que retorna o valor intermediário dos valores de entrada (GONZALEZ e WOODS, 2002).

De fato, desde que Tukey (1971) introduziu os filtros médios em processamento de sinais, muitos filtros para PDI foram desenvolvidos. Os filtros de delineação, filtros que preservam e/ou reforçam as bordas entre regiões com cores ou texturas diferentes, são filtros não lineares que se constituem em uma ferramenta útil para uma

variedade de aplicações em PDI e visão computacional. Tais filtros operam buscando por transições entre fases e escolhendo a qual fase o pixel em questão pertence, ver King e Schneider (1993). Em geral, filtros de delineação são implementados computacionalmente usando filtros de detecção de bordas (GONZALEZ e WOODS, 2002) ou através de morfologia matemática, ver Serra (1982, 1988).

Em microscopia eletrônica de varredura (MEV) de minérios, um artefato importante é o chamado efeito o halo, que pode causar erros na análise das imagens. Filtros de delineação são capazes de reduzir o típico halo que circunda fases, borrando sua interface com outras fases com diferentes níveis de cinza, em imagens de elétrons retroespalhados de espécimes polidos. Se esse artefato não for eliminado, pode ocorrer a segmentação de uma fase junto com as bordas de outra fase ou até mesmo a segmentação de uma fase intermediária inexistente. O filtro de delineação converte as transições graduais de níveis de cinza que definem os limites entre fases em passos abruptos de modo que a transição de uma fase para a outra seja realizada em um único pixel (KING e SCHNEIDER, 1993).

2. Objetivos

Pretendeu-se criar um *plugin* de filtro de delineação funcional para ser utilizado no ambiente ImageJ (SCHNEIDER et al., 2012). Para tal fim, utilizou-se o algoritmo de delineação descrito por King e Schneider (1993).

Foi definida então, a seguinte metodologia:

- Criação de um script na linguagem Python contendo o código do algoritmo descrito na Figura 1.
- Criação de um código-fonte Java que ligue códigos Python utilizando a biblioteca Libvips (CUPITT e MARTINEZ, 1996) dentro do ambiente do ImageJ.
- Criação de um repositório Github, hospedado em https://github.com/lab-mev-cetem/DelinVips, contendo todos os arquivos desenvolvidos para nosso projeto, tanto o código fonte Python quanto o código-fonte Java do *plugin*.

3. Material e Métodos

Foi utilizado um algoritmo de delineação não linear, descrito por King e Schneider (1993), conforme apresentado na Figura 1 e na Figura 2. Para nosso propósito, uma biblioteca de processamento rápido de imagens com baixa necessidade de memória chamada Libvips foi utilizada. Nossa implementação Python (versão 3.6) do algoritmo de delineação, mostrada na Figura 2, está hospedada no repositório GitHub https://github.com/lab-mev-cetem/DelinVips/blob/master/docs/delin.py.

Vale a pena ressaltar que o *script* Python proposto, pode ser compilado em qualquer interpretador Python com o seguinte comando:

> *Python nome_script_py input_image_with_extension output_image_with_extension valor_threshold*

Algorithm 1: Delin algorithm

Input : Set p and $\mathcal{N}(p)$ as the input image pixel and its neighborhood, respectively. Take $Imin$ and $Imax$ as the minimum and maximum intensity of each $\mathcal{N}(p)$. Take T as given limiar. Set $D = Imax - Imin$.

Output: Obtained output image pixel ps.

```
if D < T then
    // Pixel p is no a border ones.
    ps = p
else
    // Pixel p is a border ones.
    if p − Imin < Imax − p then
        ps = Imin
    else
        ps = Imax
    end if
end if
return ps
```

Figura 1. Algoritmo do filtro de delineação.

```python
#  Teste-prog/teste-py/delin_vips.py
import sys
import pyvips

#
# T = threshold
T = float(sys.argv[3])

#Load image file
ps = pyvips.Image.new_from_file(sys.argv[1], access='sequential')

# window size
window_size = 3

# find the max and min for each window
ps_min = ps.rank(window_size, window_size, 0)
ps_max = ps.rank(window_size, window_size, window_size * window_size - 1)

# difference
D = ps_max - ps_min

#for border pixels,
border = (ps - ps_min < ps_max - ps).ifthenelse(ps_min, ps_max)

#  output image
out_ps = (D < T).ifthenelse(ps, border)

# Write  output image into  folder where initial file is to.
out_ps.write_to_file(sys.argv[2])

```

Figura 2. Código do filtro de delineação implementado na linguagem Python.

Além disso, usando o eclipse como IDE para construir um projeto Maven, um *plugin* funcional Java chamado Delin_Vips com janela de busca de 3x3 foi desenvolvido, para plataformas Windows e Linux, com o objetivo de inseri-lo dentro do ambiente do ImageJ (software de domínio público, feito em Java destinado a processamento de imagens). Contudo, usaremos o *plugin* contido na pasta do projeto no nosso repositório https://github.com/lab-mev-cetem/DelinVips para verificar o desempenho do *plugin* em imagens de teste.

4. Resultados e Discussão

Para verificar a funcionalidade do *plugin* proposto, utilizamos em nossos experimentos imagens de microscopia eletrônica de varredura de seções polidas de um minério sulfetado de cobre. A descrição dessas seções e o procedimento de aquisição de imagens põem ser encontrados em Gomes et al. (2014). A imagem teste, apresentada na Figura 3, tem de resolução de 1024x1024 pixels, arquivo TIFF, 8 bit e de 1,1 MB. Assim, para essa imagem e com o parâmetro T = 40, o tempo de execução do *plugin* Delin_Vips foi de 0,792 s.

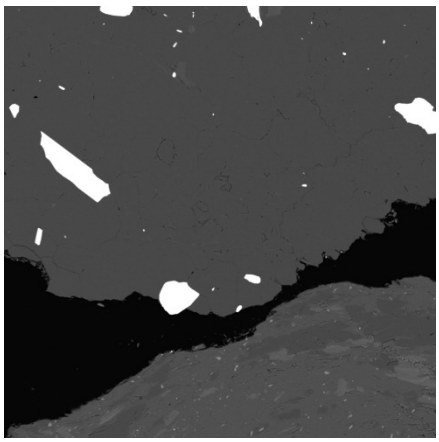

Figura 3. Imagem de teste.

A Figura 4 mostra em detalhe uma região da imagem de teste (Figura 3) e das imagens resultantes após a aplicação do *plugin* Delin_Vips com valores de T de 40 e 100, respectivamente. Observando essas imagens, fica evidente que as transições entre as fases ficam mais nítidas após a filtragem.

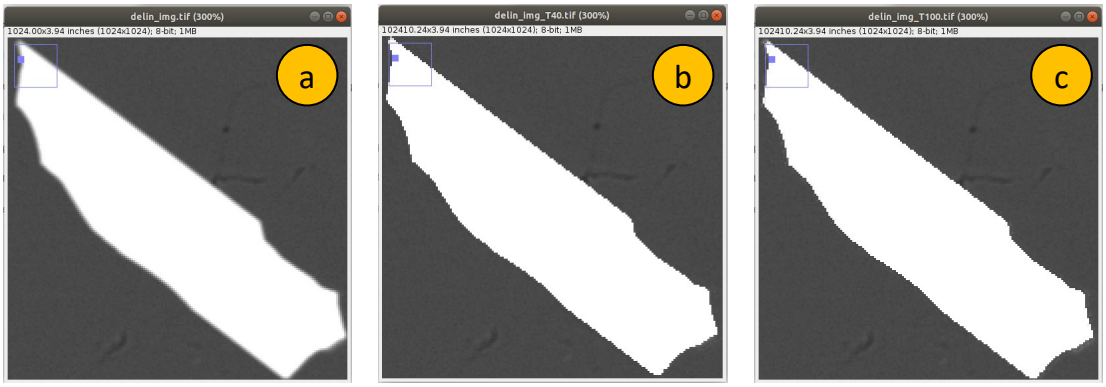

Figura 4. Detalhe da imagem de teste mostrando o resultado da aplicação do *plugin* Delin_vips: (a) imagem original; (b) resultado com T = 40; (c) resultado com T = 100.

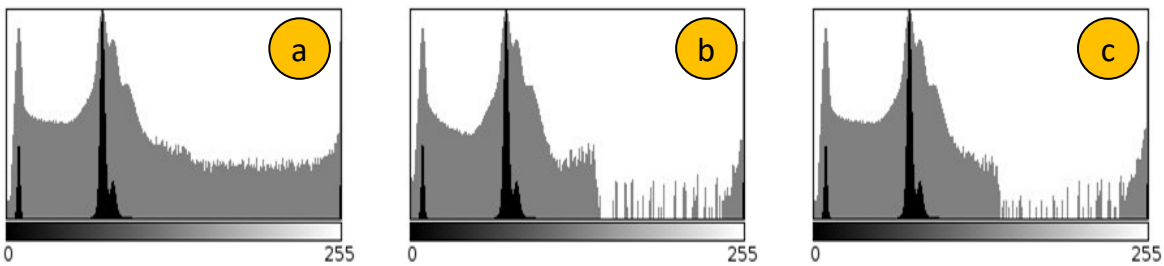

Figura 5. Histograma, em escalas linear e logarítmica, da: (a) imagem de teste (Figura 2); (b) imagem resultante da aplicação do *plugin* Delin_vips com T = 40; (c) imagem resultante com T = 100.

A Figura 5 apresenta os histogramas, em escala linear (preta) e escala logarítmica (cinza), da imagem de teste (Figura 3) e das imagens resultantes da aplicação do *plugin* Delin_vips com T = 40 e T = 100.

Conforme pode ser visto na imagem da Figura 3 e nos histogramas da Figura 5, essa imagem tem basicamente 4 fases: a resina de embutimento (preta) com níveis de cinza em torno de 9; duas fases de ganga (cinza) com níveis de cinza em torno de 74 e 81; e o sulfeto (branco) com níveis de cinza perto de 255. Cada fase é evidenciada no histograma da imagem por um pico em torno de um nível de cinza característico. Os picos das três primeiras fases são visíveis, já o pico da quarta fase (branca) quase não é notado pois resume-se praticamente ao nível de cinza 255.

Observa-se nos histogramas mostrados na Figura 5, notadamente nos em escala logarítmica (em cinza), que a aplicação do *plugin* reduziu bastante a presença de pixels com nível de cinza na faixa entre 145 a 245. Essa faixa de níveis de cinza, no histograma da imagem original (Fig. 5a), representa os pixels nas interfaces entre a fase branca e as demais fases, os pixels do efeito halo. Com a aplicação do filtro de delineação, a esses pixels são atribuídos níveis de cinza correspondentes a uma das fases, praticamente eliminando os níveis de cinza de faixa, como pode ser visto nas Fig. 5b e Fig. 5c.

O fato da faixa entre 145 a 245 não ser visível no histograma em escala linear e ser visível em escala logarítmica indica que, em relação ao número total de pixels da imagem (1 MPixel), são poucos os pixels com esses níveis de cinza. Todavia, esses pixels estão concentrados nas bordas dos grãos (interface entre fases) e podem ser tomados como uma outra fase em um procedimento de segmentação. Sistemas de mineralogia automatizada sem filtro de delineação geralmente atribuem a essa outra fase uma composição química intermediária entre as fases realmente presentes no minério. Esse erro, além de adicionar um mineral ausente à assembleia mineralógica, pode causar erro na determinação da textura e exposição do minério, já que essa fase intermediária estaria encapsulando grãos.

Vale mencionar ainda que, mesmo para imagens muito grandes, o *plugin* proposto ainda preserva eficiência na execução do algoritmo. Em um teste feito com uma imagem com a mesma origem e características da imagem de teste da Figura 3, mas com 64000x32000 pixels e 2 GB, o tempo de execução do *plugin* Delin_vips foi de 1 m 15,466 s.

5. Conclusão

Neste trabalho, foi estudado o método de delineação para imagens, o qual é capaz de mitigar o efeito halo em imagens minerais obtidas por MEV e, consequentemente, melhorar sua segmentação, especialmente com a utilização de métodos clássicos de limiarização (*thresholding*).

Visando esse objetivo, foi desenvolvido o *plugin* Delin_vips, implementado em linguagem Java, para o ambiente ImageJ. O *plugin* funciona como um *binding* e pode ser usado com qualquer *script* Python que utilize a biblioteca Libvips.

O *plugin* Delin_vips encontra-se hospedado no repositório GitHub https://github.com/lab-mev-cetem/DelinVips.

6. Agradecimentos

Ao Programa de Capacitação Institucional (PCI) do CNPq, com processo número 300636/2019-9, pelo financiamento do projeto. Ao meu orientador, professor Otávio Gomes, pela parceria, apoio e motivação na pesquisa em processamento de imagens.

7. Referências Bibliográficas

CUPITT, J.; MARTINEZ, K. VIPS: An image processing system for large images, **Proc. SPIE**, v. 2663, p. 19-28, 1996.

GOMES, O. F. M. **Microscopia co-localizada: novas possibilidades na caracterização de minérios**. 2007. 103p. Tese (Doutorado) - Departamento de Ciência dos Materiais e Metalurgia, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro (Brasil).

GOMES, O.F.M.; OLIVEIRA D.M.; SOBRAL, L.G.S., PIRARD E. Characterization of Particle Damage and Surface Exposure of a Copper Ore Processed by Jaw Crusher, HPGR and Electro-dynamic Fragmentation. In: **Characterization of Minerals, Metals, and Materials 2014**. John Wiley & Sons, Ltd; 2014. p. 245–252.

GONZALEZ, R.C.; WOODS, R.E. **Digital Image Processing**. 2. ed. Upper Saddle River (NJ, USA): Prentice-Hall, 2002.

KING, R.P.; SCHNEIDER, C.L. An effective SEM-based image analysis system for quantitative mineralogy. **Kona Powder and Particle Journal**, v. 11, n. 0, p. 165-177, 1993.

SCHNEIDER, C.A.; RASBAND, W.S.; ELICEIRI, K.W. NIH Image to ImageJ: 25 years of image analysis. **Nature methods**, v. 9(7), p. 671-675, 2012.

SERRA, J. **Image Analysis and Mathematical Morphology**. London (UK): Academic Press, 1982.

SERRA, J. **Image Analysis and Mathematical Morphology:** Volume 2. London (UK): Academic Press, 1988.

TUKEY, J.W. **Exploratory Data Analysis**, Addison Wesley, 1971.