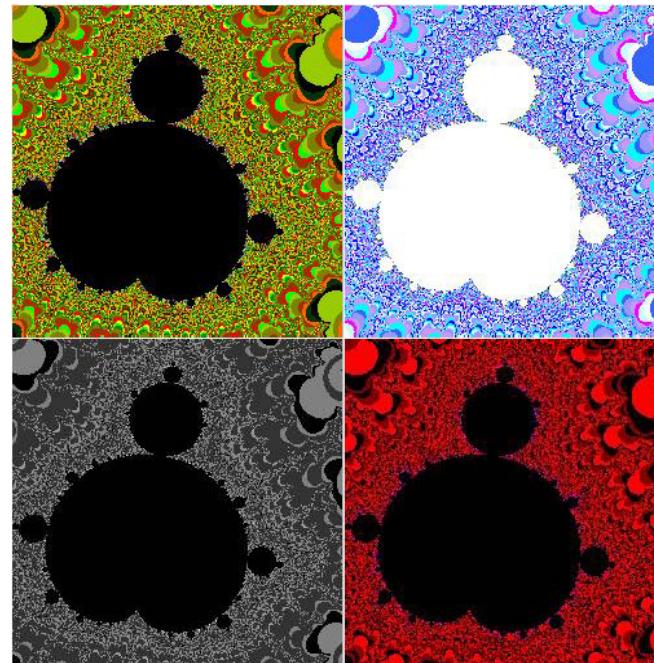




ERCEMAPI 2009
29 de Outubro
Parnaíba-PI



MC 5: Técnicas de Processamento Digital de Imagens com Java

Iális Cavalcante – Engenharia da Computação
UFC / Campus de Sobral

Objetivos

- Apresentar técnicas básicas que envolvem Processamento Digital de Imagens (PDI);
- Apresentar APIs Java voltada para PDI;
- Desenvolver uma pequena aplicação em Java para descrição de regiões.

E quem é o apresentador?

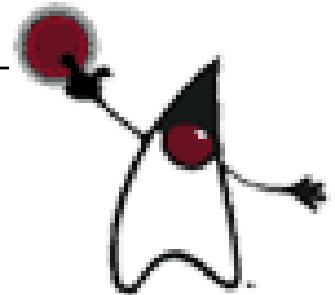


Agenda

1. Introdução
2. APIs Java para Imagem
3. Técnicas de Processamento Digital de Imagens (PDI)
4. Proposta de um Aplicativo de PDI
5. Considerações Finais



- Visão Geral sobre PDI
- Aplicações Reais

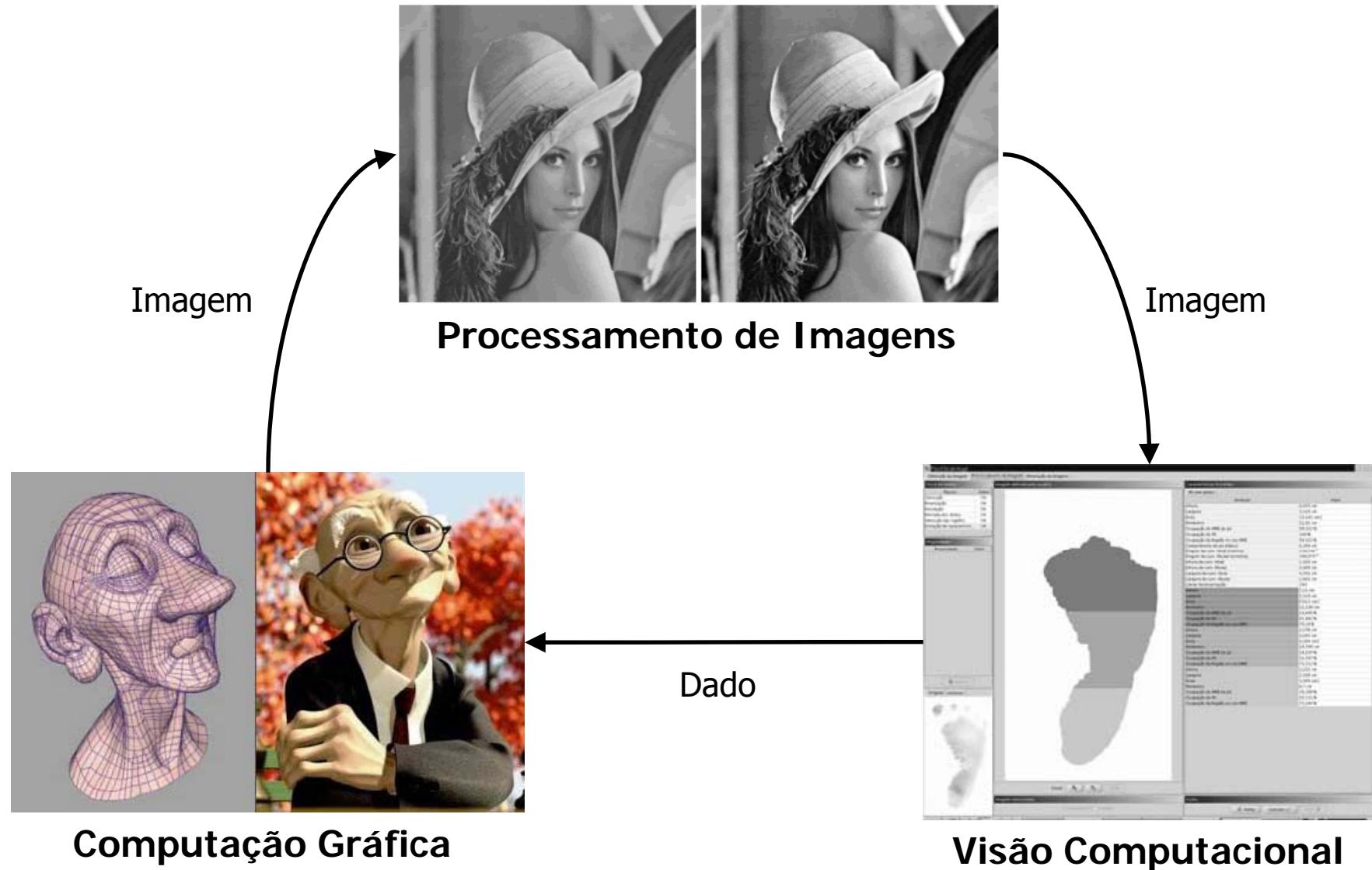


1. INTRODUÇÃO

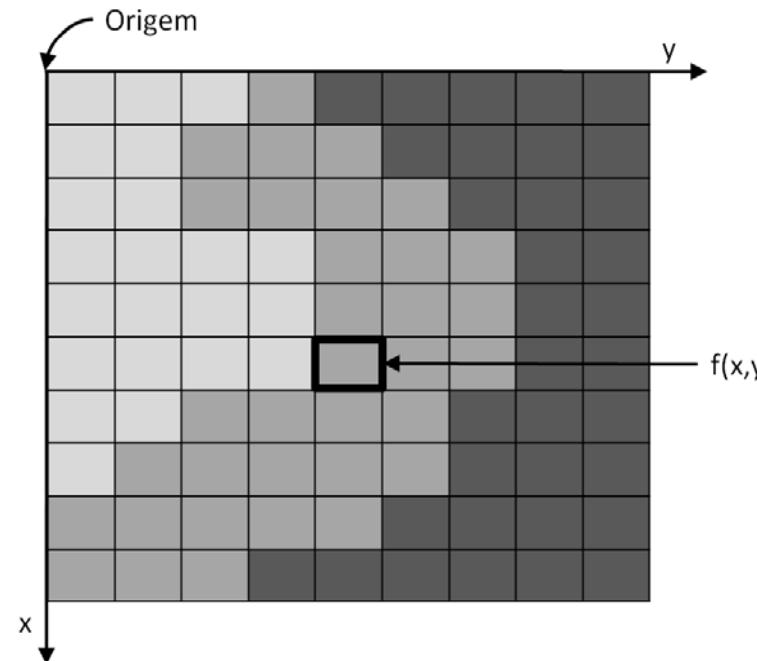
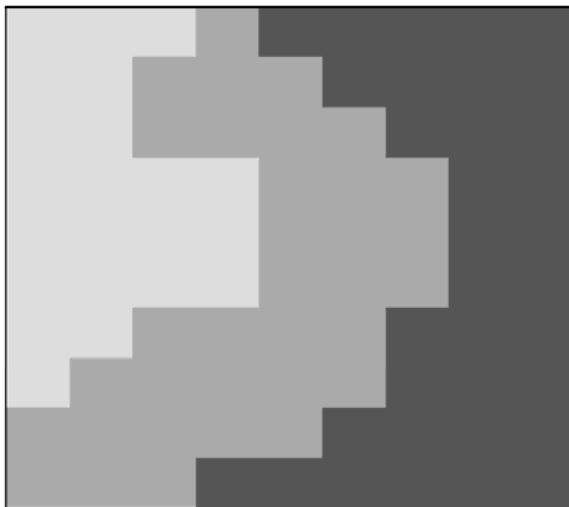
Introdução

- PDI possui alguns conceitos em comum com Visão Computacional e Computação Gráfica (caso de Análise de Imagens - *FootScanAge*).
- *Pixels* formam uma imagem – função $f(x,y)$
- Primeiros registros de estudo – NASA (Guerra Fria)
- Aplicações Multidisciplinares

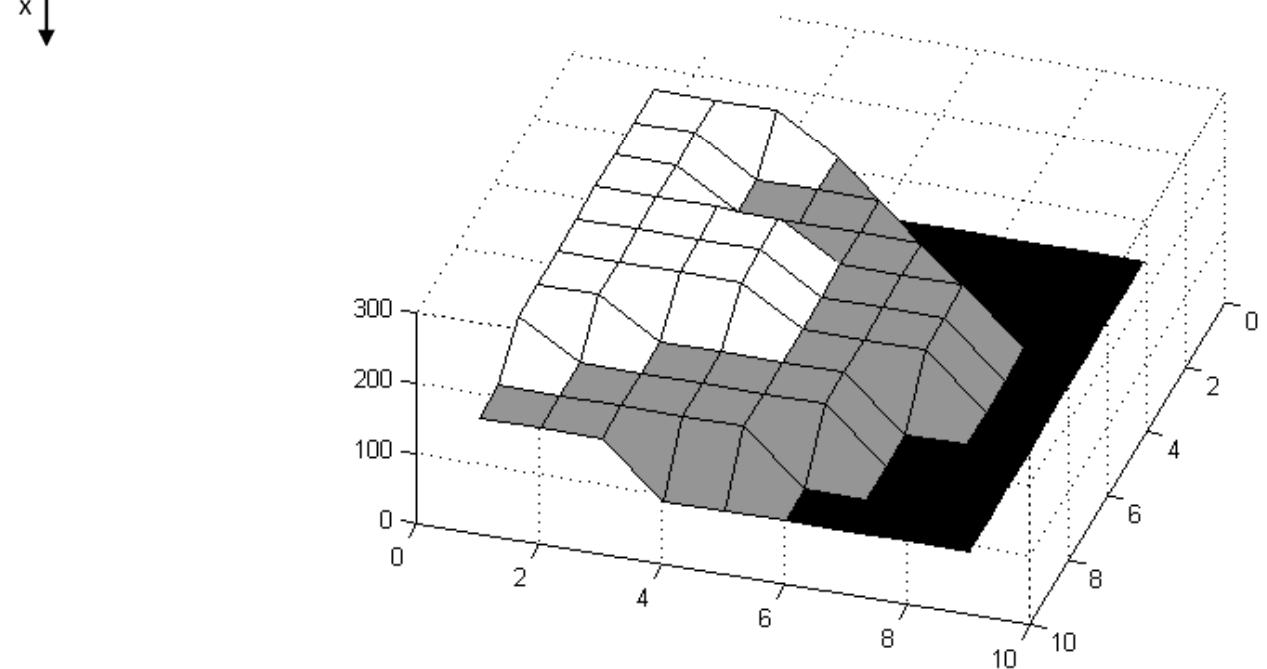
Comparativo



Definição de Imagem



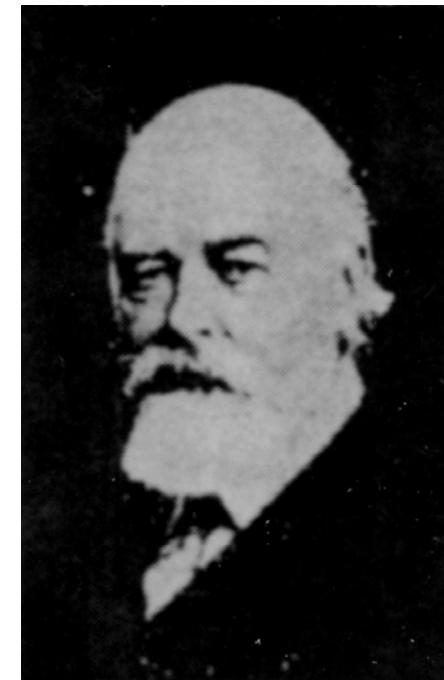
218	218	218	165	90	90	90	90	90	90
218	218	165	165	165	90	90	90	90	90
218	218	165	165	165	165	90	90	90	90
218	218	218	218	165	165	165	90	90	90
218	218	218	218	165	165	165	90	90	90
218	218	218	218	165	165	165	90	90	90
218	218	218	218	165	165	165	90	90	90
218	218	165	165	165	165	90	90	90	90
218	165	165	165	165	165	90	90	90	90
165	165	165	165	90	90	90	90	90	90



Histórico



1921



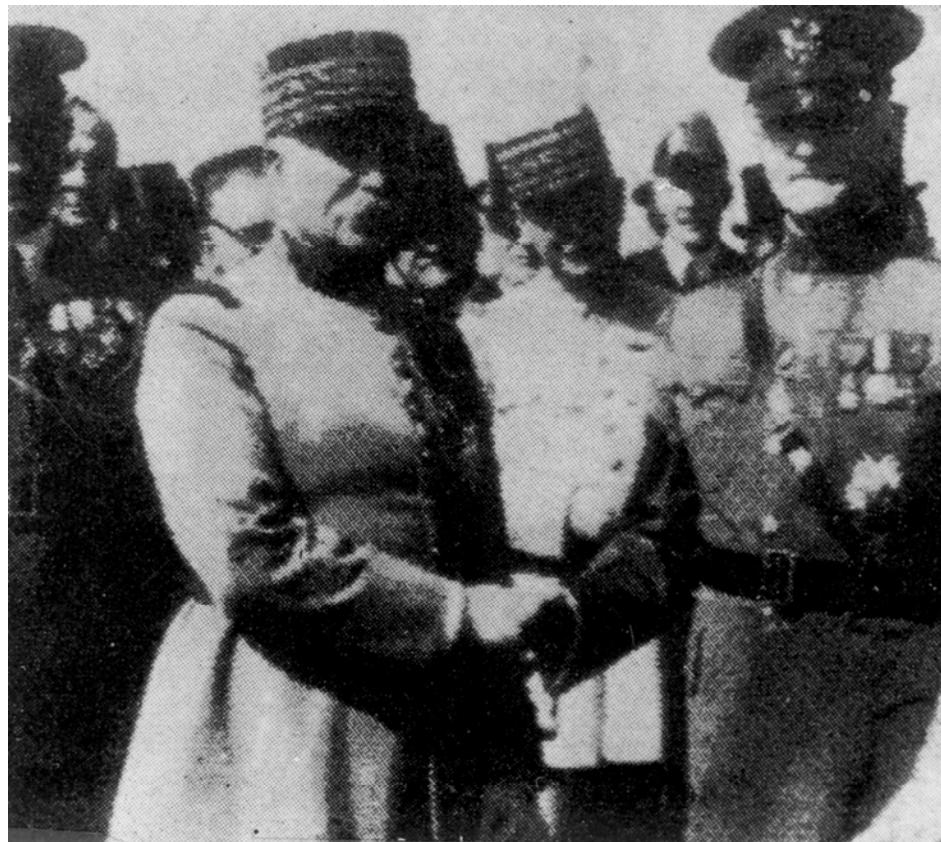
1922

Anos 20 – Sistema Bartlane

Transmissão de Imagens via cabo submarino

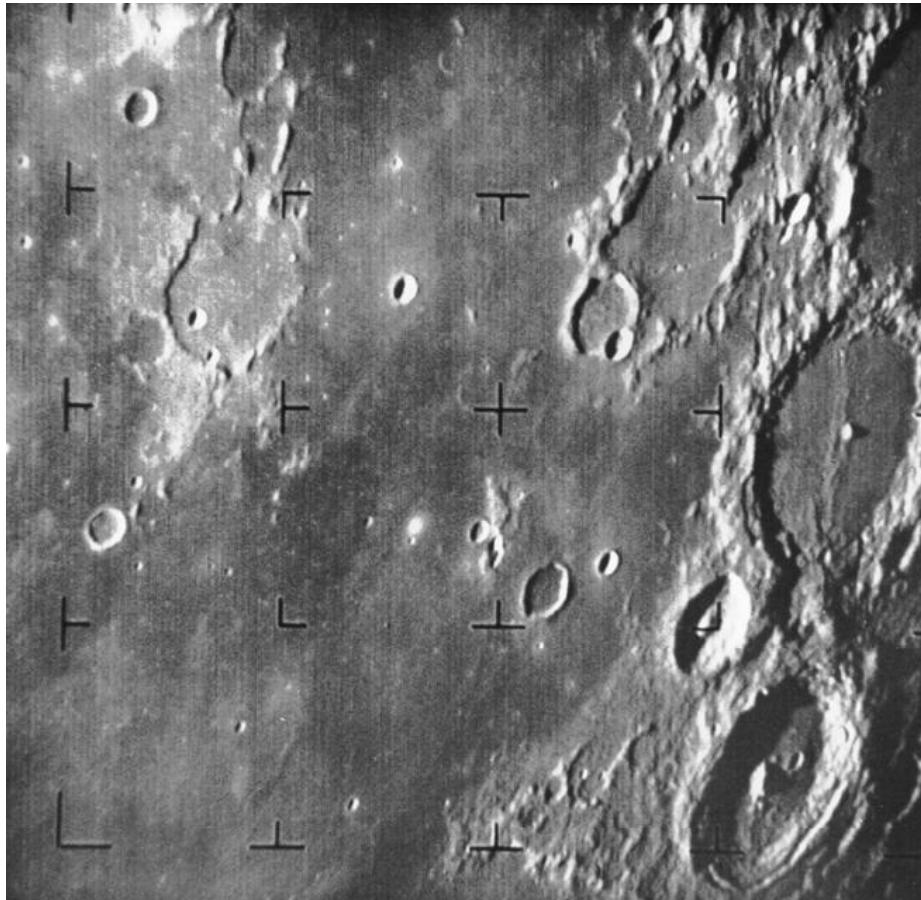
Uma semana para três horas – Londres para Nova York

Histórico



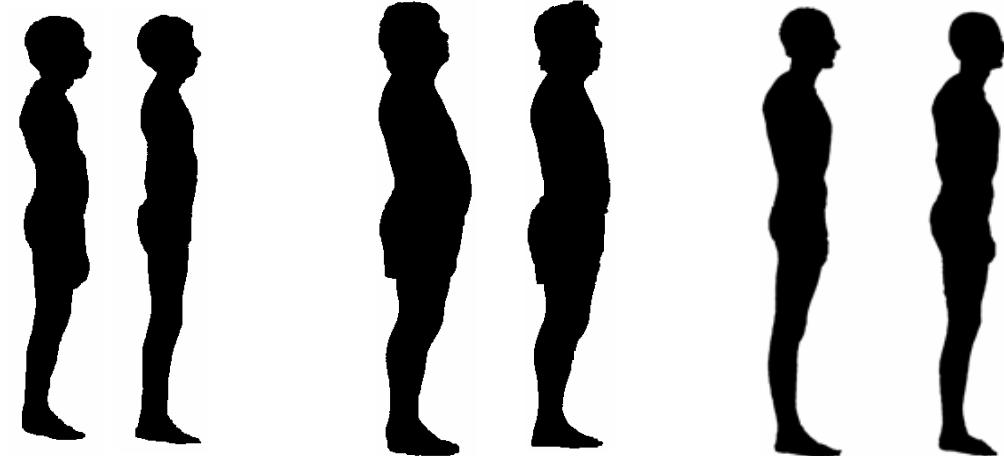
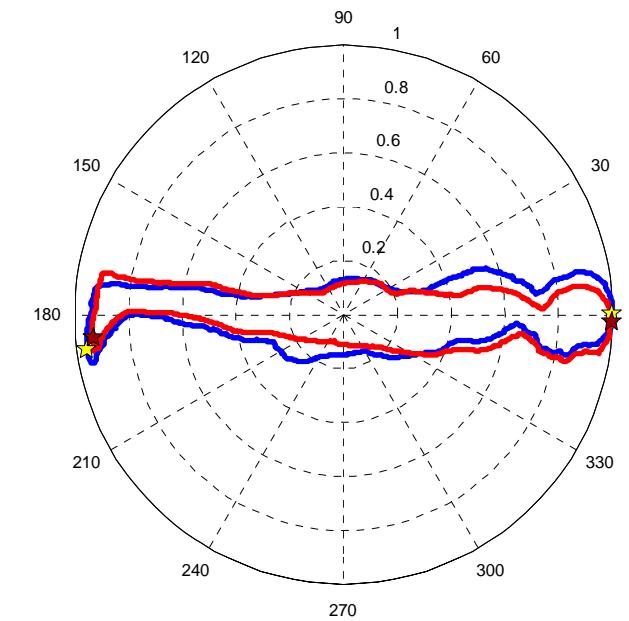
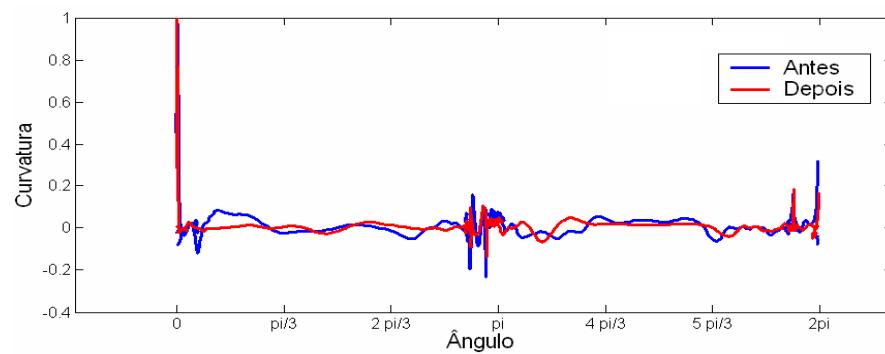
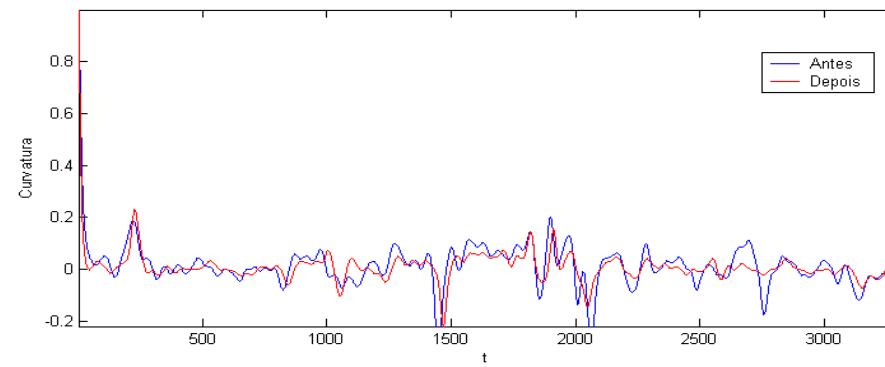
Anos 20 (1925) – Sistema Bartlane
Aumento para cinco níveis de brilho

Histórico

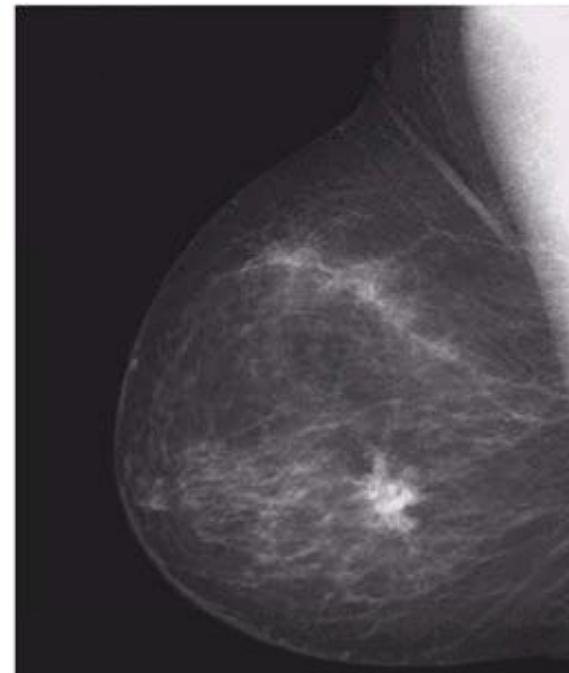
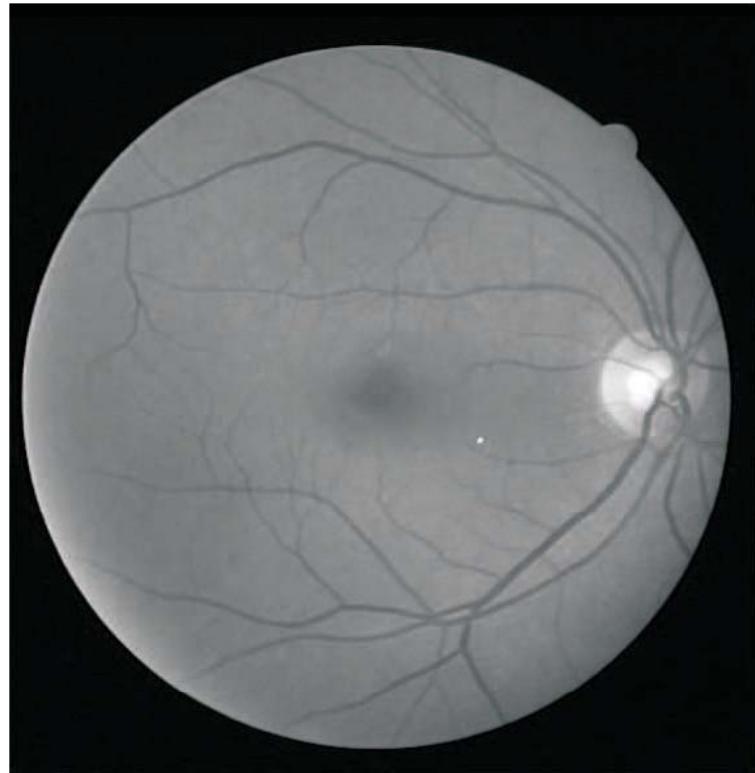


Primeira imagem da Lua.
Obtida pela nave espacial americana Ranger 7.
31 de julho de 1964, 9:09 (17 minutos antes do impacto).

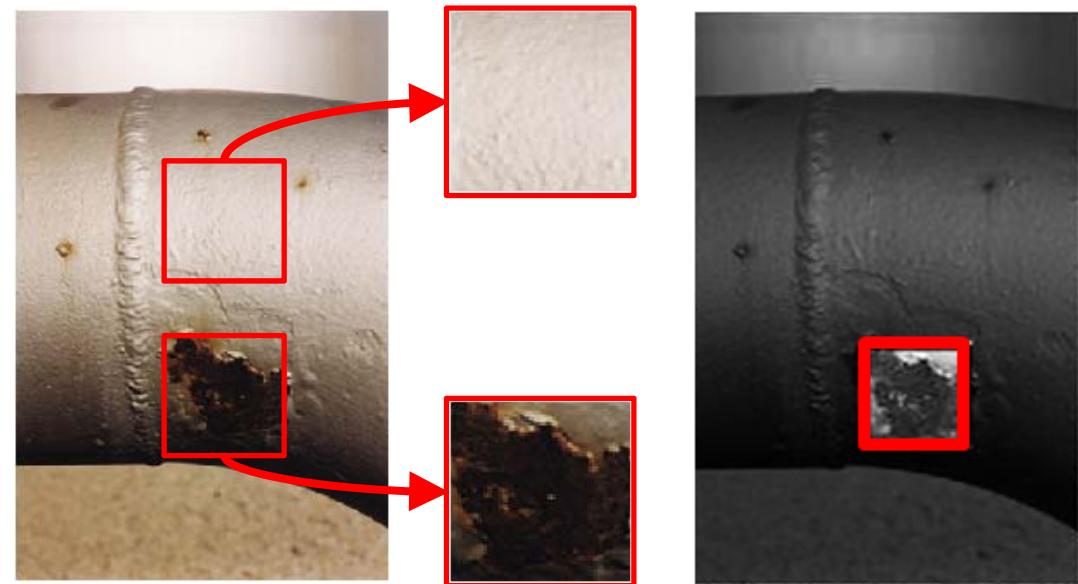
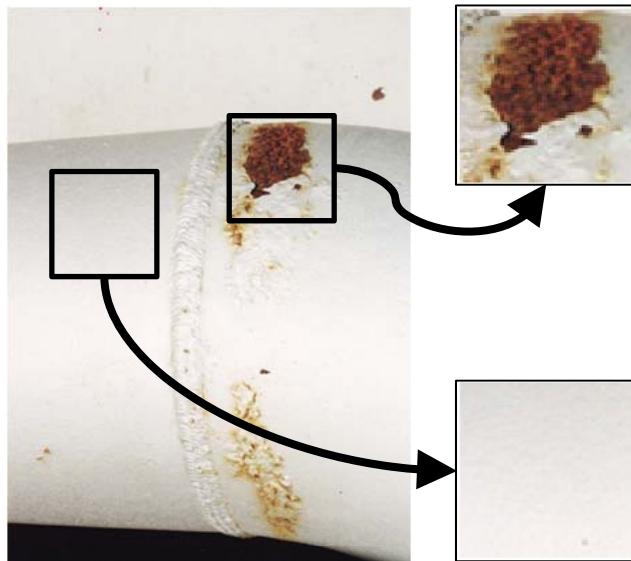
Aplicações em Saúde



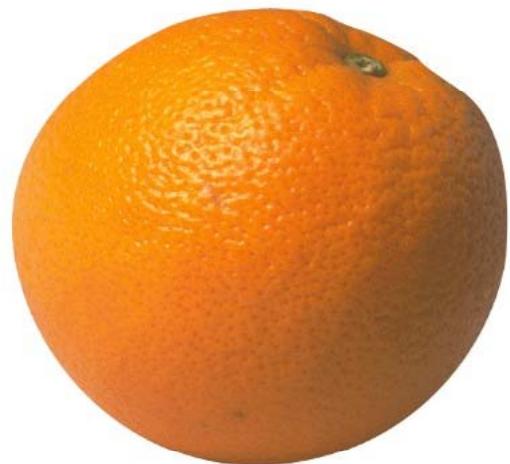
Aplicações em Saúde



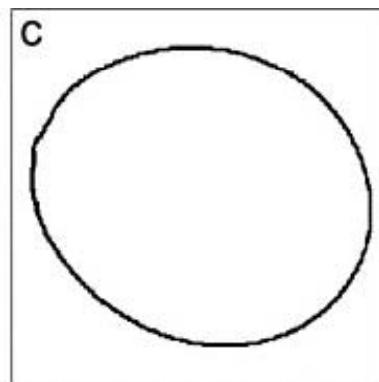
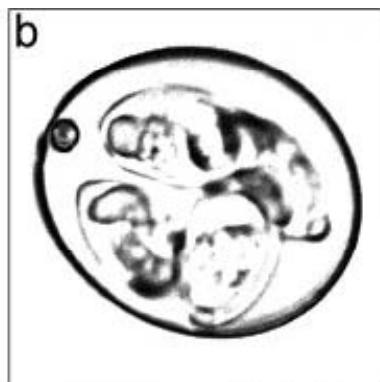
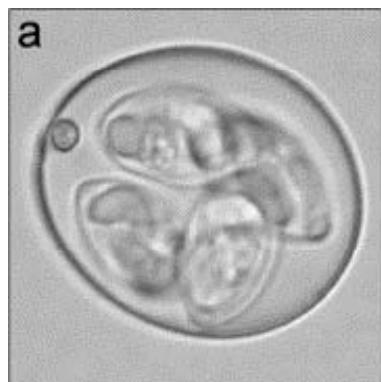
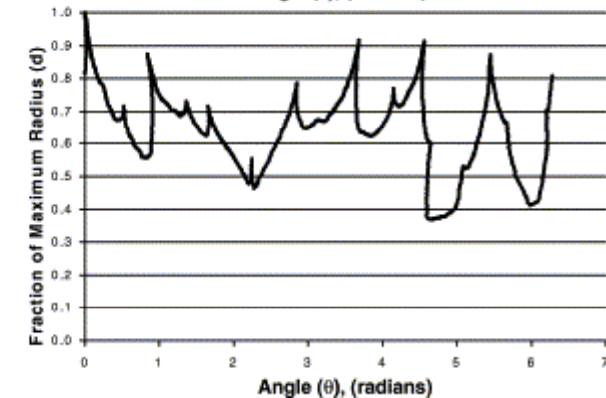
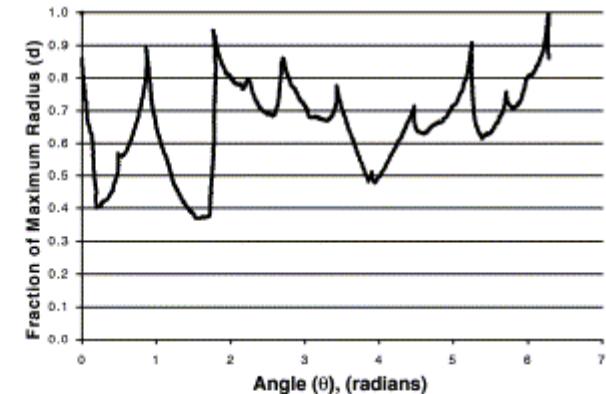
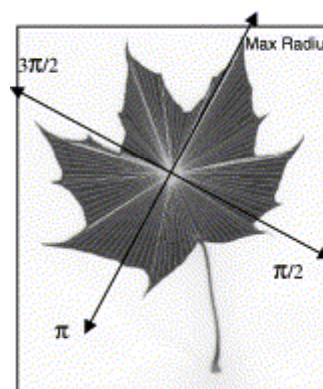
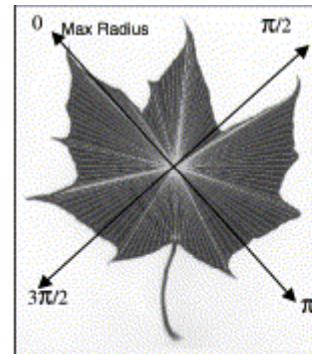
Aplicações na Indústria Petrolífera



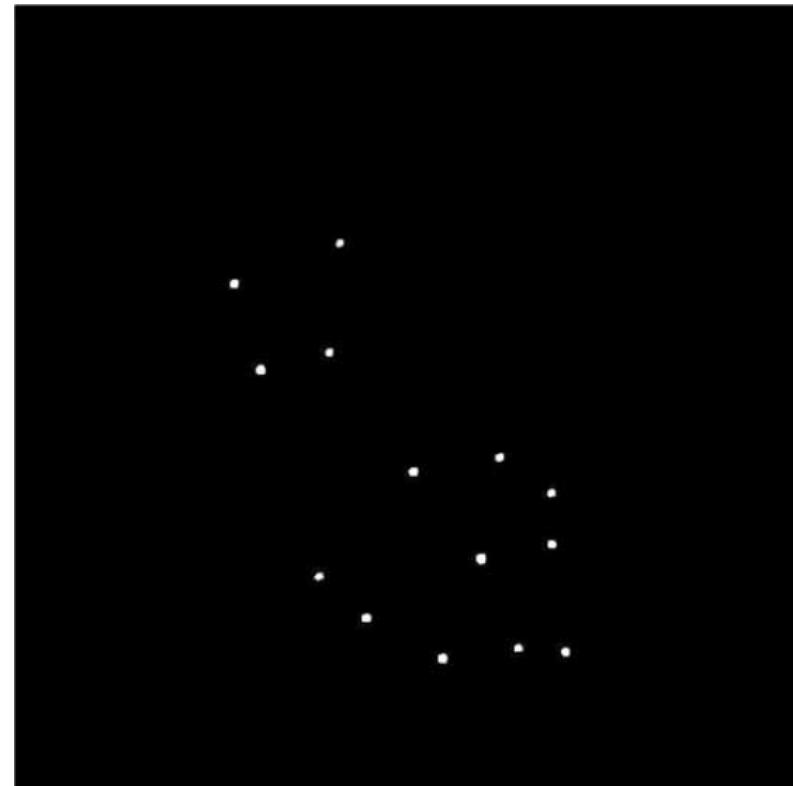
Aplicações na Agropecuária

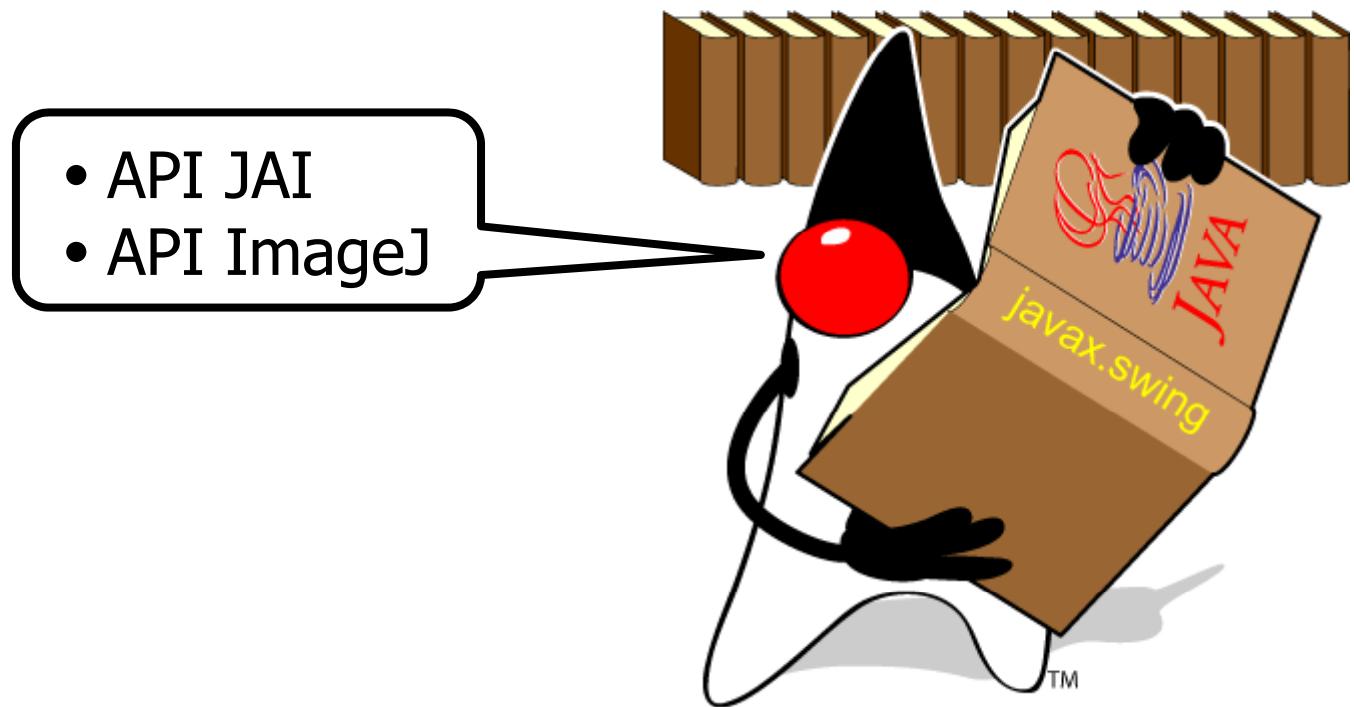


Aplicações na Biologia



Aplicações em Imagens SAR



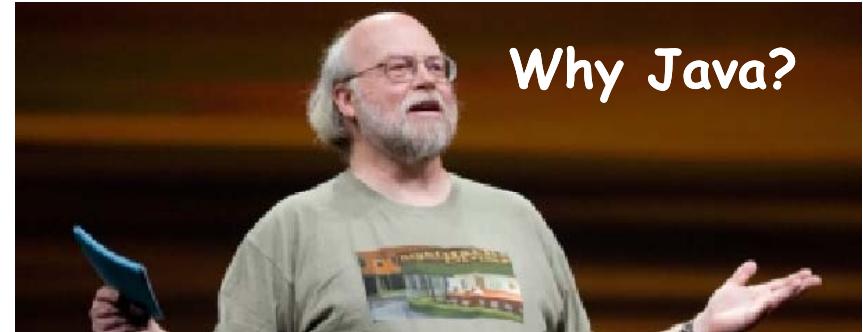


2. APIS JAVA PARA IMAGEM

Formatos de Imagens

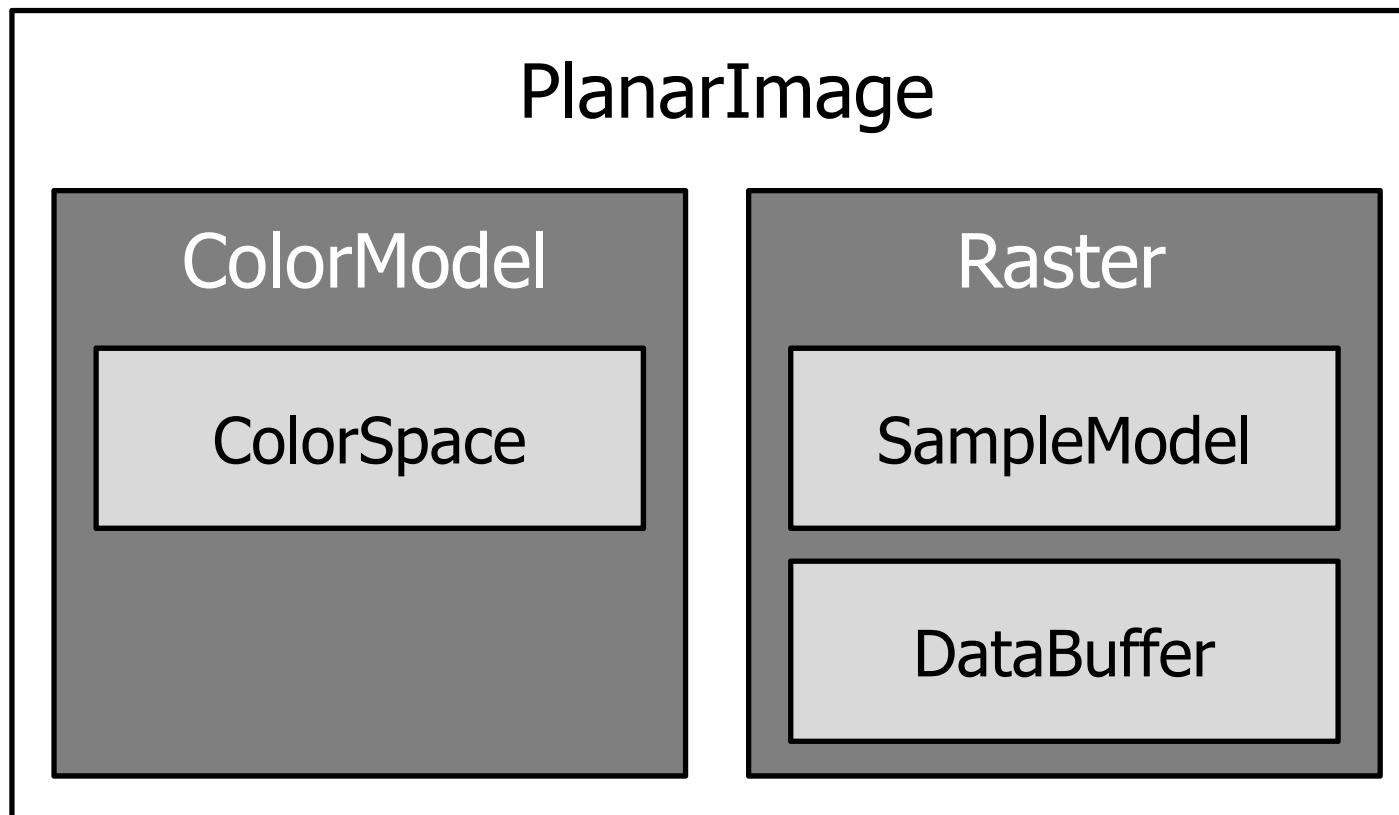
- TIFF, GIF, PNG, JPEG, BMP, PBM, PGM, PMN
- Padronizados nos anos 80 e 90.
- Suportados pela maioria das APIs e linguagens de programação.

Uso da linguagem Java



- Atrativos:
 - Livre;
 - Portável (“write once, run anywhere”);
 - Sintaxe similar à linguagem C;
 - Possui facilidade de internacionalização dos caracteres;
 - Vasta documentação;
 - Coletor de lixo (para desalocação automática de memória);
 - Facilidade de criação de programação distribuída e concorrente;
 - Paradigma Orientado a Objetos.

Java Advanced Imaging (JAI) API



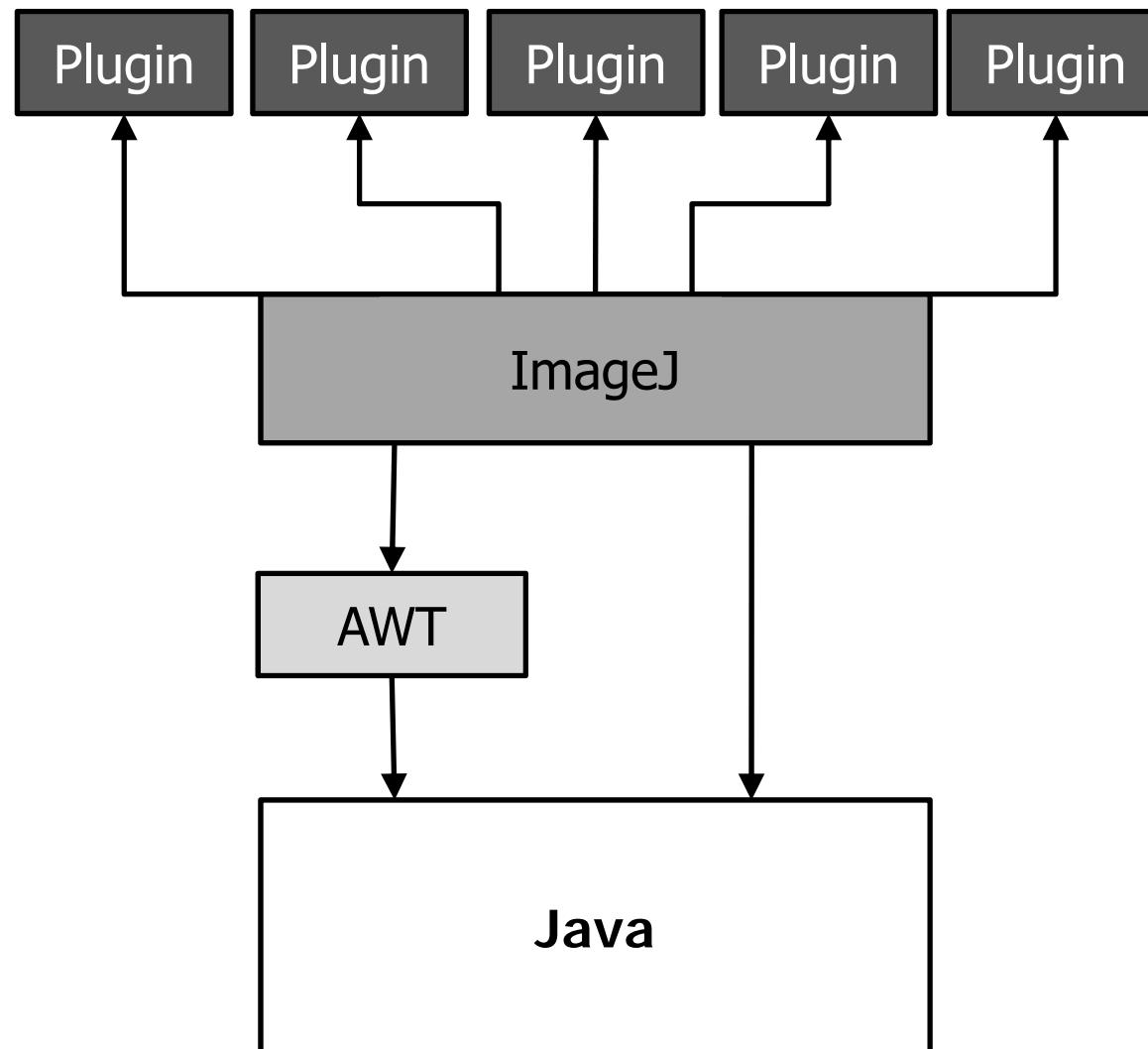
Estrutura da classe de suporte a imagem na API JAI. Adaptado de Santos (2004).

Java Advanced Imaging (JAI) API

- Leitura/Escrita de Imagens

```
public class EScomJAI {  
    public static void main(String[] args) {  
        // leitura da imagem  
        PlanarImage image = JAI.create("fileload", "Lenna.png");  
        double[][] bandCombine = {{0.21f,0.71f,0.08f,0.0f}};  
        // conversão do modelo de cores  
        image = JAI.create("BandCombine", image, bandCombine);  
        // escrita do arquivo  
        JAI.create("filestore",image,"lennagl.png","PNG");  
    }  
}
```

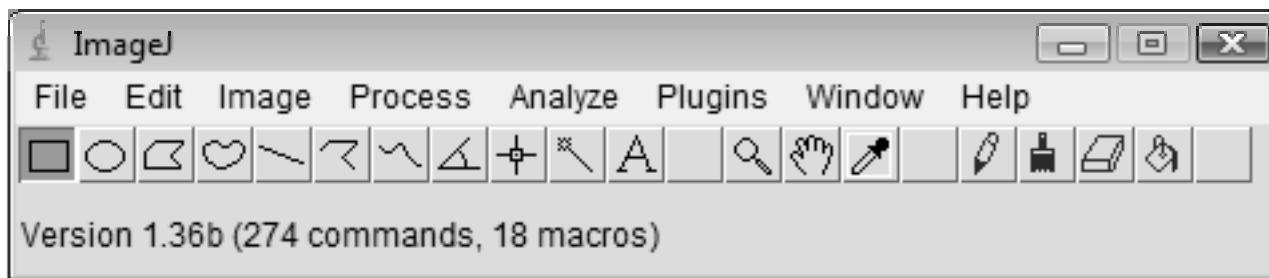
ImageJ API



Estrutura geral da API ImageJ. Adaptado de Burger & Burge (2009).

ImageJ API Software

- Uso de plugins expande a comunidade de desenvolvedores;
- Disponibiliza um software com a execução de todas as funcionalidades implementadas;



Listagem: ExibeImagem.java

```
public class ExibeImagem extends JFrame {  
    public ExibeImagem() throws IOException { // início do construtor  
        BufferedImage imagem = ImageIO.read(new File("Lenna.png"));  
        String infoImagem = "Dimensões: "+imagem.getWidth()  
            +"x"+imagem.getHeight()+" Bandas: "+ imagem.getRaster().getNumBands();  
        ImageIcon icone = new ImageIcon(imagem);  
        JLabel labImagem = new JLabel(icone);  
        (...) // adiciona labImagem e infoImage à interface  
        this.setVisible(true);  
    } // fim do construtor  
    // Método principal  
    public static void main(String args[]){  
        try{ // tratamento de exceção de E/S  
            ExibeImagem appExibeImg = new ExibeImagem();  
        }catch(IOException exc){  
            System.out.println("Erro de leitura! "+exc.getMessage());  
        }  
    } // fim do método principal  
} // fim da classe
```

Resultado: ExibeImagem.java



Imagen Original. Fonte: Imagem adaptada de <http://www.lenna.org/>.



Imagen exibida pela classe.

- Tipos de Imagens
- Realce
- Segmentação
- Morfologia Matemática



3. TÉCNICAS DE PROCESSAMENTO DIGITAL DE IMAGENS

Técnicas de Base para Aplicativo

- **Realce – Negativo e Filtragem (Passa-Baixa e Passa-Alta)**
- **Segmentação – Limiarização e Watershed**
- **Morfologia Matemática – Erosão, Dilatação, Abertura e Fechamento**

Tipos de Imagens

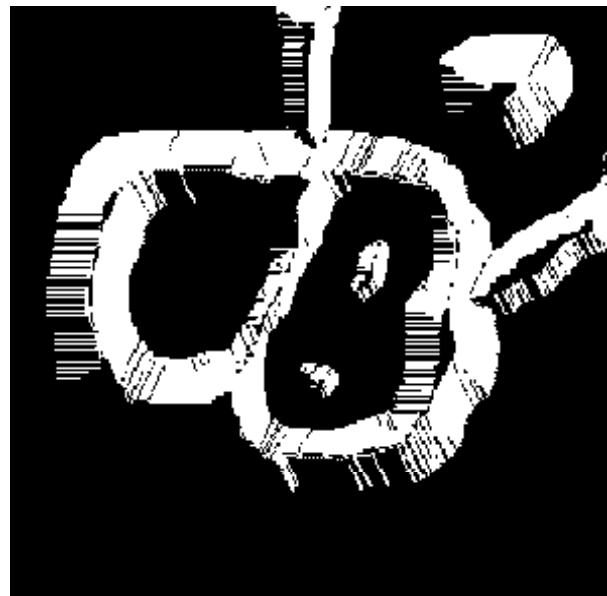


Imagen Binária

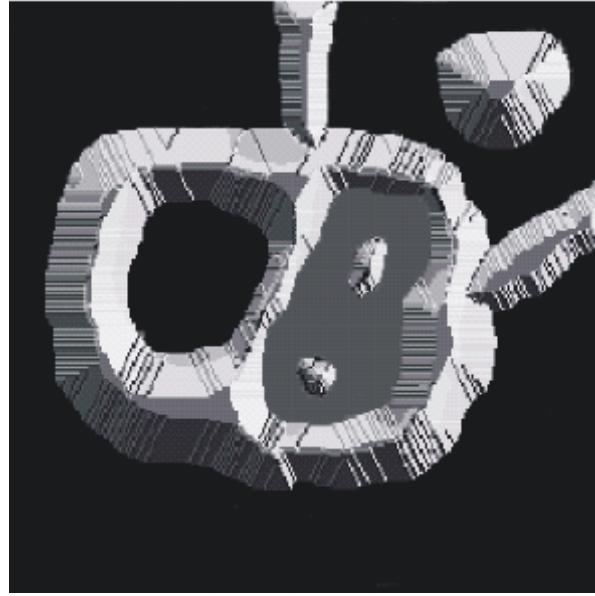


Imagen em
Níveis de Cinza

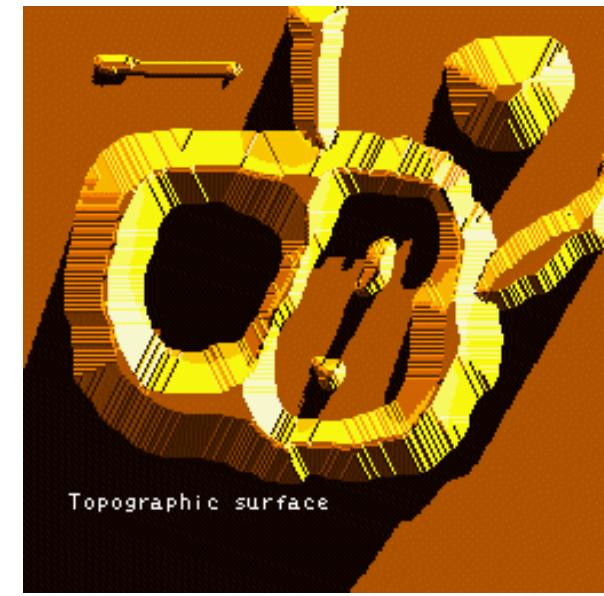
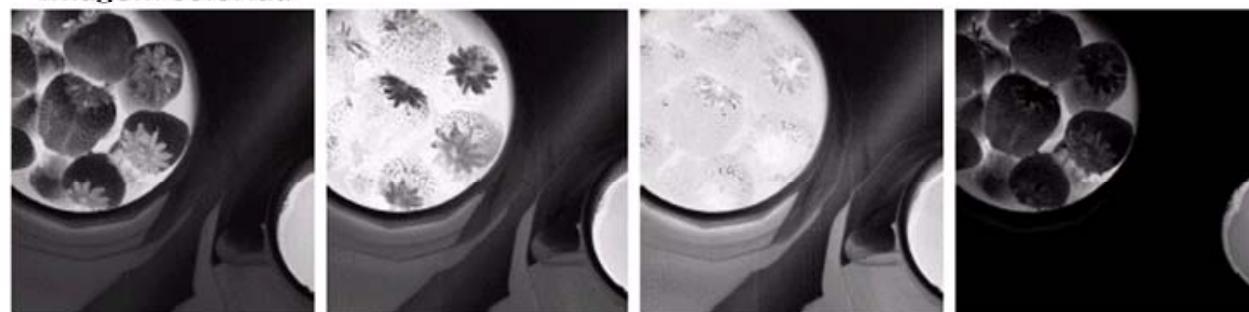


Imagen Colorida



Imagen Colorida

CMYK



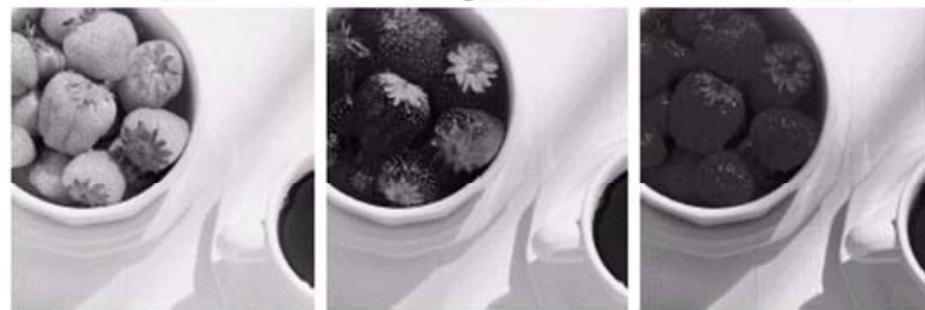
Ciano

Magenta

Amarelo

Preto

RGB



Vermelho

Verde

Azul

HSV



Matiz

Saturação

Intensidade

```
double[][] bandCombine = { {R, G, B, 0.0f} };  
image = JAI.create("BandCombine", image, bandCombine);
```



R = 1.0f
G = 0.0f
B = 0.0f



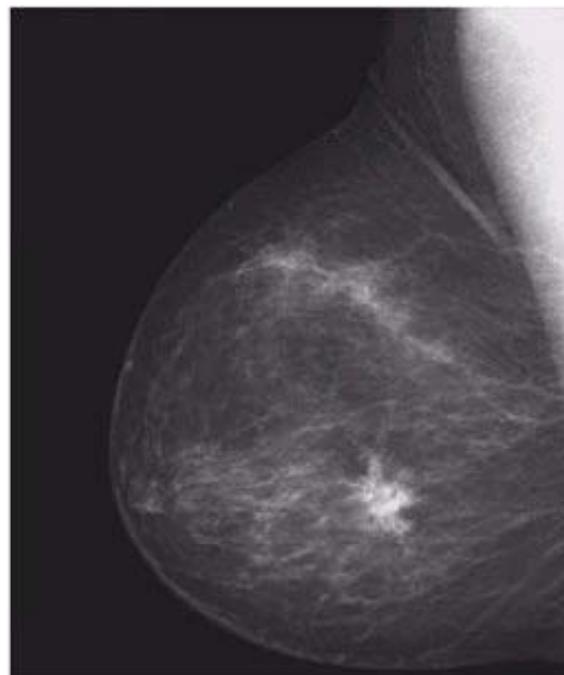
R = 0.0f
G = 1.0f
B = 0.0f



R = 0.0f
G = 0.0f
B = 1.0f

Negativo de uma Imagem

- Negativo de uma Imagem
 - Realce com processamento ponto-a-ponto
 - Aplicações em imagens médicas
 - Reverter a ordem do preto para o branco
 - Intensidade da imagem de saída diminui
 - Intensidade da entrada aumenta
 - $s = L - 1 - r$



Negativo de uma Imagem

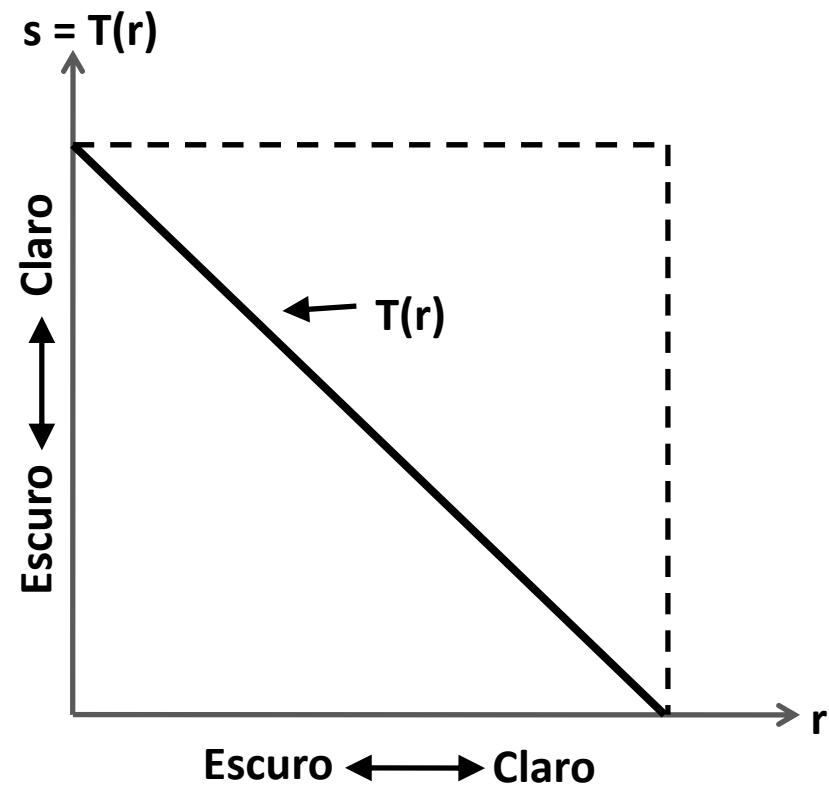


Imagen Original



Imagen Processada

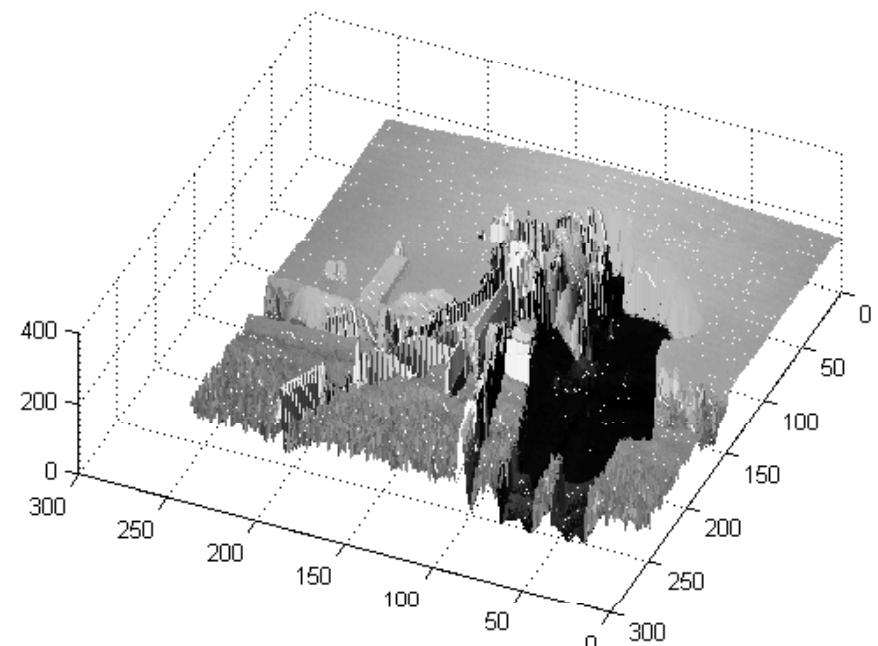
Presença de Ruído em Imagens

- Distorções em uma imagem afetam a percepção da sua real informação;
- Uma distorção crítica é a presença de ruído;
 - Implica em nova informação na imagem;
- Normalmente os ruídos são definidos por uma distribuição estatística:
 - modelo gaussiano, speckle, poisson, etc.

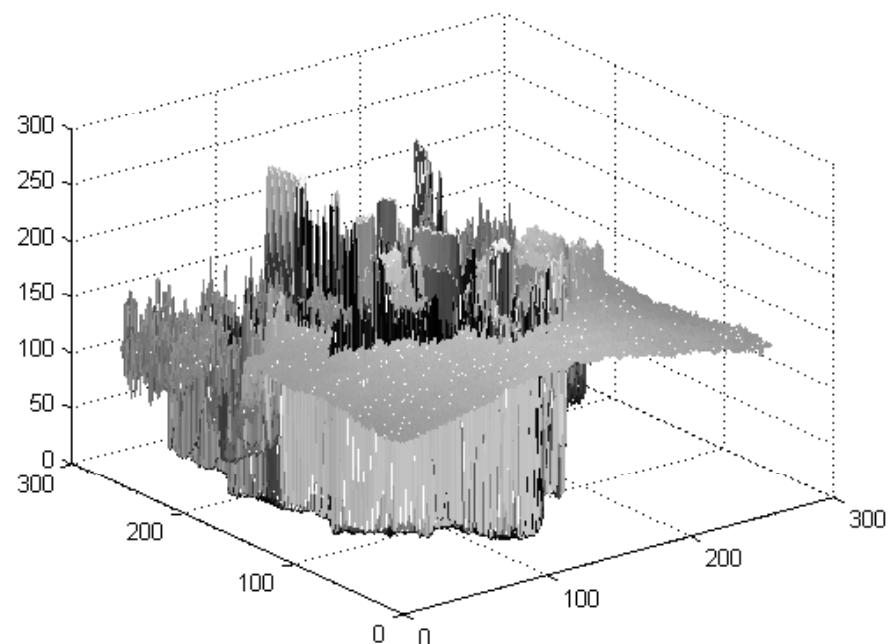
Efeito do Ruído



Imagen Original



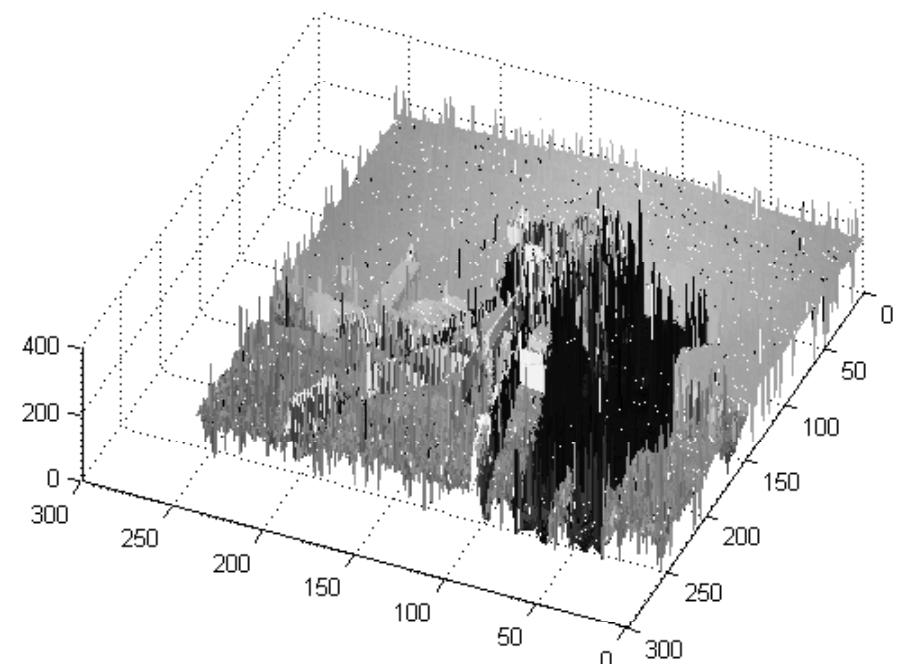
Mudanças de Perspectivas



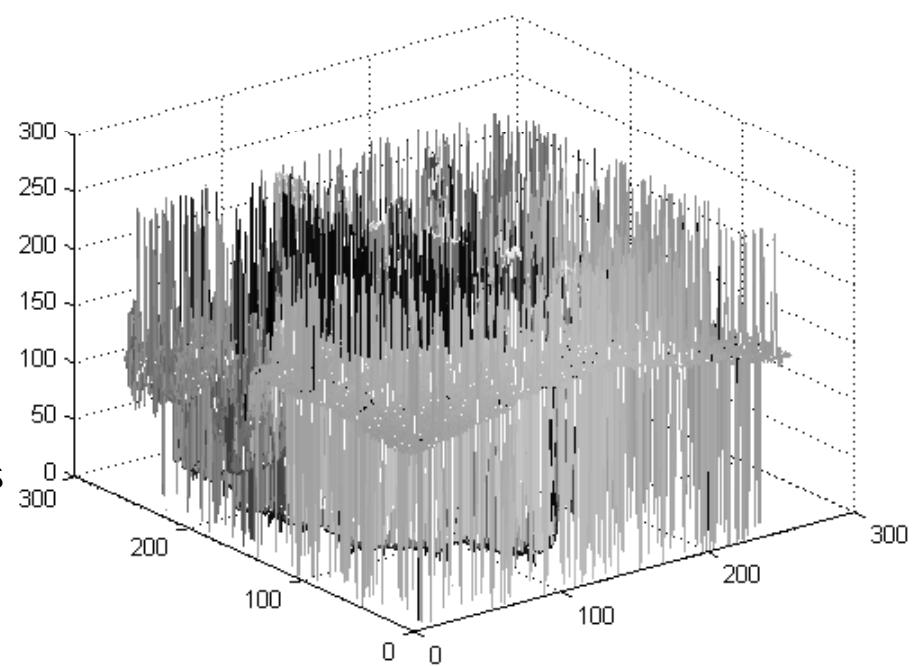
Efeito do Ruído



**Imagen Ruidosa
“Sal e Pimenta”**



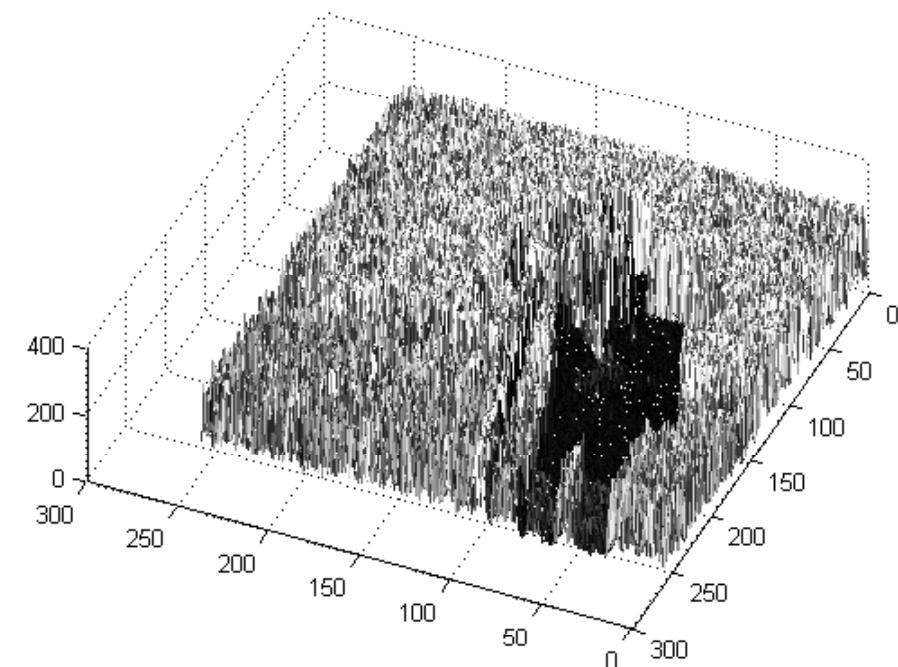
Mudanças de Perspectivas



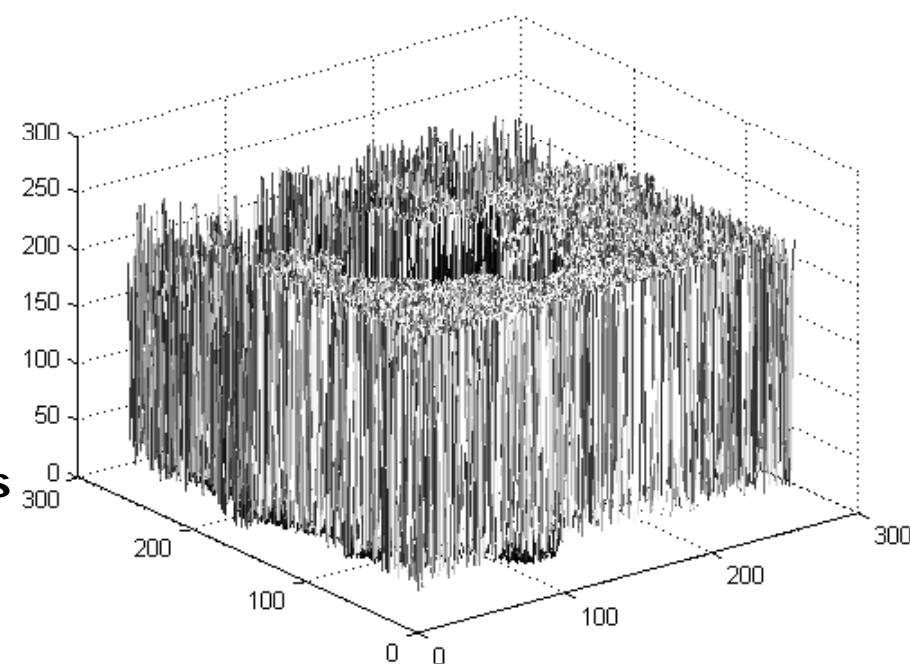
Efeito do Ruído



Imagen Ruidosa
Speckle



Mudanças de Perspectivas



Efeito do Ruído



Imagen Original



Presença de ruído Poisson.

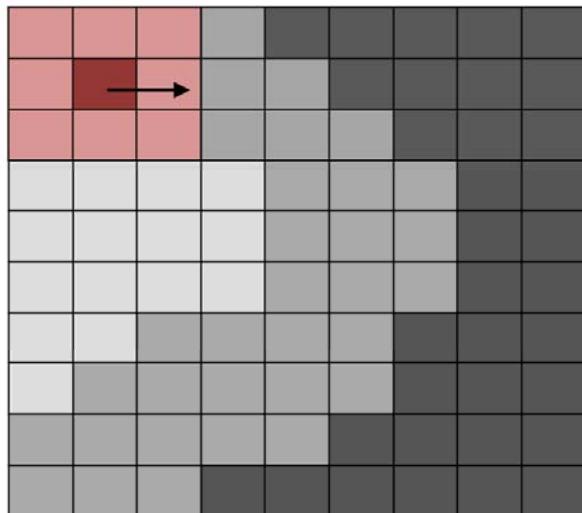


Presença de ruído
"Sal e pimenta".

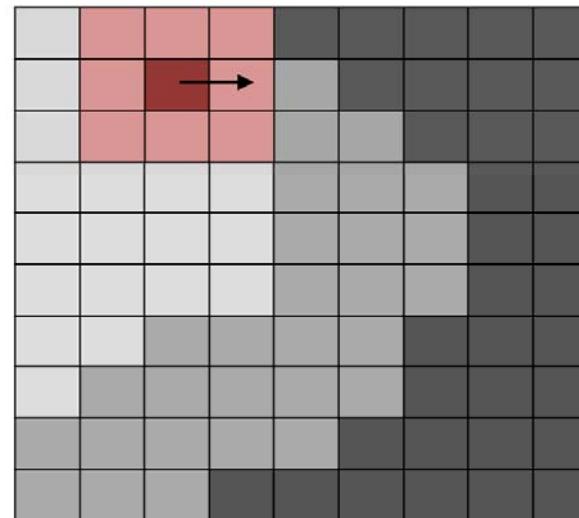


Presença de ruído *Speckle*.

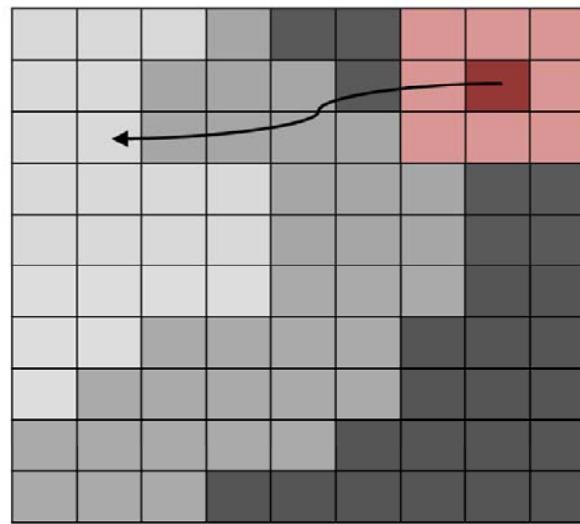
Realce com Processamento por Máscara



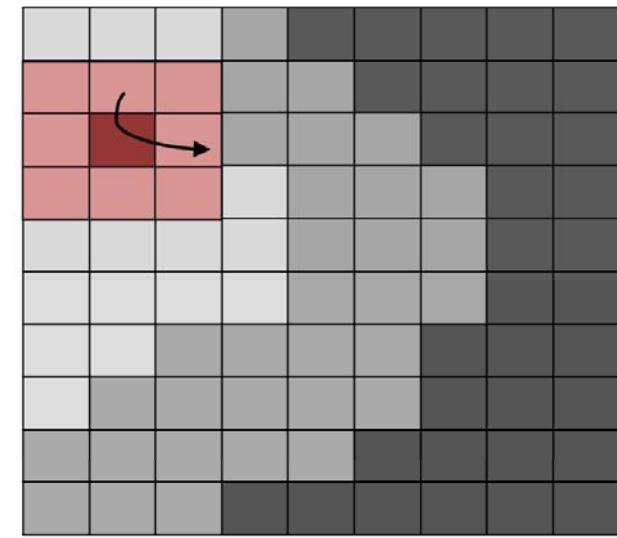
(1)



(2)



(3)



(4)

Realce com Processamento por Máscara

Exemplo de Máscara (3x3)

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

- $g(x,y) = T[f(x,y)]$ (T opera em uma vizinhança de pixels)
- $z'_5 = R = w_1 z_1 + w_2 z_2 + \dots + w_9 z_9$ (Filtro Linear)
 - z'_5 : soma ponderada de pixels na vizinhança de z_5
- $z_5 = \max(z_k, k = 1, 2, \dots, 9)$ (Filtro Não-Linear)
 - Não satisfaz a condição de um filtro linear

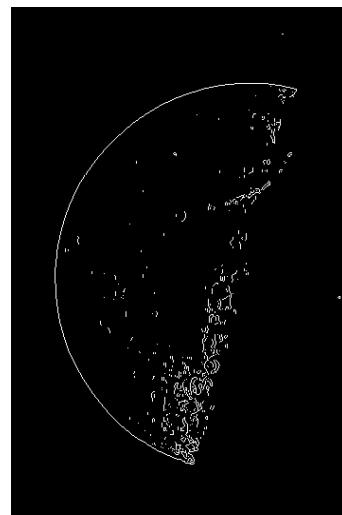
Tipos de Filtragem

- A distribuição de valores na máscara define o tipo de filtragem:
 - **Passa-alta**
 - Aguçamento das bordas;
 - **Passa-baixa**
 - Supressão de ruído, borramento da imagem;
 - **Passa-banda**
 - Restauração da imagem.

Detectores de Bordas (Gradiente – Passa-Alta)



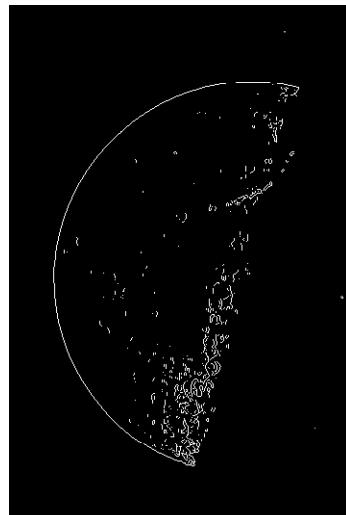
Imagen Original



Filtro de Prewitt

-1	-1	-1
0	0	0
1	1	1

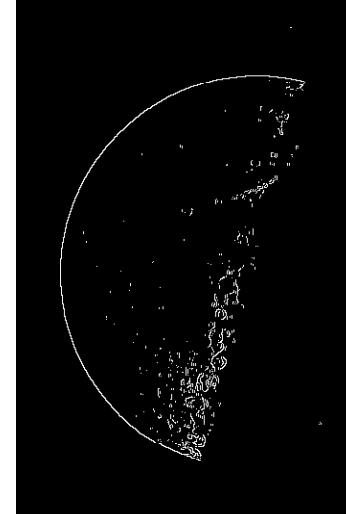
-1	0	1
-1	0	1
-1	0	1



-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Filtro de Sobel

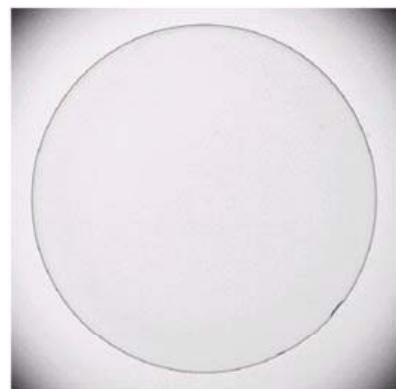


1	0
0	-1

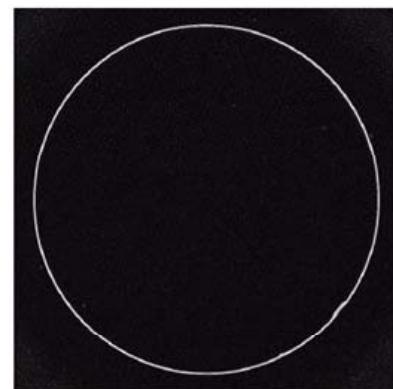
0	1
-1	0

Filtro de Roberts

Definição de Gradiente

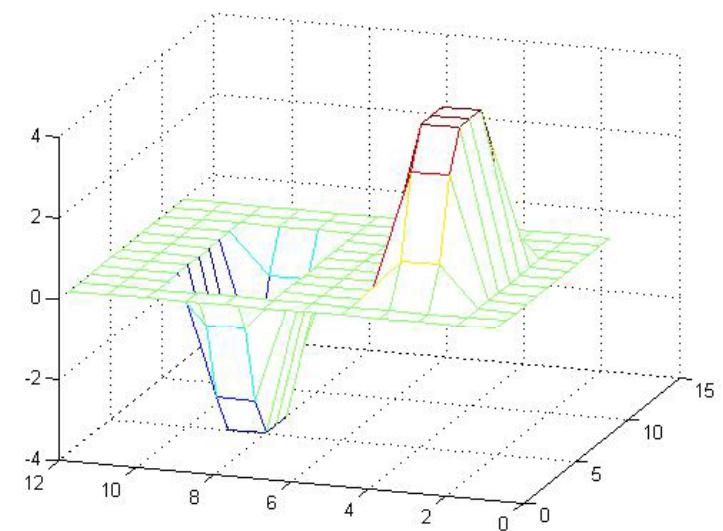


(a)



(b)

Figura – (a) Imagem original; (b) Imagem após aplicação do filtro de Sobel.



```
s = [1 2 1; 0 0 0; -1 -2 -1];  
A = zeros(10);  
A(3:7,3:7) = ones(5);  
H = conv2(A,s);  
mesh(H)
```

Filtro de Suavização (passa-baixa)

- São úteis para redução de ruído e borramento de imagens
 - Detalhes são perdidos
 - O tamanho da máscara (ou seja, vizinhança) determina o grau de suavização e perda de detalhes.
 - Os elementos da máscara devem ser números positivos
 - Exemplo: Média Local

$$\frac{1}{9} \times \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$
$$\frac{1}{16} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

– $R = 1/9 (1z_1 + 1z_2 + \dots + 1z_9)$

normalizado: dividido por 9

Filtro por mediana (não-linear)

- Substitua $f(x,y)$ pela mediana $[f(x',y')]$ onde (x',y') pertence a vizinhança
- Muito efetivo na remoção do ruído com componentes do tipo espiada (“spike”, ocorrências aleatórias de valores brancos e pretos)
- Preserva melhor as bordas
- Exemplo:

10	20	20
20	15	20
25	20	100

→ Ordenar
(10,15,20,20,20,20,20,25,100)

- Mediana = 20, então substitua (15) por (20)

Filtragem da Mediana



Imagen Ruidosa
"Sal e Pimenta"



Filtragem com Janela 3x3



Filtragem com Janela 5x5



Filtragem com Janela 7x7

Limiarização

- Comprime a faixa de nível de cinza de pouco interesse
 - Inclinação da linha entre [0,1]
- Alargamento da faixa de nível de cinza de maior interesse
 - Inclinação da linha monotonicamente crescente
 - $r_1 = r_2$
 - $s_1 = 0$ e $s_2 = L - 1$

Limiarização

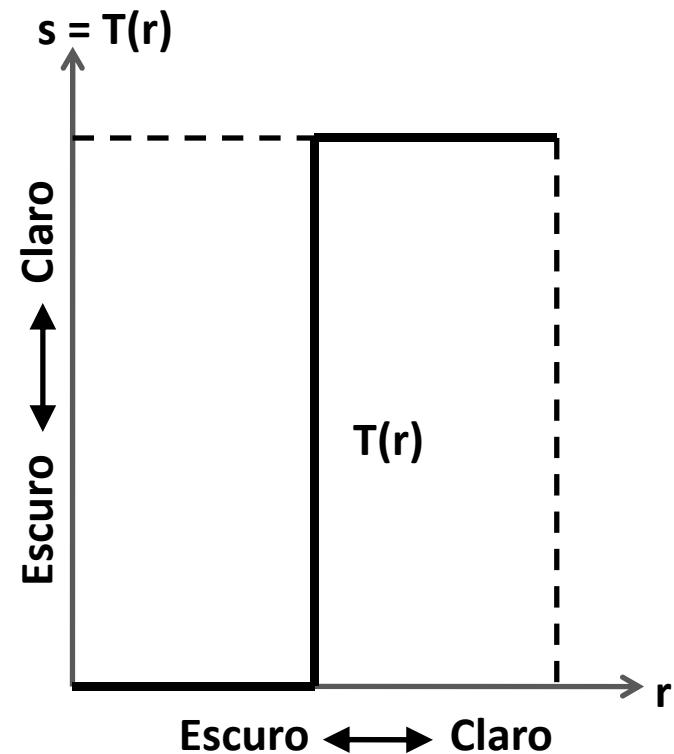


Imagen Original



Imagen Processada

Limiarização



Imagen Original



Limiar de 130



Limiar de 140



Limiar de 150

Watershed

- Considerando uma imagem em tom de cinza como uma superfície topográfica:
 - Todos os vales foram perfurados na sua área mais funda da superfície;
 - Toda a superfície é lentamente preenchida por água, este irá progressivamente inundar todas as bacias da imagem;
 - Represas podem ser aumentadas onde a água vem de dois pontos distintos, podendo se unir;
 - Ao fim da inundação, cada bacia é cercada por cumes (represas) representando seu limite.
 - Este processo define a divisão das regiões presentes na imagem.

Watershed

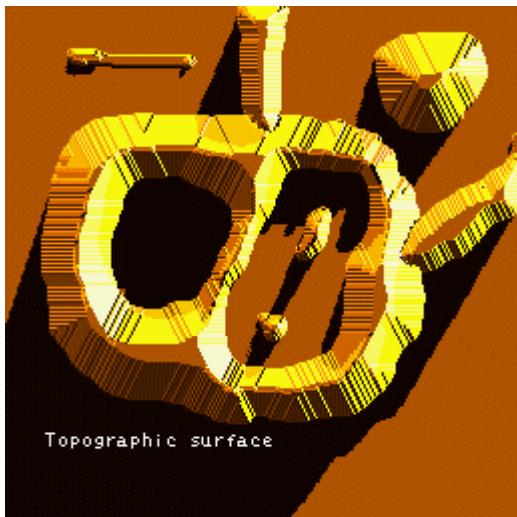


Imagen Original

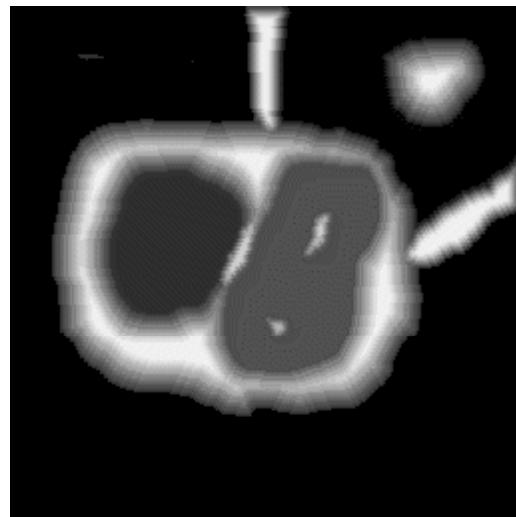


Imagen em Níveis de Cinza

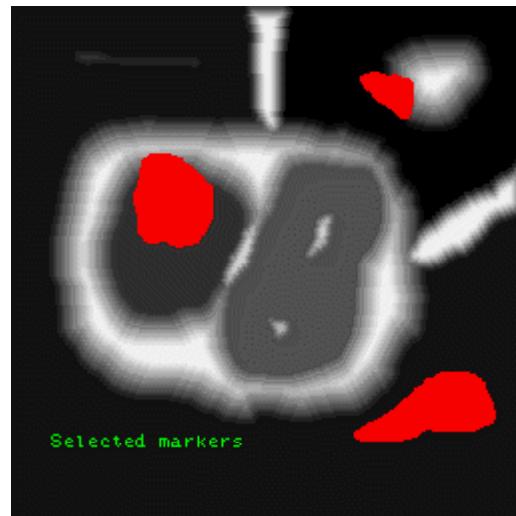


Imagen com Marcadores

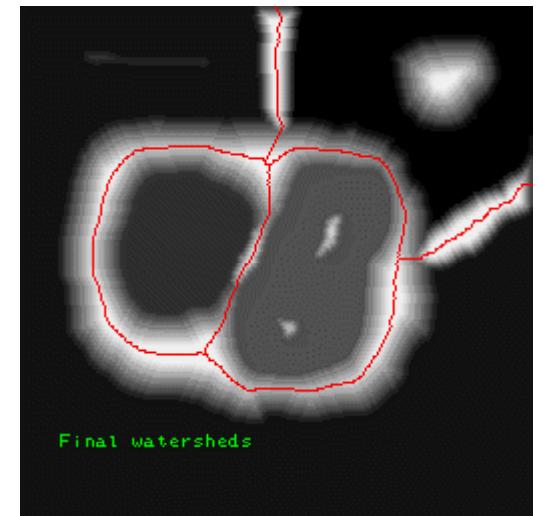
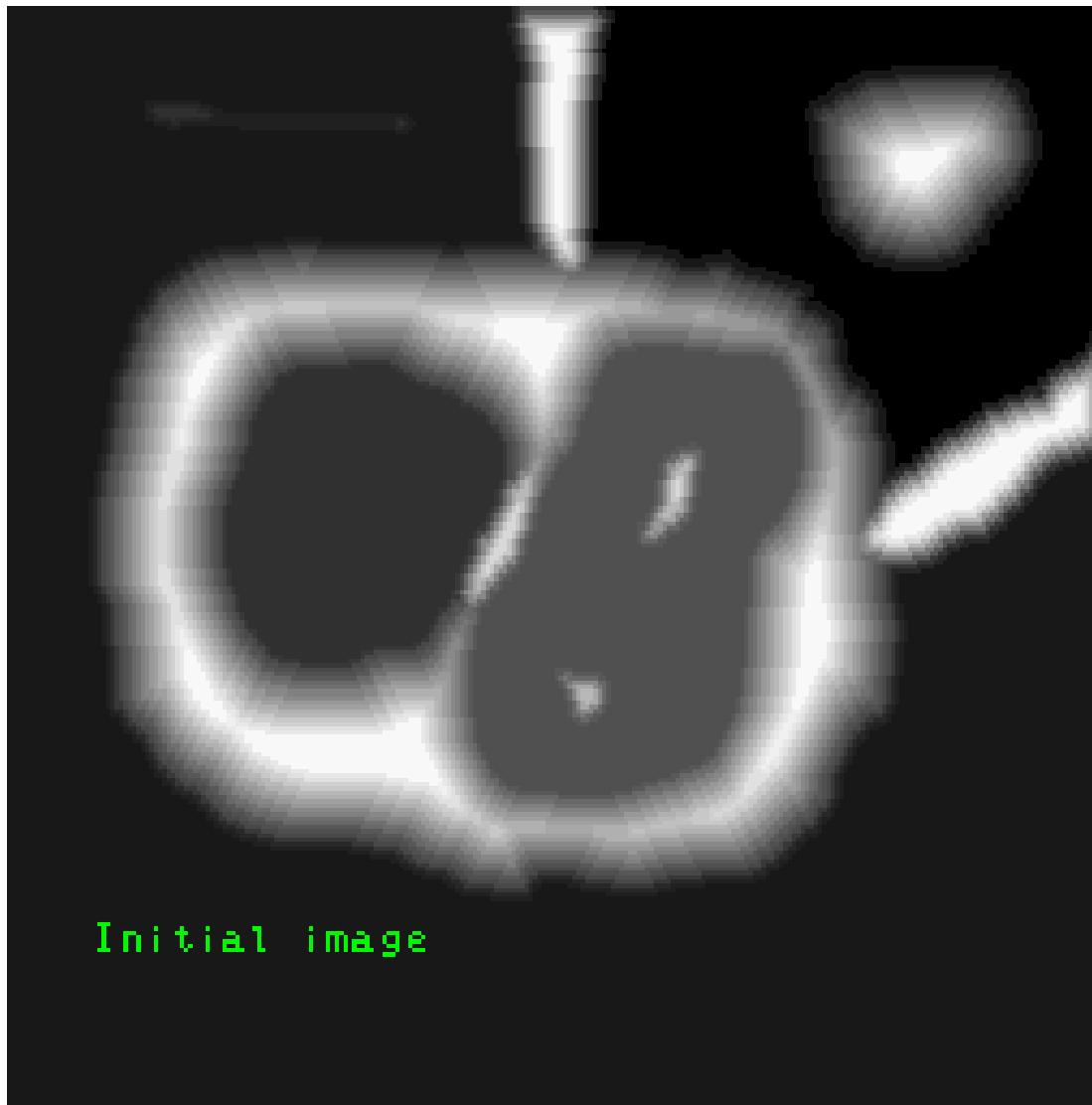


Imagen Final

Watershed



Initial image

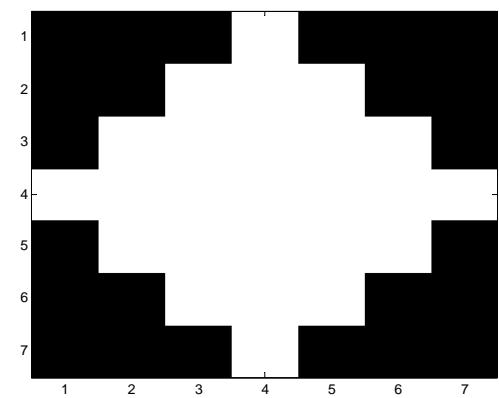
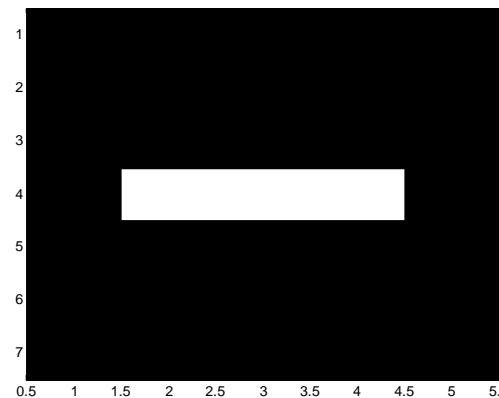
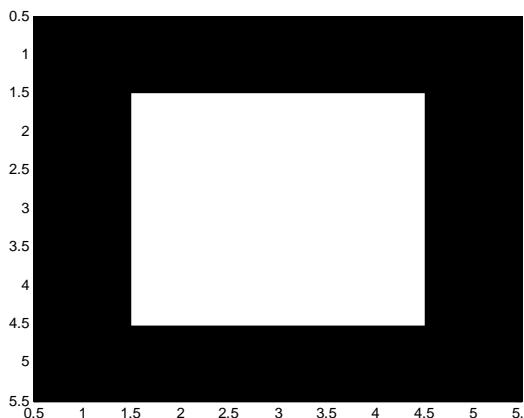
Animação sobre Watershed. Fonte: <http://cmm.ensmp.fr/~beucher/wtshed.html>.

Watershed

- Os objetos das imagens são definidos como as regiões de muitos valores de escala de cinza constantes.
- O operador gradiente irá realçar os limites dos objetos (fim da inundaçāo).
- Os marcadores indicam, de forma bruta, onde os objetos estão localizados na imagem (ponto de partida da inundaçāo).

Morfologia Matemática

- Elemento Estruturante (EE):
 - *Structuring Element (SE)*
 - Pequeno conjunto usado para reconhecer a morfologia do objeto de interesse em uma imagem.
 - Forma e tamanho devem ser adaptados para as propriedades geométricas da imagem processada.

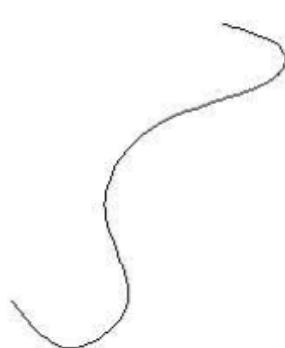


Morfologia Matemática

- Erosão e Dilatação
- A partir destes: Abertura e Fechamento

$$\delta_B(X) = \bigcup_{x \in X} B_x.$$

$$\varepsilon_B(X) = \{x \mid B_x \subseteq X\}.$$

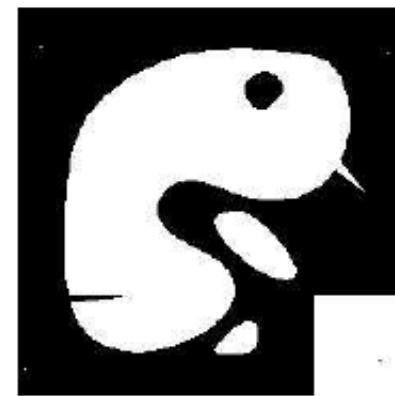


X



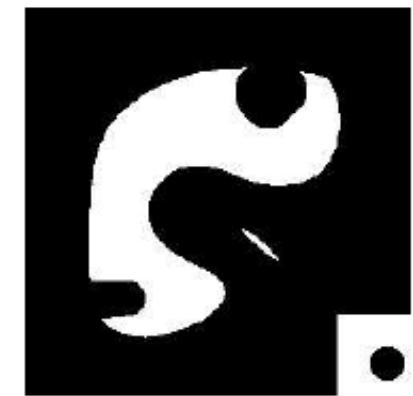
B

$\delta_B(X)$



X

B



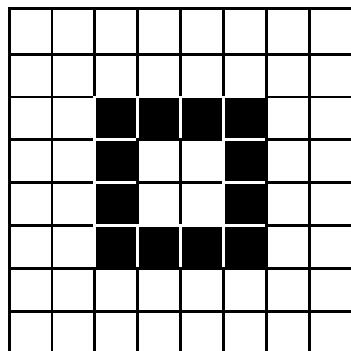
$\varepsilon_B(X)$

Morfologia Matemática

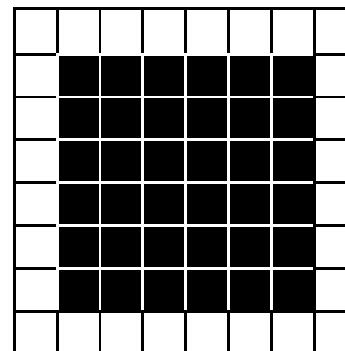
- Erosão não é o inverso da dilatação!
 - Uma erosão seguida de uma dilatação nem sempre retorna a imagem original.

$$\varepsilon_B(\delta_B(X)) \neq \delta_B(\varepsilon_B(X)) \neq X$$

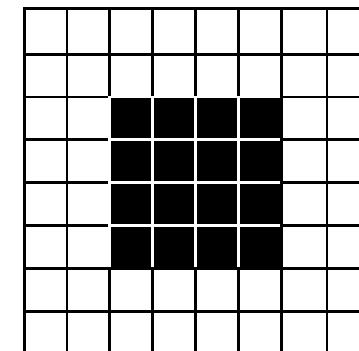
- B → quadrado 3x3



X



$\delta_B(X)$



$\varepsilon_B((\delta_B(X)))$

Morfologia Matemática

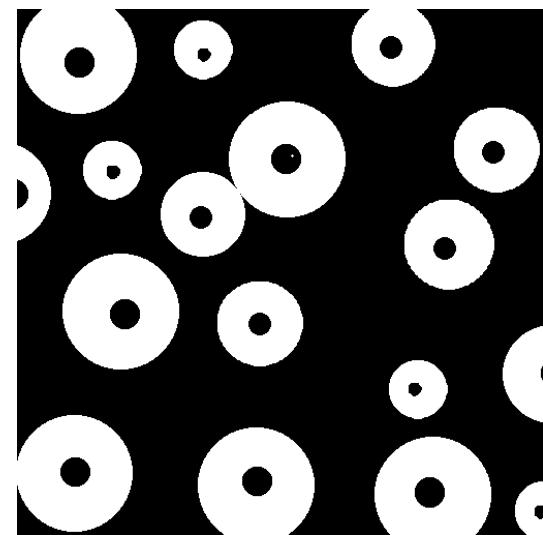
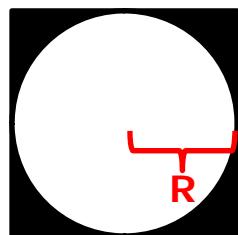
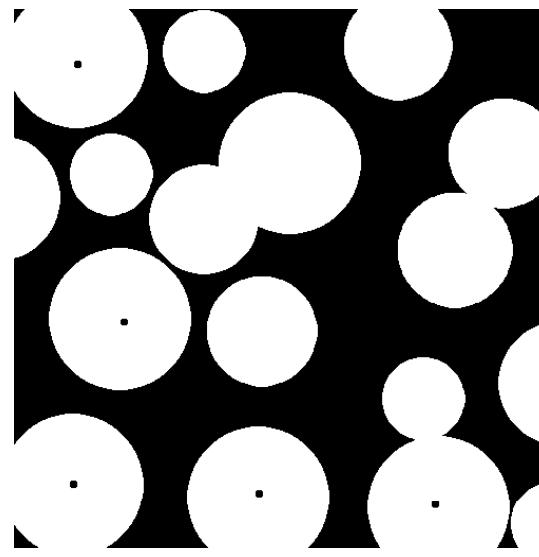


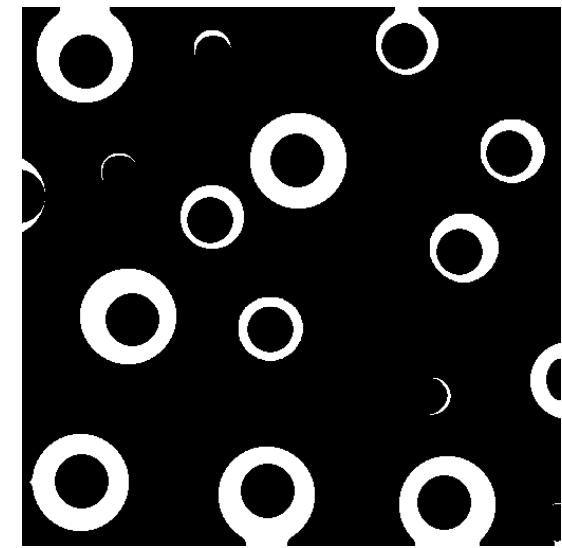
Imagen Original



Elemento Estruturante
Raio = 11



Resultado da Dilatação



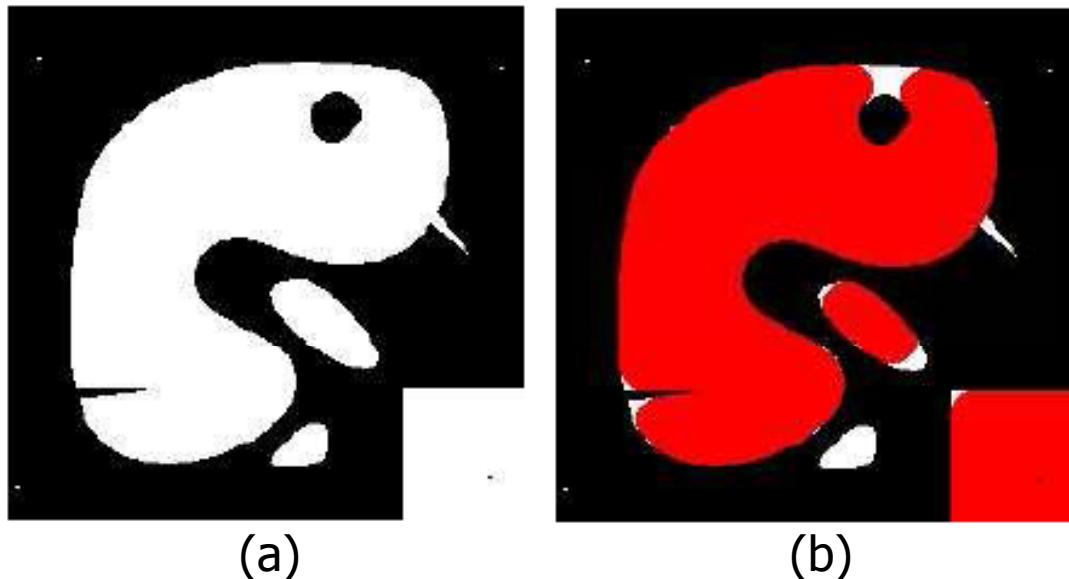
Resultado da Erosão

Morfologia Matemática

- Abertura
 - Resultado da erosão seguida da dilatação;
 - Usando o mesmo elemento estruturante.

- $\gamma_B(X) = \delta_B(\varepsilon_B(X))$

- Propriedades:
 - Idempotente;
 - Crescente;
 - Anti-extensiva; ?



Figura—Abertura com EE circular: (a) imagem original e (b) sobreposição do resultado.

Resultado da abertura é sempre menor ou igual à imagem original.

Morfologia Matemática

- Fechamento
 - Dual da Abertura;
 - Resultado da dilatação seguida da erosão
 - Usando o mesmo elemento estruturante.
 - $\varphi_B(X) = \varepsilon_B(\delta_B(X))$
 - Propriedades:
 - Idempotente;
 - Crescente;
 - Extensiva; ?

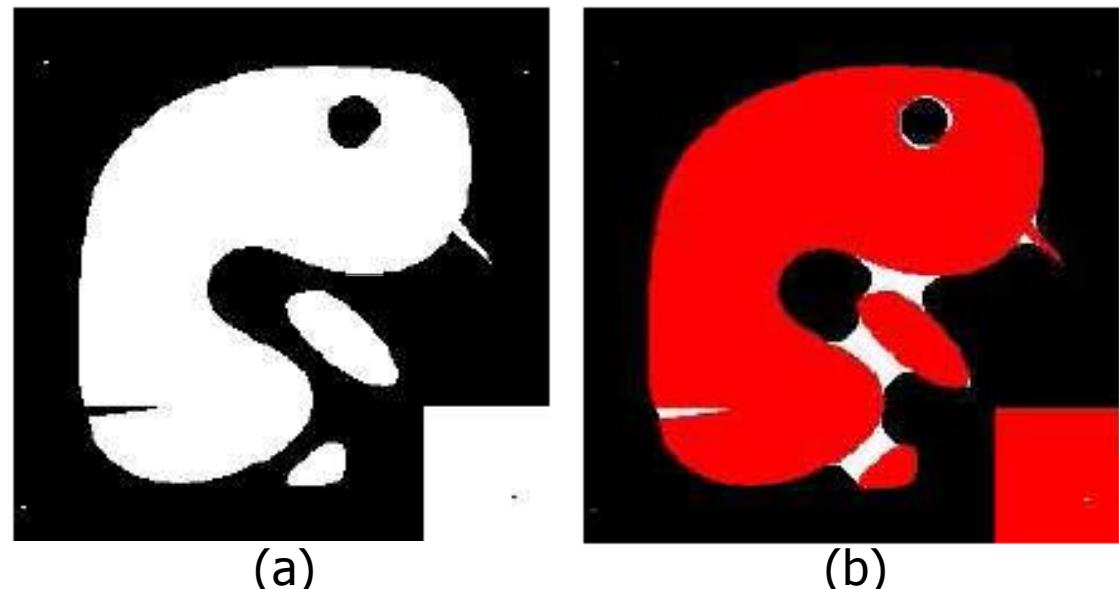


Figura – Fechamento com EE circular:
(a) imagem original e (b) sobreposição do resultado.

Resultado da abertura é sempre maior ou igual à imagem original.

Morfologia Matemática

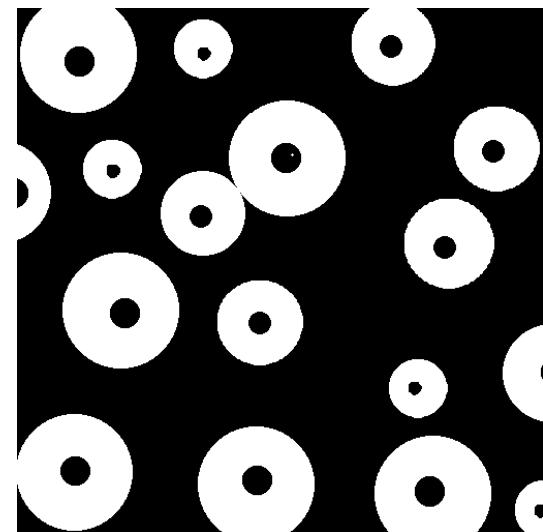
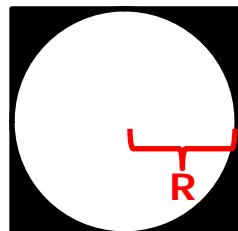
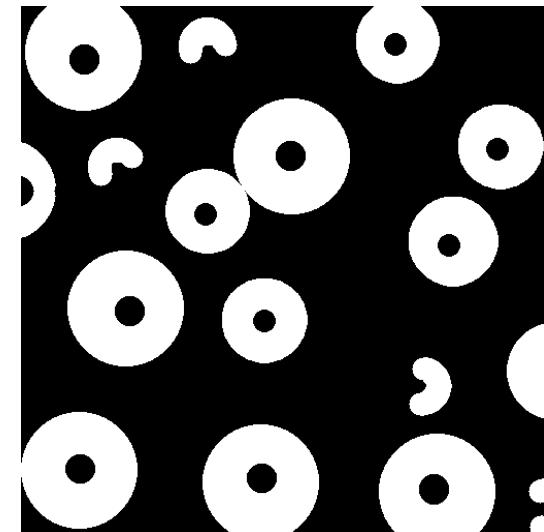


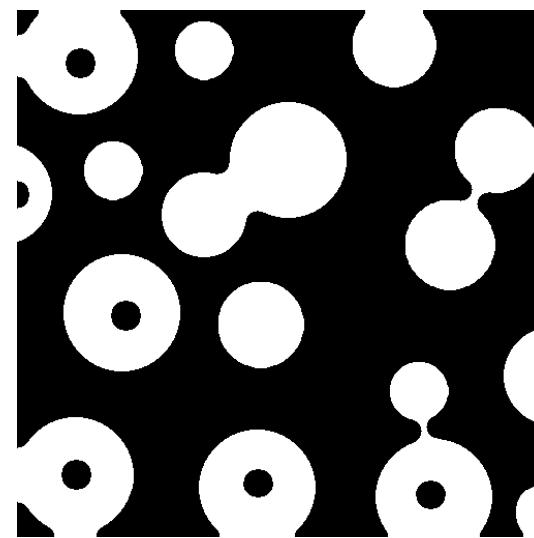
Imagen Original



Elemento Estruturante
Raio = 11



Resultado da Abertura



Resultado do Fechamento

Morfologia Matemática



Imagen Original



Resultado da Erosão



Resultado da Dilatação



Resultado da Abertura



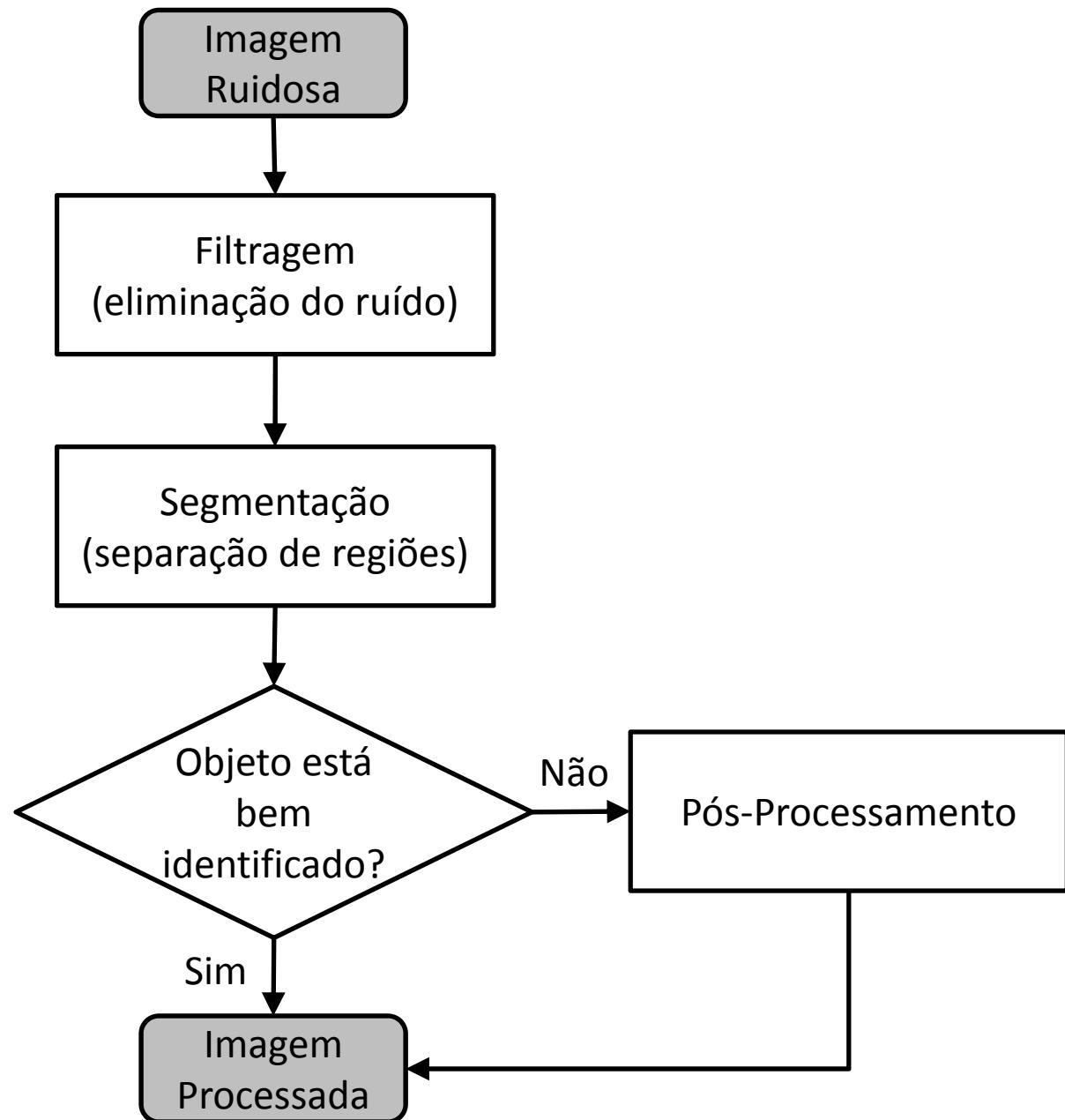
Resultado do Fechamento



- Definição do aplicativo
- Uso da API ImageJ
- Desenvolvimento

4. PROPOSTA DE UM APLICATIVO DE PDI

Fluxograma



Aplicando Filtro da Mediana

```
// objeto "imagem" da classe java.awt.Image instanciado  
// com a imagem original  
ByteProcessor processador = new ByteProcessor(imagem);  
processador.medianFilter();  
// resultado do filtro é passado ao objeto "imagem"  
imagem = processador.createImage();
```

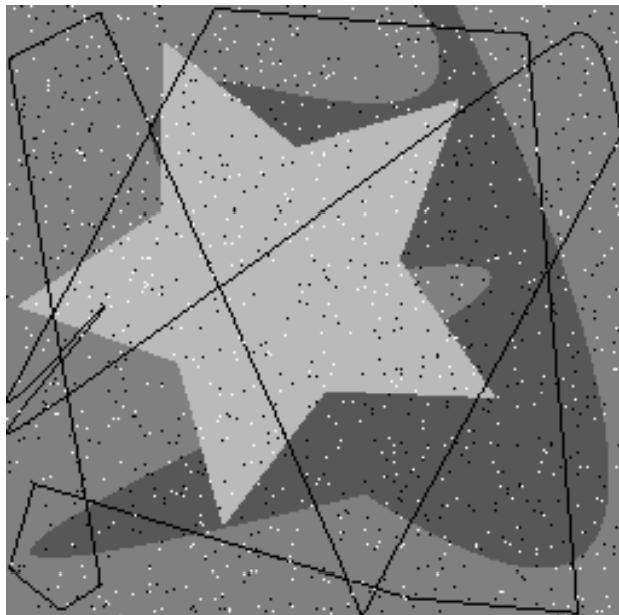


Imagen Ruidosa



Imagen Filtrada

Aplicando Segmentação

```
// objeto "imagem" da classe java.awt.Image instanciado  
// com a imagem filtrada  
ByteProcessor processador = new ByteProcessor(imagem);  
// "valorLimiar" é inteiro e que indica o limiar da segmentação  
processador.threshold(valorLimiar);  
// resultado da limiarização é passado ao objeto "imagem"  
imagem = processador.createImage();
```



Imagen Filtrada



Imagen Segmentada

Aplicando Fechamento

```
// objeto "imagem" da classe Image instanciado com a imagem segmentada  
ByteProcessor processador = new ByteProcessor(imagem);  
// etapa de dilatação - 1º passo do fechamento  
processador.dilate(1,0);  
imagem = processador.createImage();  
processador = new ByteProcessor(imagem);  
// etapa de erosão - 2º passo do fechamento  
processador.erode(1,0);  
imagem = processador.createImage();
```



Imagen Segmentada

Imagen Final

Melhor uso de memória

- Evitar o acúmulo de variáveis estáticas no método principal
 - Para auxiliar o desempenho do coletor de lixo da máquina virtual Java.
- O uso de variáveis estáticas deve ser evitado
 - Ocupam muito recurso de memória
 - São as últimas instâncias a serem eliminadas pelo *garbage collector*

Melhor uso de memória

- Execução do *garbage collector*.
 - System.gc() ou
 - Runtime rt = Runtime.getRuntime();
rt.gc();
long mem = rt.freeMemory();
- Execução frequente de *gc()* pode resultar na degradação do desempenho.

Listagem: SegRdI.java (visão geral)

```
public class SegRdI extends JFrame {  
    // atributos  
    JPanel painelPrinc, painelBotoes;  
    JButton btnAbrir, btnProc, btnLimpar, btnSair;  
    File fileName;      ImagePlus imagemIJ;  
    // inicialização dos componentes gráficos  
    public void iniciaComponentes() { ... }  
    // método da ação ao clicar o botão Abrir  
    private void abrirActionPerformed(ActionEvent evt) { ... }  
    // método da ação ao clicar o botão Processar  
    private void processarActionPerformed(ActionEvent evt) { ... }  
    // método da ação ao clicar o botão Limpar  
    private void limparActionPerformed(ActionEvent evt) { ... }  
    // método da ação ao clicar o botão Sair  
    private void sairActionPerformed(ActionEvent evt) { System.exit(0); }  
    // construtor padrão  
    public SegRdI() { this.iniciaComponentes(); }  
    // método principal  
    public static void main(String[] args) { SegRdI aplicacao = new SegRdI(); }  
} // fim da classe
```

Listagem: SegRdI.java (inicia interface gráfica)

```
public void iniciaComponentes(){
    this.setLayout(new BorderLayout());    painelPrinc = new JPanel();
    painelPrinc.setLayout(new BorderLayout()); this.add(painelPrinc, BorderLayout.CENTER);
    painelBotoes = new JPanel(); painelBotoes.setLayout(new FlowLayout());
    this.add(painelBotoes, BorderLayout.SOUTH);

    // adicionando botões
    btnAbrir = new JButton(); painelBotoes.add(btnAbrir); btnAbrir.setText(" Abrir ... ");
    btnProc = new JButton(); painelBotoes.add(btnProc); btnProc.setText("Processar");
    // -> faz-se o mesmo para os botões "btnLimpar" e "btnSair"

    // configurar ações dos botões
    btnAbrir.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) { abrirActionPerformed(evt); } } );
    btnProc.addActionListener( ... ); // relacionar com o método processarActionPerformed
    btnLimpar.addActionListener( ... ); // relacionar com o método limparActionPerformed
    btnSair.addActionListener( ... ); // relacionar com o método sairActionPerformed

    this.setVisible(true);          this.setSize(450,350);
    this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
} // fim do método initComponents
```

Listagem: SegRdl.java (limpar interface)

```
private void limparActionPerformed(ActionEvent evt) {  
    ImagePlus imp = new ImagePlus();  
    ImageCanvas ic = new ImageCanvas(imp);  
    painelPrinc.removeAll();  
    painelPrinc.add(ic,BorderLayout.CENTER);  
} // fim do método limparActionPerformed
```

Listagem: SegRdl.java (abrir imagem)

```
private void abrirActionPerformed(ActionEvent evt) {
    // exibe caixa de diálogo para abrir arquivo de imagem
    JFileChooser dialogo = new JFileChooser();
    dialogo.setFileSelectionMode(JFileChooser.FILES_ONLY);
    int result = dialogo.showOpenDialog(this);
    if (result == JFileChooser.CANCEL_OPTION) return;
    // recupera arquivo selecionado
    fileName = dialogo.getSelectedFile();
    // exibe erro se inválido
    if (fileName == null || fileName.getName().equals("")) {
        JOptionPane.showMessageDialog(this, "Nome de Arquivo Inválido",
            "Nome de Arquivo Inválido", JOptionPane.ERROR_MESSAGE);
        return; }
    imagemIJ = new ImagePlus(fileName.toString());
    JScrollPane sp = new JScrollPane( JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
        JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
    ImageCanvas ic = new ImageCanvas(imagemIJ); sp.add(ic);
    sp.setSize(imagemIJ.getWidth(), imagemIJ.getHeight());
    painelPrinc.add(sp,BorderLayout.CENTER);
} // fim do método abrirActionPerformed
```

Listagem: SegRdI.java (processamento da imagem - I)

```
private void processarActionPerformed(ActionEvent evt) {  
    Image image = imagemIJ.getImage();  
    ByteProcessor byteProc = new ByteProcessor(image);  
    byteProc.medianFilter();  
    image = byteProc.createImage();  
    ImagePlus imFilt = new ImagePlus("filtragem",image);  
    // descobrindo maior valor de nível de cinza  
    int max = -1;  
    for(int lin = 0; lin < imFilt.getHeight(); lin++){  
        for(int col = 0; col < imFilt.getWidth(); col++){  
            int[] pixels = imFilt.getPixel(col, lin);  
            if (pixels[0]>max) max = pixels[0]; }  
    }  
    image = imFilt.getImage(); byteProc = new ByteProcessor(image);  
    // aplicando a segmentação através de limiarização  
    byteProc.threshold(max-1);  
    image = byteProc.createImage();  
    ImagePlus imSeg = new ImagePlus("segmentacao",image);  
    image = imSeg.getImage(); byteProc = new ByteProcessor(image);
```

Listagem: SegRdI.java (processamento da imagem - II)

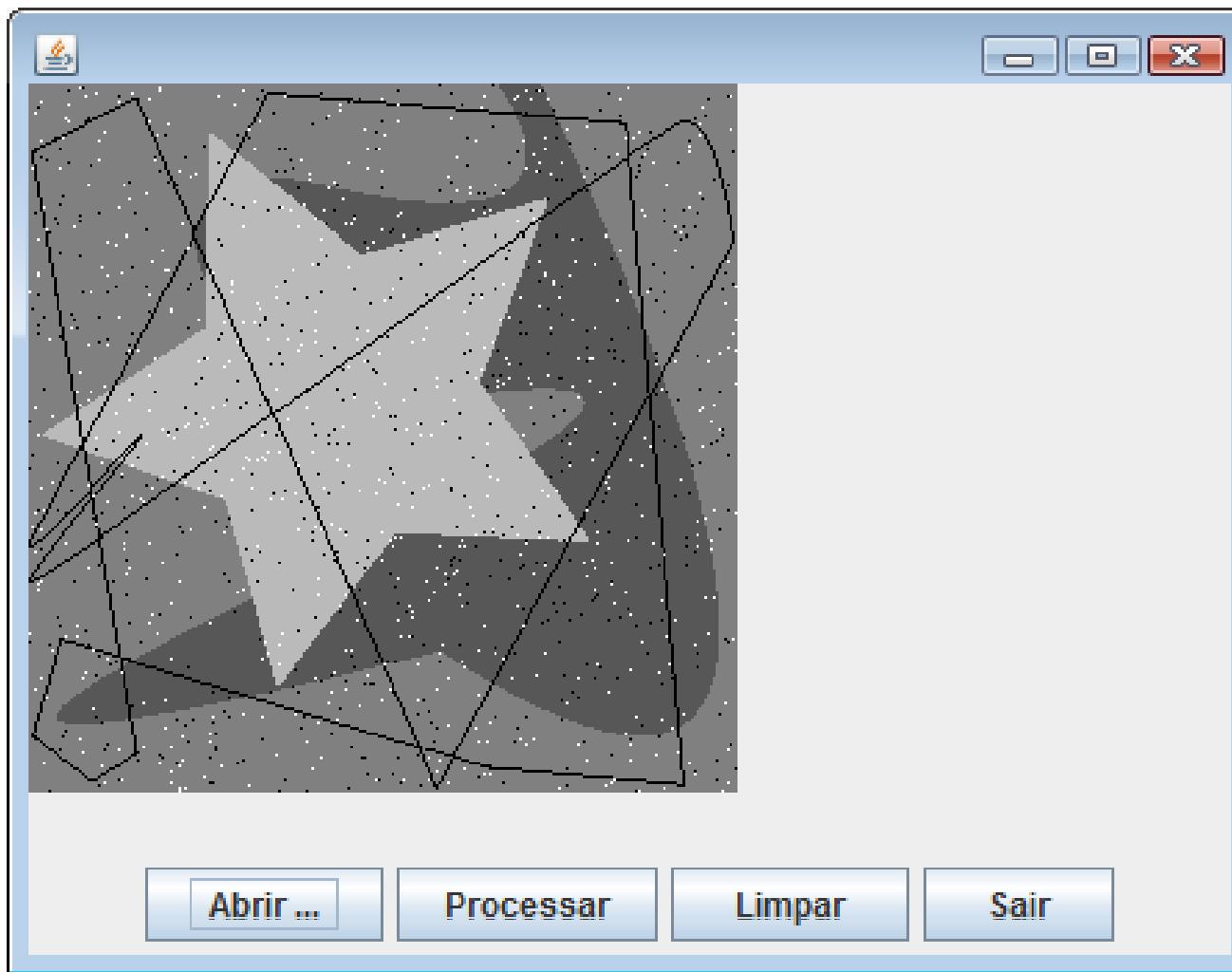
```
// inicialmente aplica-se a dilatação
byteProc.dilate(1,0);
image = byteProc.createImage();
ImagePlus imDil = new ImagePlus("dilatacao",image);
image = imDil.getImage();      byteProc = new ByteProcessor(image);
// posteriormente aplica-se a erosão
byteProc.erode(1,0);
image = byteProc.createImage();
ImagePlus imErosao = new ImagePlus("erosao",image);
JScrollPane sp = new JScrollPane(
    JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
    JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
ImageCanvas ic = new ImageCanvas(imErosao); sp.add(ic);
sp.setSize(imErosao.getWidth(), imErosao.getHeight());
paineiPrinc.removeAll();
paineiPrinc.add(sp,BorderLayout.CENTER);
} // fim do método processarActionPerformed
```

Listagem: SegRdI.java (acrescentando escrita)

```
// adotando a API Java 2D (alternativa para ImageJ)
BufferedImage bi = new BufferedImage(imErosao.getWidth(),
                                      imErosao.getHeight(), BufferedImage.TYPE_BYTE_BINARY);
Graphics2D g2 = bi.createGraphics();
g2.drawImage(image, 0, 0, null);
g2.dispose();
try { ImageIO.write(bi, "png", new File("imagemFinal.png")); }
catch(IOException ioe) { System.out.println( ioe.getMessage() ); }

// ou ainda, misturando com a API JAI
PlanarImage imagemFinal = JAI.create("awtimage", image);
JAI.create("filestore",imagemFinal,"imagemFinal.png","PNG");
```

Aplicativo



Pra encerrar o assunto...

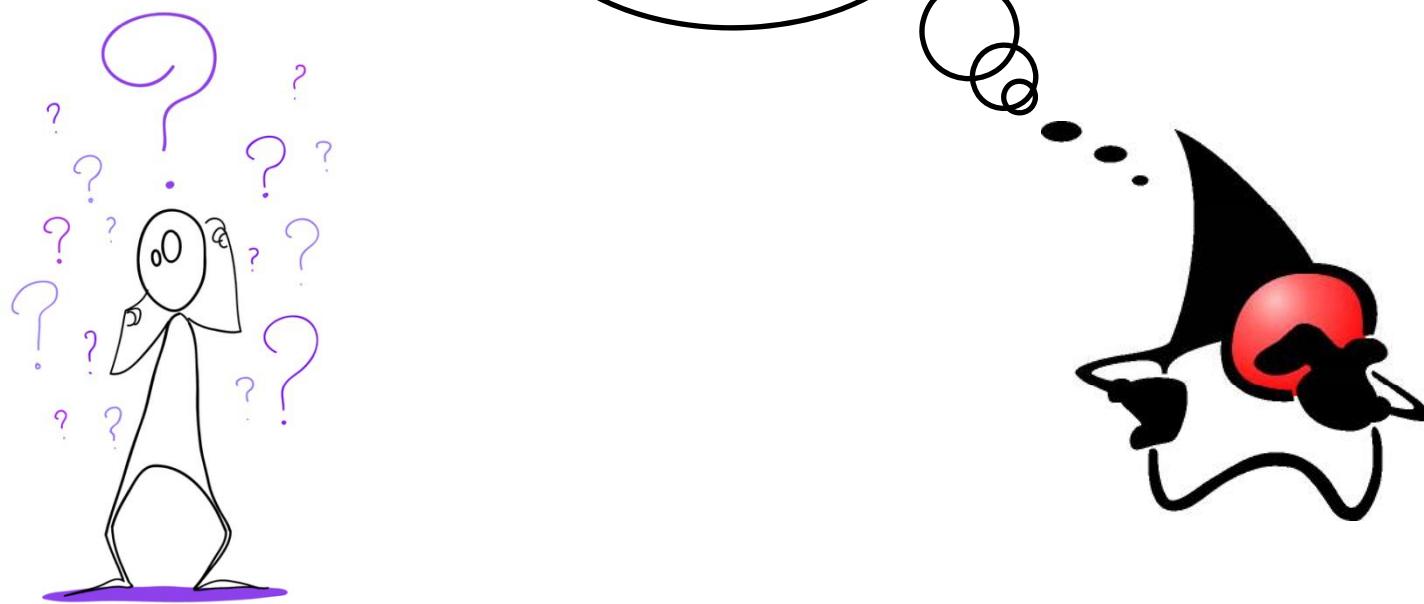


5. CONSIDERAÇÕES FINAIS

Conclusões e Considerações Finais

- Algumas noções de PDI foram expostas para o desenvolvimento de um aplicativo específico;
- Nas APIs Java:
 - a JAI apresenta melhor tratamento de entrada/saída;
 - ImageJ traz mais implementações básicas em sua versão padrão;
 - As duas são bem flexíveis e extensíveis;
 - Pode-se criar um aplicativo combinando as duas APIs.

Dúvidas?



Se gostou, então podemos conversar ...

... encontre-me aqui:



<http://engcomp.sobral.ufc.br/professores/ialis/>



<http://www.linkedin.com/pub/ialis-cavalcante/13/b35/46>



<http://osum.sun.com/profile/IalisJunior>



<http://www.slideshare.net/ialis>



ialis@ufc.br



ERCEMAPI 2009

de 28 a 30 de Outubro em Parnaíba - PI

Obrigado, ...

... e vamos para o almoço!

