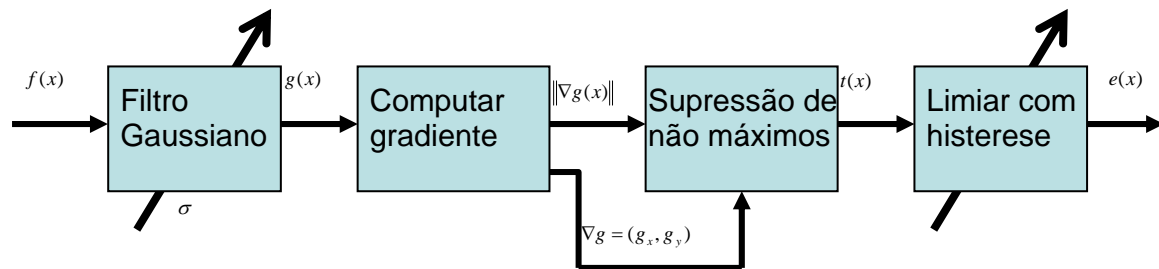


Roteiro

Segmentação e detecção de bordas

1. Introdução

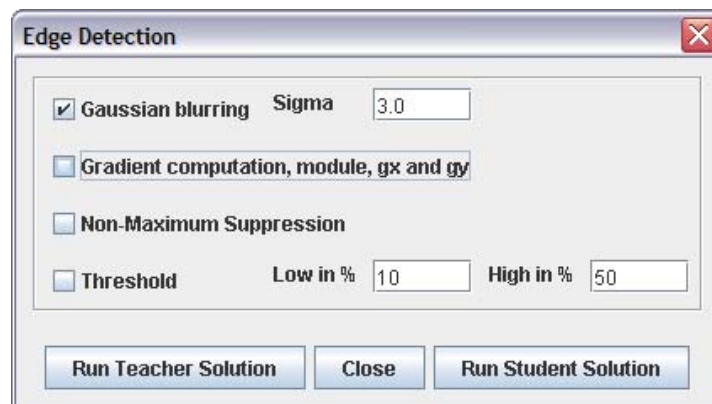
É proposto a implementação de um eficiente versão do detector de bordas de Canny-Derliche. O algoritmo será implementado passo a passo seguindo este diagrama.



Para chamar os métodos de detecção de bordas implementados no ImageJ, use o plugin **Edge Detection** e selecione as operações desejadas através da caixa de diálogo. Este plugin mostra apenas a saída da última operação. Use a imagem *test.tif* para verificar os diferentes passos do algoritmo.

Sintaxe Java

Computa a raiz quadrada	<code>y = Math.sqrt(x);</code>
Computa o valor absoluto	<code>y = Math.abs(x);</code>
Computa o arco tangente em radianos	<code>angle = Math.atan2(dy, dx);</code>
Valor de π	<code>Math.PI</code>
Get the length of an array A[]	<code>int n = A.length;</code>



2. Borramento Gaussiano

Escolhemos o filtro Gaussiano como operação de suavização. Ele é implementado eficientemente através de uma cascata de N filtros exponenciais simétricos. Na prática, optamos por N=3. O valor do desvio padrão σ (equivalente ao tamanho da janela) é entrado via caixa de dialogo.

2.1 Determinação dos pólos

A variância equivalente σ^2 de uma cascata de filtros é a soma das variâncias das operações das operações individuais; aqui, $\sigma^2 = 2Na/(1-a)^2$. Encontre a relação entre os valores de σ , N e os valores dos pólos usados na seção 2.2.

2.2 Implementação de um filtro Gaussiano 2D

Escreva o método `blurring()`, o qual implementa o filtro Gaussiano de modo separável, no arquivo `Code.java`. Existe um método chamado:

```
double[] out = Convolver.convolveIIR(double input[], doubles poles[]);
```

que opera o filtro simétrico IIR em um sinal 1D `input[]` com o filtro definido pelos seus pólos `poles[]`. Substitua `a=0.5` no código pelo valor correto do pólo ($a < 1$).

Se você não conseguir, chame `TeacherCode.blurring()`.

3. Computação do gradiente

3.1 Implementação de filtros de gradiente

Escreva o código para filtrar uma imagem com h_x e h_y no método `gradient()`. A saída do método deve um arranjo (array) com 3 imagens `ImageAccess grad[3]`:

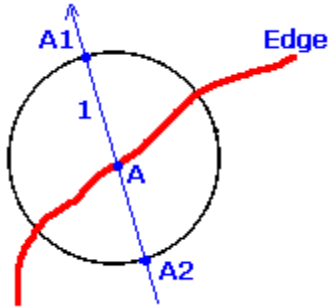
$$h_x = \frac{1}{12} \begin{bmatrix} -1 & 0 & 1 \\ -4 & 0 & 4 \\ -1 & 0 & 1 \end{bmatrix} \quad h_y = \frac{1}{12} \begin{bmatrix} -1 & -4 & -1 \\ 0 & 0 & 0 \\ 1 & 4 & 1 \end{bmatrix}$$

- `grad[0]` contem o módulo do gradiente
- `grad[1]` contem g_x , a saída do filtro h_x
- `grad[2]` contem g_y , a saída do filtro h_y

É recomendado a implementação separável que é mais rápida, mas você pode usar a não separável que é mais fácil de codificar se quiser. No primeiro caso, não se esqueça de implementar as condições de fronteira reflexivas explicitamente no filtro 1D.

Se você não conseguir implementar esta etapa, chame `TeacherCode.gradient()`.

4. Supressão de Não-Maximos



4.1 Determine as posições A1 e A2

Para um ponto $\mathbf{A} = (x_A, y_A)$ na imagem, compute as posições dos pontos:

- $\mathbf{A1} = (x_{A1}, y_{A1}) = \mathbf{A} + \mathbf{u}$
- $\mathbf{A2} = (x_{A2}, y_{A2}) = \mathbf{A} - \mathbf{u}$

onde \mathbf{u} é um vetor unitário na direção do gradiente

Evite usar funções trigonométricas, pois elas não são eficientes

4.2 Implementação

Escreva o código que implementa supressão de não-máximos usando o seguinte algoritmo: O valor da magnitude $G(\mathbf{A})$ é considerado apenas se ele for maior ou igual a $G(\mathbf{A1})$ e $G(\mathbf{A2})$ e se $G(\mathbf{A})$ não for igual a 0, caso contrário ele é suprimido; i.e., igualado a 0.

Para computar $G(\mathbf{A1})$ e $G(\mathbf{A2})$ use um método de interpolação que retorna um valor computado para a imagem em qualquer posição espacial (x_a, y_a) —não necessariamente inteiro. Este método da classe `ImageAccess` pode ser evocado assim:

```
double ga, xa, ya;  
ga = image.getInterpolatedPixel(xa, ya);
```

se você não conseguir implementar esta questão, chame
`Teacher.suppressNonMaximum()`.

5. Limiar com histerese

Os segmentos de contornos acima de T_{high} (limiar superior) são crescidos de uma maneira para incluir todos os pontos conectados maiores que T_{low} (limiar inferior). Os valores dos dois limiares são entrados via caixa de diálogos como porcentagens do gradiente máximo

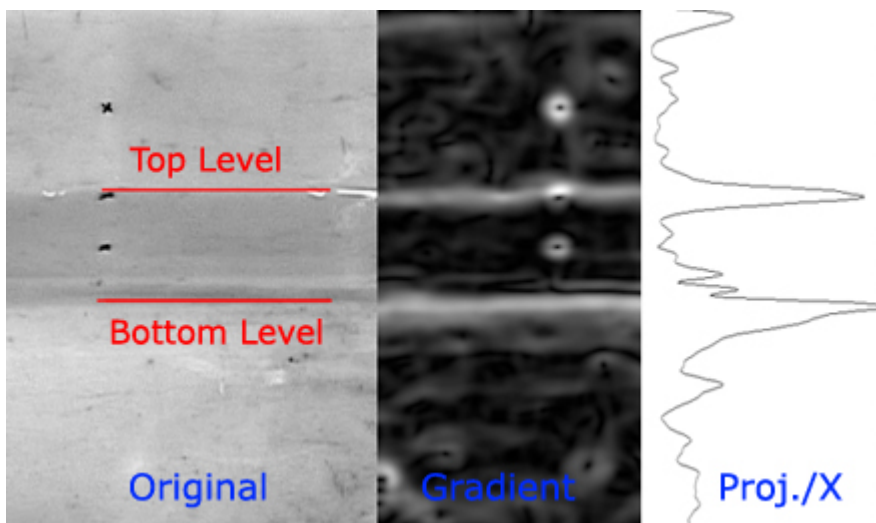
Aplique o detector de bordas para obter diferentes níveis de detalhes (como mostrado abaixo) nas bordas da imagem "mandelbrot.tif" ajustando os parâmetros σ , T_{high} and T_{low} . Salve os parâmetros e as imagens de saída e insira-os no relaorio.doc.



Imagem Original

Resultados Esperado: Detalhes Resultados Esperados: Global

6. Aplicação: medidas do nível da água



Propomos medir níveis de água baseados numa seqüência de imagens *water-level.tif*. Façamos as suposições que o nível da água é horizontal e que o reservatório nunca é completamente cheio ou completamente vazio. O método é baseado na detecção de dois máximos na projeção da imagem de gradiente.

6.1 Computar a projeção sobre o eixo Y

Escreva o método `computeXProjectionGradient` o qual computa a projeção de intensidades na direção X.

$$p(y) = \sum_{x=0}^{N_x-1} \|\nabla g(x, y)\|$$

Os metodos das seções prévias devem ser juntamente com o método `DisplayTools.plot(double p[], String labelAbscissa, String labelOrdinate)`; que deve ser usado para 'plotar' uma função.

Aplique o plugin **Compute XProjection Gradient** a uma imagem individual: *water-level-20.tif*. Reporte as duas posições y correspondentes aos dois valores máximos na projeção em *relatorio.doc*.

6.2 Detecção da primeira linha horizontal

Escreva o método `measureLevel()` o qual encontra a posição y_{\max} correspondents ao máximo da projeção para cada imagem da seqüência. A entrada de `measureLevel` é uma seqüência de imagens armazenadas num arranjo de `ImageAccess`.

- Desenhe uma linha horizontal em y_{\max} como um número inteiro t usando `DisplayTools.drawLine(int t, int ymax);`
- Imprima os valores $y_{\max}(t)$ usando `IJ.write()` ou 'plote' eles usando `DisplayTools.plot()` e preencha o *relatorio.doc*.

6.3 Detecção das duas linhas

Complete o método `measureLevel()` para achar as duas posições y correspondentes aos dois máximos da projeção para cada imagem da seqüência. O reservatório nunca está totalmente vazio; então, existem pelo menos 10 pixels entre os dois máximos.

- Desenhe um retangulo entre $y1$ e $y2$ na imagem número t usando `DisplayTools.drawLevels(int t, int y1, int y2);`

- Compute as alturas da água $h(t)$, imprima os valores ou 'plote-os', e preencha o *relatorio.doc*.