Introdução ao Processamento de Imagens Digitais em Java com Aplicações em Ciências Espaciais

Escola de Verão do Laboratório Associado de Computação e Matemática Aplicada

Rafael Santos

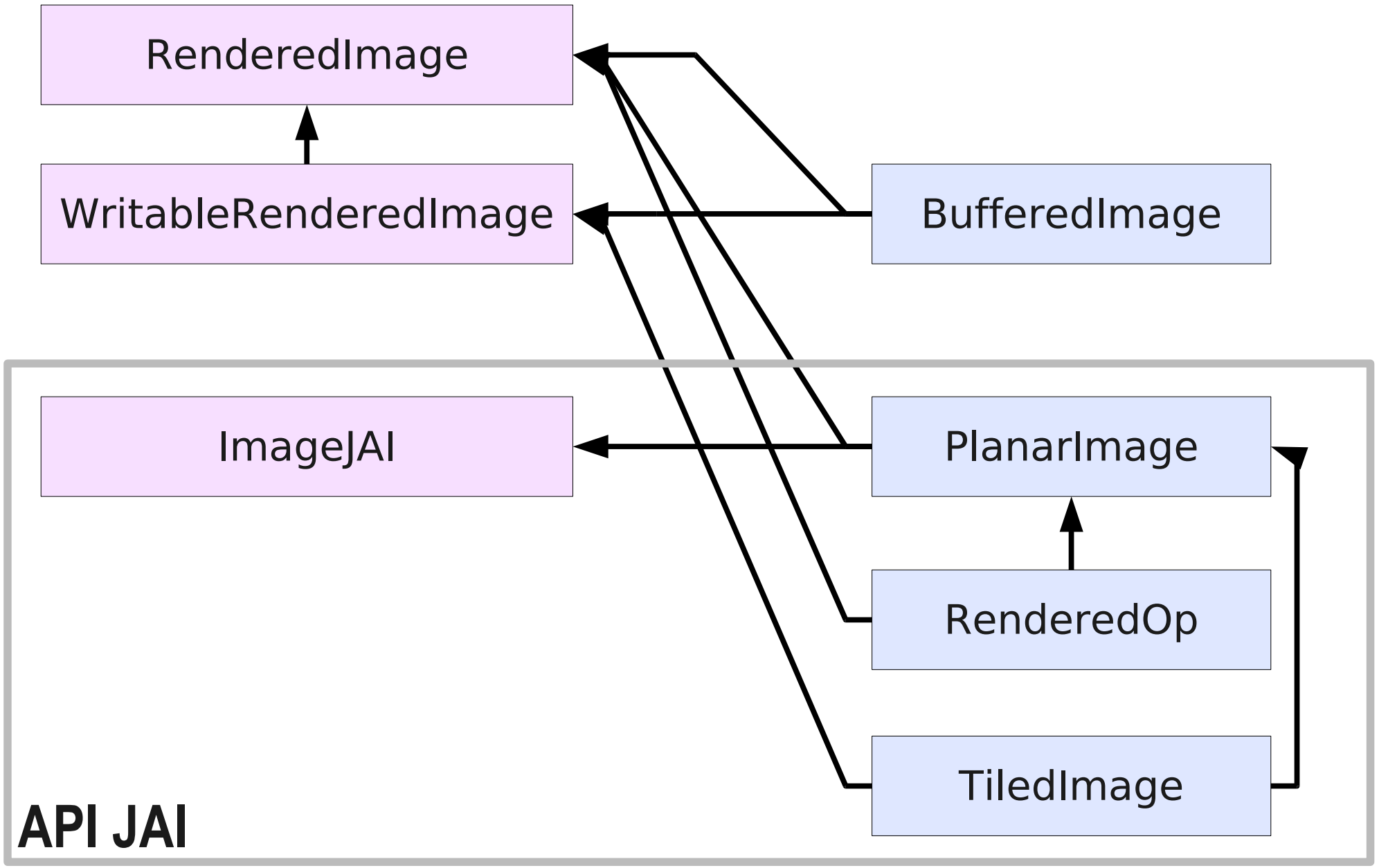
Programa

- ***Dia 1:*** Introdução e aplicações. Representação de imagens digitais. Criando e armazenando imagens.
- ***Dia 2:*** Visualização de imagens.
- ***Dia 3:*** Manipulação de pixels e regiões. Operadores da API JAI.
- ***Dia 4:*** Outros operadores da API JAI. Implementação de algoritmos.

Operadores da API JAI: Introdução

- Classe `JAI` provê método `create`.
- Vários operadores são registrados, chamados de forma unificada.
- Parâmetros (se houver) são passados através de instância de `ParameterBlock`.
- Método retorna instância de `RenderedOp` → *cast* para `PlanarImage` se necessário.

Representação de Imagens

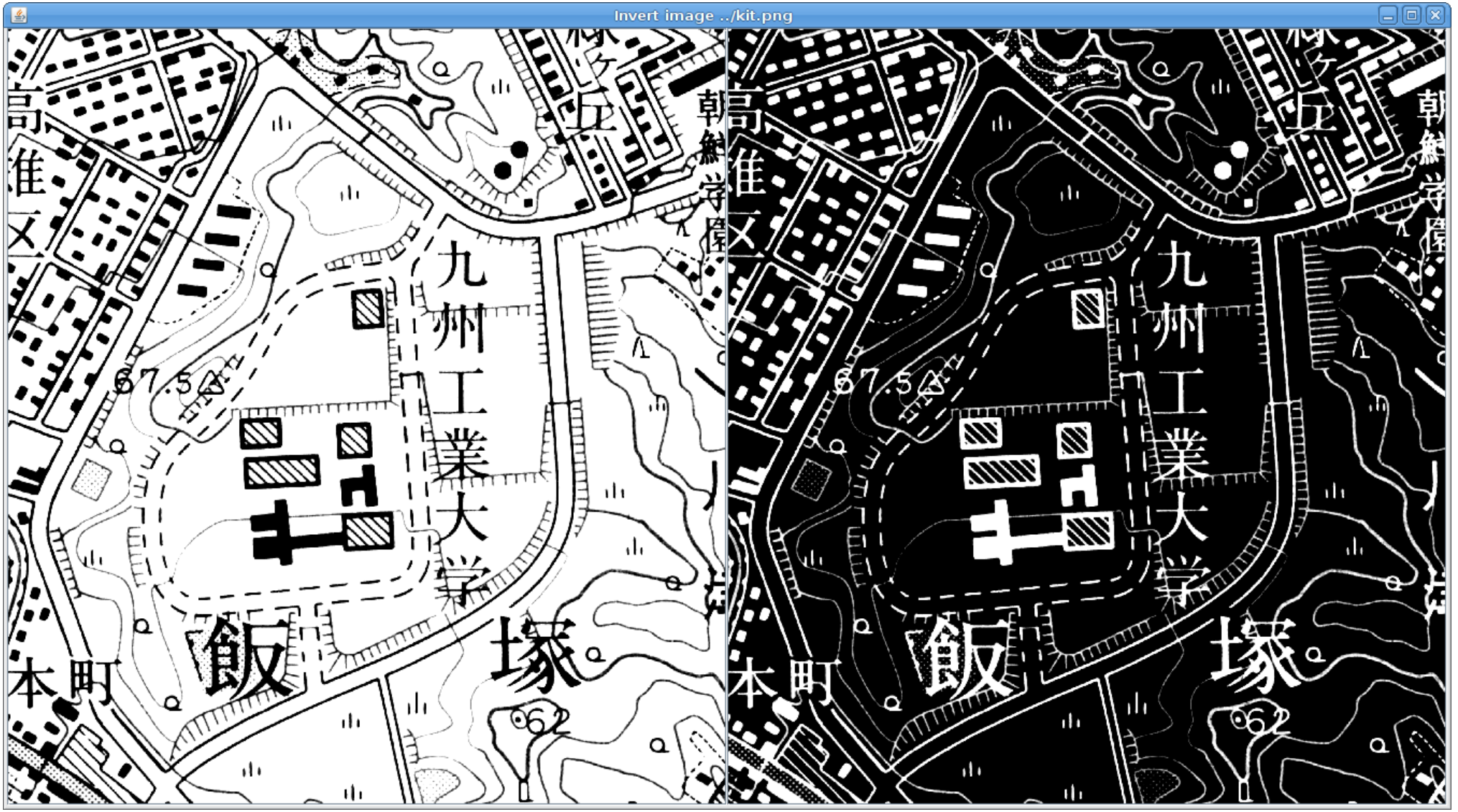


Operadores da API JAI: *invert*

- Inverte os valores dos pixels.
  - Tipos com sinal: saída = -entrada
  - Tipos sem sinal: saída = máximo - entrada

```
public static void main(String[] args)
  {
  PlanarImage input = JAI.create("fileload", args[0]);
  PlanarImage output = JAI.create("invert", input);
  JFrame frame = new JFrame();
  frame.setTitle("Invert image "+args[0]);
  frame.getContentPane().add(
      new DisplayTwoSynchronizedImages(input,output));
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.pack();
  frame.setVisible(true);
  }
```

Operadores da API JAI: invert

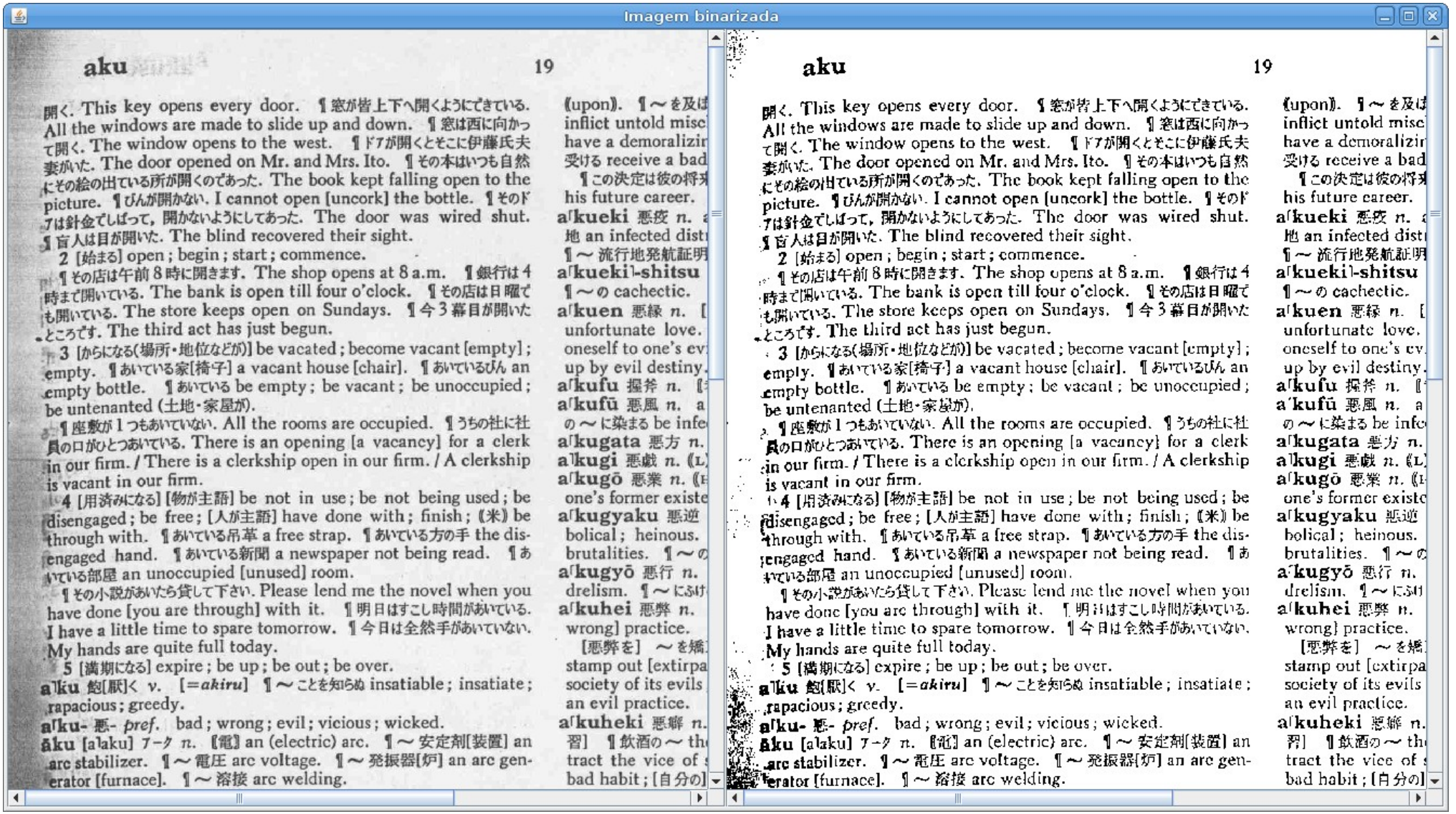


Operadores da API JAI: *binarize*

- Transforma pixels em valores binários por comparação com constante (1 se ≥ constante).

```
public static void main(String[] args)
  {
  PlanarImage imagem = JAI.create("fileload", args[0]);
  ParameterBlock pb = new ParameterBlock();
  pb.addSource(imagem);
  pb.add(127.0);
  PlanarImage binarizada = JAI.create("binarize", pb);
  JFrame frame = new JFrame("Imagem binarizada");
  frame.add(new DisplayTwoSynchronizedImages(imagem,binarizada));
  frame.pack();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setVisible(true);
  }
```

Operadores da API JAI: *binarize*

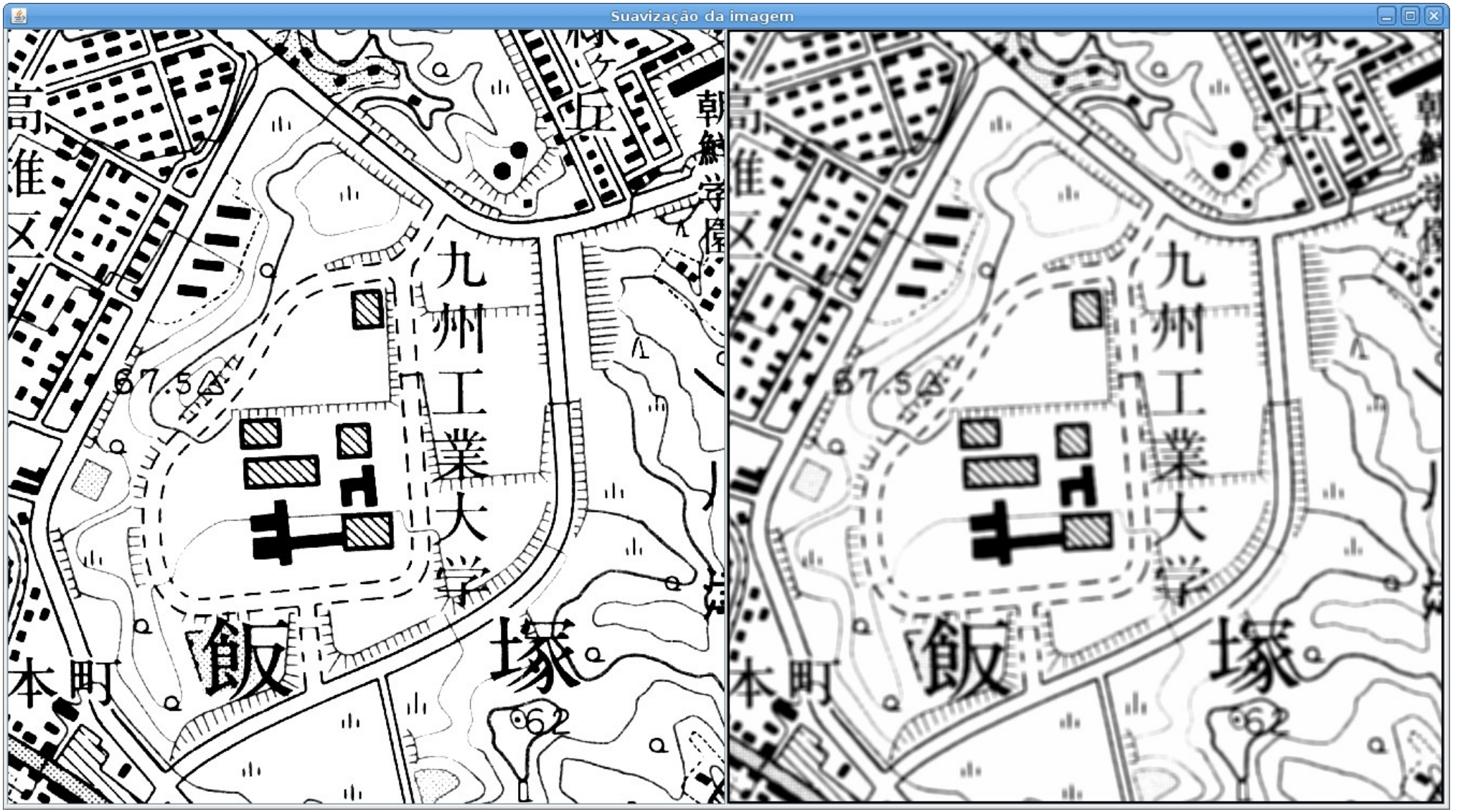


Operadores da API JAI: *convolve*

- Convolução com um *kernel*.
  - Este exemplo: suavização.

```
public static void main(String[] args)
  {
  PlanarImage imagem = JAI.create("fileload", args[0]);
  float[] kernelMatrix = { 1f/25f, 1f/25f, 1f/25f, 1f/25f, 1f/25f,
                           1f/25f, 1f/25f, 1f/25f, 1f/25f, 1f/25f,
                           1f/25f, 1f/25f, 1f/25f, 1f/25f, 1f/25f,
                           1f/25f, 1f/25f, 1f/25f, 1f/25f, 1f/25f,
                           1f/25f, 1f/25f, 1f/25f, 1f/25f, 1f/25f};
  KernelJAI kernel = new KernelJAI(5,5,kernelMatrix);
  PlanarImage bordas = JAI.create("convolve",imagem,kernel);
  JFrame frame = new JFrame("Suavização da imagem");
  frame.add(new DisplayTwoSynchronizedImages(imagem,bordas));
  frame.pack();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setVisible(true);
  }
```

Operadores da API JAI: *convolve*

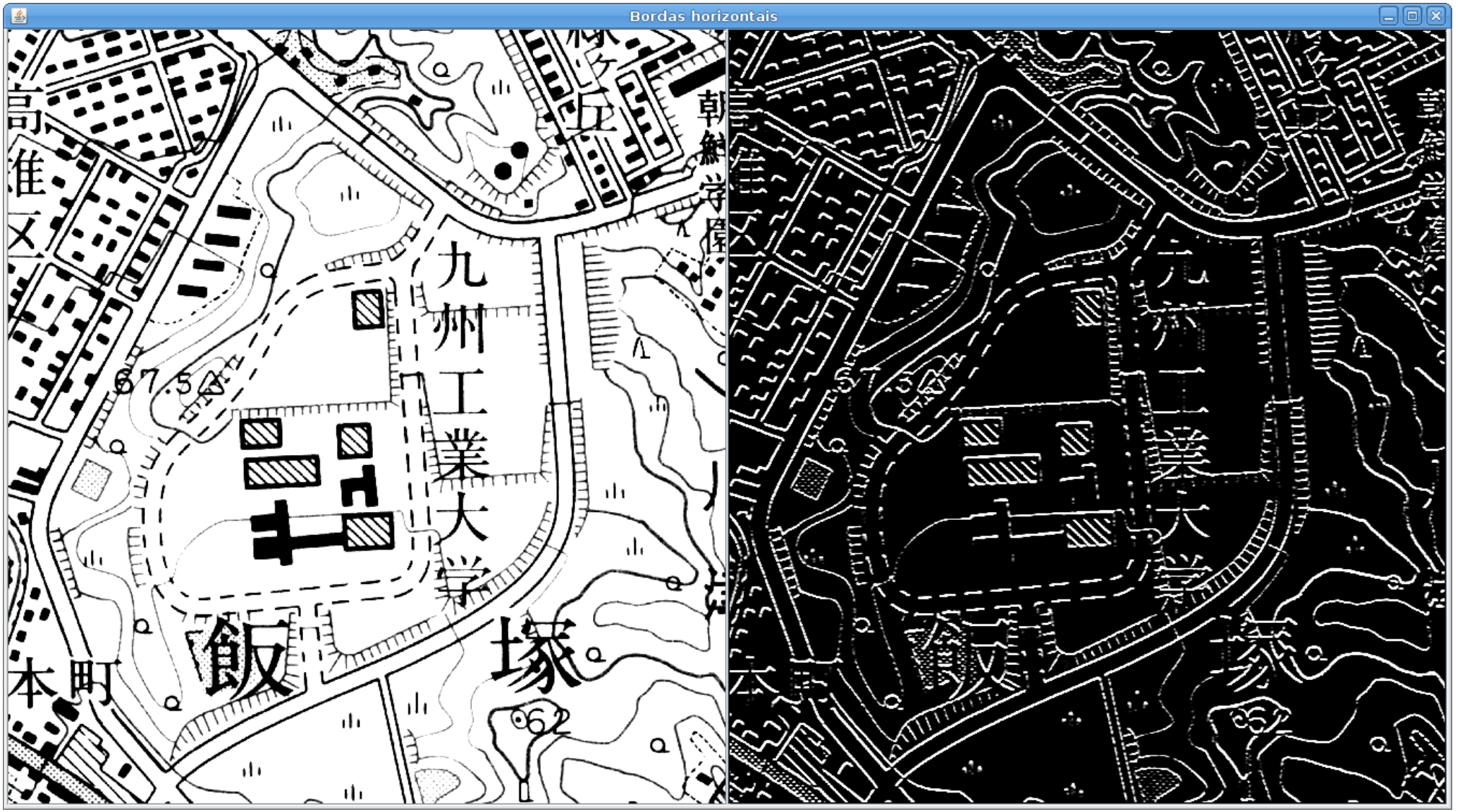


Operadores da API JAI: *convolve*

- Convolução com um *kernel*.
  - Este exemplo: detecção de bordas horizontais (Sobel).

```
public static void main(String[] args)
  {
  PlanarImage imagem = JAI.create("fileload", args[0]);
  float[] kernelMatrix = { -1, -2, -1,
                            0,  0,  0,
                            1,  2,  1 };
  KernelJAI kernel = new KernelJAI(3,3,kernelMatrix);
  PlanarImage bordas = JAI.create("convolve",imagem,kernel);
  JFrame frame = new JFrame("Bordas horizontais");
  frame.add(new DisplayTwoSynchronizedImages(imagem,bordas));
  frame.pack();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setVisible(true);
  }
```

Operadores da API JAI: *convolve*



Operadores da API JAI: *dilate*

- Expansão de regiões da imagem com elemento estrutural.

```
public static void main(String[] args)
  {
  PlanarImage imagem = JAI.create("fileload", args[0]);
  float[] estrutura = {   0,   0,   0,   0,   0,   0,   0,
                          0,   1,   1,   1,   1,   1,   0,
                          0,   1,   1,   1,   1,   1,   0,
                          0,   1,   1,   1,   1,   1,   0,
                          0,   1,   1,   1,   1,   1,   0,
                          0,   1,   1,   1,   1,   1,   0,
                          0,   0,   0,   0,   0,   0,   0};
  KernelJAI kernel = new KernelJAI(7,7,estrutura);
  ParameterBlock p = new ParameterBlock();
  p.addSource(imagem);
  p.add(kernel);
  PlanarImage dilatada = JAI.create("dilate",p);
  JFrame frame = new JFrame("Imagem dilatada");
  frame.add(new DisplayTwoSynchronizedImages(imagem,dilatada));
  frame.pack();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setVisible(true);
  }
```

Operadores da API JAI: *dilate*

Regiões brancas são dilatadas!

Operadores da API JAI: *erode*

- Redução de regiões da imagem com elemento estrutural.

```
public static void main(String[] args)
  {
  PlanarImage imagem = JAI.create("fileload", args[0]);
  float[] estrutura = {   0,   0,   0,   0,   0,   0,   0,
                          0,   1,   1,   1,   1,   1,   0,
                          0,   1,   1,   1,   1,   1,   0,
                          0,   1,   1,   1,   1,   1,   0,
                          0,   1,   1,   1,   1,   1,   0,
                          0,   1,   1,   1,   1,   1,   0,
                          0,   0,   0,   0,   0,   0,   0};
  KernelJAI kernel = new KernelJAI(7,7,estrutura);
  ParameterBlock p = new ParameterBlock();
  p.addSource(imagem);
  p.add(kernel);
  PlanarImage erodida = JAI.create("erode",p);
  JFrame frame = new JFrame("Imagem erodida");
  frame.add(new DisplayTwoSynchronizedImages(imagem,erodida));
  frame.pack();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setVisible(true);
  }
```

Operadores da API JAI: *erode*

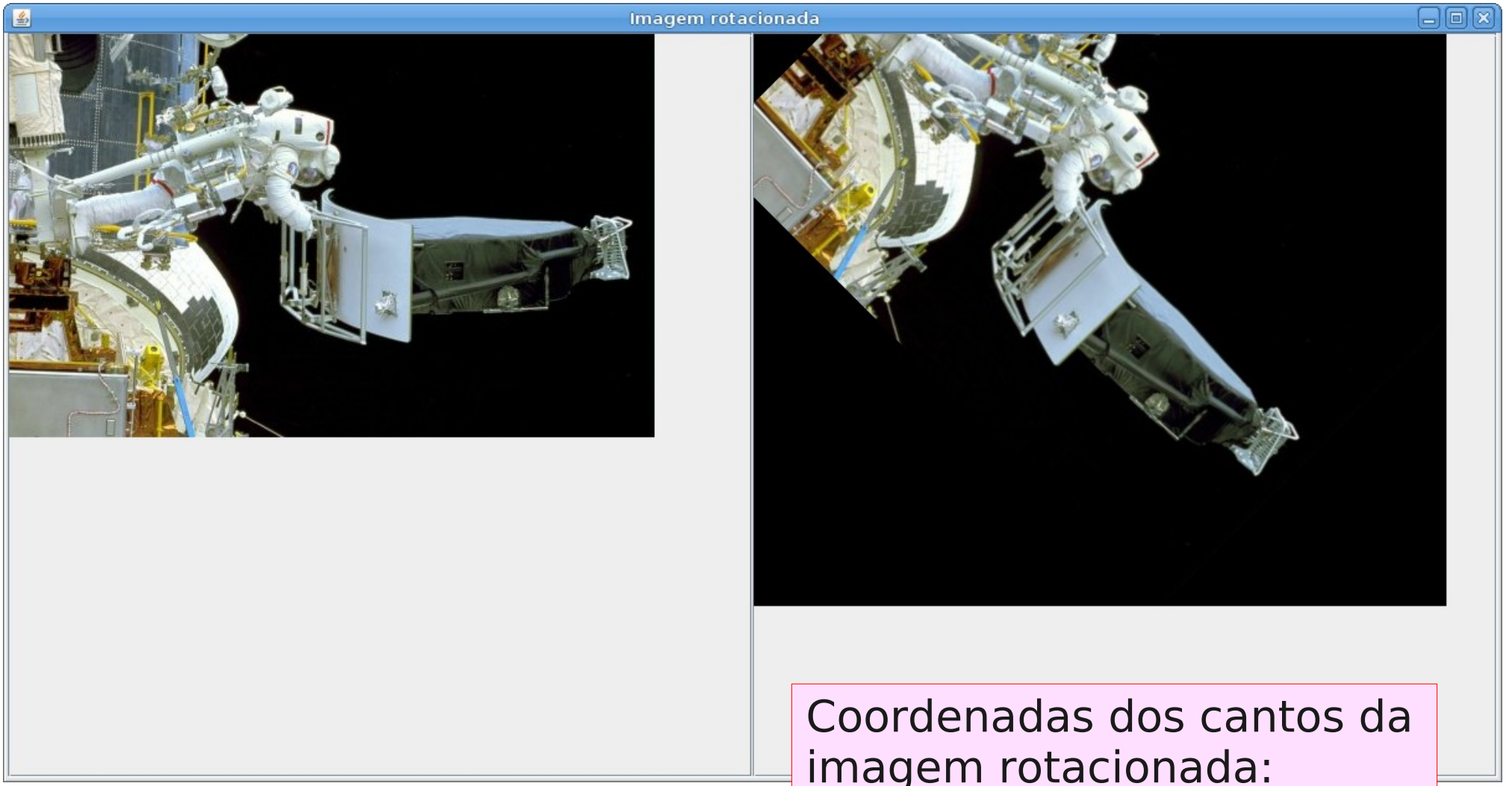
Regiões brancas são dilatadas!

Operadores da API JAI: *rotate*

- Rotação dos pixels da imagem em redor de um ponto.

```
public static void main(String[] args)
  {
  PlanarImage imagem = JAI.create("fileload",args[0]);
  float angle = (float)Math.toRadians(45);
  // Usamos o centro da imagem para rotação
  float centerX = imagem.getWidth()/2f;
  float centerY = imagem.getHeight()/2f;
  ParameterBlock pb = new ParameterBlock();
  pb.addSource(imagem);
  pb.add(centerX);
  pb.add(centerY);
  pb.add(angle);
  pb.add(new InterpolationBilinear());
  PlanarImage rotacionada = JAI.create("rotate", pb);
  JFrame frame = new JFrame("Imagem rotacionada");
  frame.add(new DisplayTwoSynchronizedImages(imagem,rotacionada));
  frame.pack();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setVisible(true);
  }
```

Operadores da API JAI: *rotate*



Coordenadas dos cantos da imagem rotacionada:
(-39, -136) – (558, 461)

Translação da Origem de Imagens

Original

Região para recorte

Origem
200,200

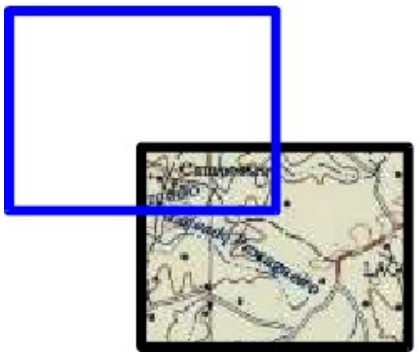
Tamanho
400x300

Região recortada

Mínimo
200,200

Tamanho
400x300

Máximo
600,500



Recorte e translação

Mínimo
0,0

Tamanho
400x300

Máximo
400,300

Operadores da API JAI: *rotate*

- JAI permite imagens com pixels com coordenadas negativas!
  - `DisplayJAI`, `ImageIO` e `JAI.create("filestore")` não.
  - Solução: mover a origem da imagem com o operador `translate`.

```
public static void main(String[] args)
  {
  PlanarImage imagem = JAI.create("fileload",args[0]);
  float angle = (float)Math.toRadians(45);
  // Usamos o centro da imagem para rotação
  float centerX = imagem.getWidth()/2f;
  float centerY = imagem.getHeight()/2f;
  ParameterBlock pb = new ParameterBlock();
  pb.addSource(imagem);
  pb.add(centerX);
  pb.add(centerY);
  pb.add(angle);
  pb.add(new InterpolationBilinear());
  PlanarImage rotacionada = JAI.create("rotate", pb);
```

Operadores da API JAI: *rotate*

```
// Ajustamos a origem da imagem
pb = new ParameterBlock();
pb.addSource(rotacionada);
pb.add((float)-rotacionada.getMinX());
pb.add((float)-rotacionada.getMinY());
PlanarImage rotacionadaOK =
  JAI.create("translate",pb,null);
JFrame frame = new JFrame("Imagem rotacionada");
frame.add(
   new DisplayTwoSynchronizedImages(imagem,rotacionadaOK));
frame.pack();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
}
```

Operadores da API JAI: *rotate*

Operadores da API JAI: *scale*

- Aumenta ou diminui a quantidade de pixels na imagem.
  - Valores dos pixels podem ser interpolados.

```
public static void main(String[] args)
  {
  PlanarImage imagem = JAI.create("fileload",args[0]);
  float scale = 0.3f;
  ParameterBlock pb = new ParameterBlock();
  pb.addSource(imagem);
  pb.add(scale);
  pb.add(scale);
  pb.add(0.0F);
  pb.add(0.0F);
  pb.add(new InterpolationNearest());
  PlanarImage reescalada = JAI.create("scale", pb);
  JFrame frame = new JFrame("Imagem reescalada");
  frame.add(new DisplayTwoSynchronizedImages(imagem,reescalada));
  frame.pack();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setVisible(true);
  }
```

Operadores da API JAI: *scale*

Operadores da API JAI: *crop*, *translate*, *scale*

- Pequena aplicação que recorta e amplia uma região em uma imagem.
- Parâmetros passados pela linha de comando.

```
public static void main(String[] args)
  {
  PlanarImage imagem = JAI.create("fileload",args[0]);
  ParameterBlock pb = new ParameterBlock();
  float x = Float.parseFloat(args[1]);
  float y = Float.parseFloat(args[2]);
  float w = Float.parseFloat(args[3]);
  float h = Float.parseFloat(args[4]);
  float z = Float.parseFloat(args[5]);
```

Operadores da API JAI: *crop*, *translate*, *scale*

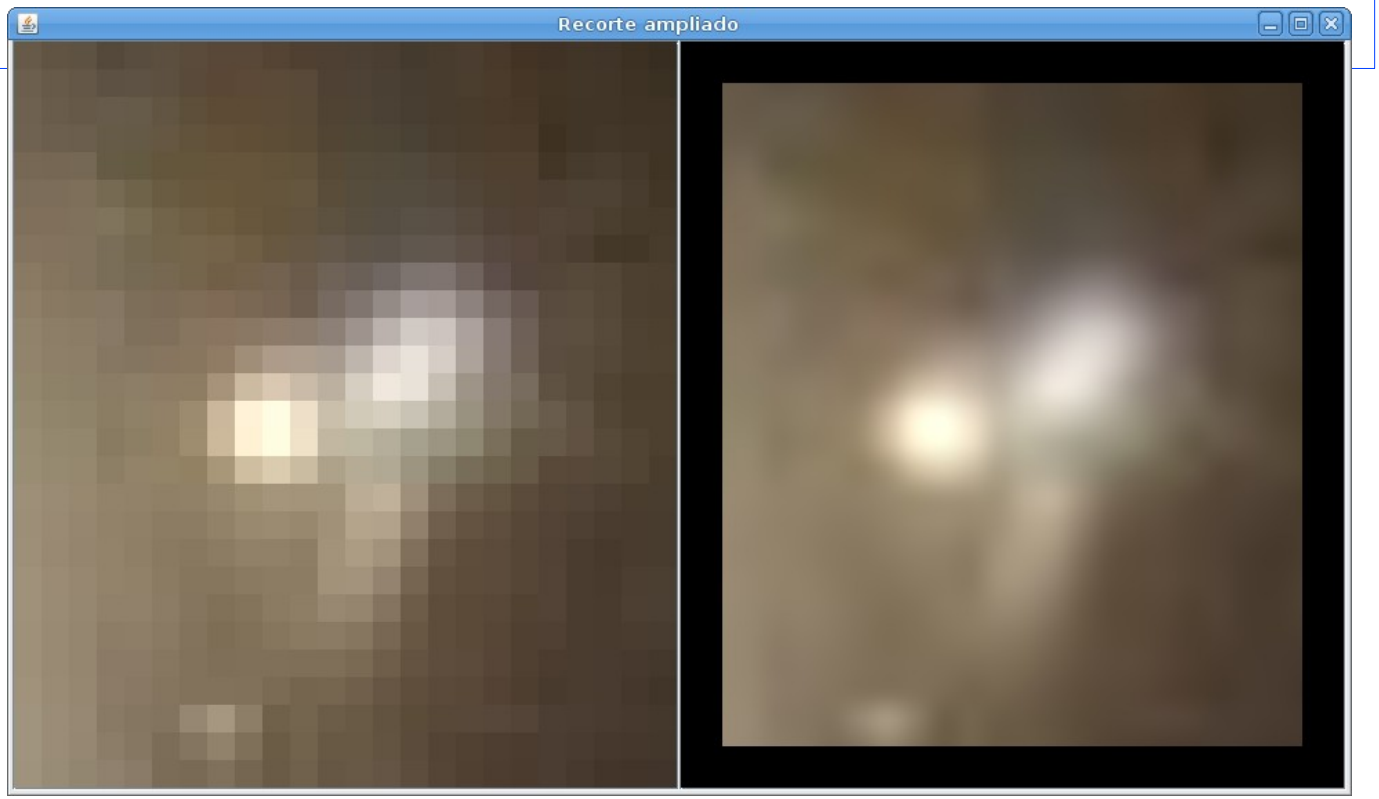
```
// Recorta
pb.addSource(imagem);
pb.add(x);
pb.add(y);
pb.add(w);
pb.add(h);
PlanarImage recortada = JAI.create("crop",pb,null);
// Reposiciona
pb = new ParameterBlock();
pb.addSource(recortada);
pb.add((float)-x);
pb.add((float)-y);
PlanarImage recortadaOK = JAI.create("translate",pb,null);
```

Operadores da API JAI: *crop*, *translate*, *scale*

```
// Amplia (2 versões)
pb = new ParameterBlock();
pb.addSource(recortadaOK);
pb.add(z);
pb.add(z);
pb.add(0.0F);
pb.add(0.0F);
pb.add(new InterpolationNearest());
PlanarImage resultado1 = JAI.create("scale", pb);
pb = new ParameterBlock();
pb.addSource(recortadaOK);
pb.add(z);
pb.add(z);
pb.add(0.0F);
pb.add(0.0F);
pb.add(new InterpolationBicubic(2));
PlanarImage resultado2 = JAI.create("scale", pb);
```

Operadores da API JAI: *crop*, *translate*, *scale*

```
JFrame frame = new JFrame("Recorte ampliado");
frame.add(
  new DisplayTwoSynchronizedImages(resultado1,resultado2));
frame.pack();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
}
```



java wvc/operadores/Recorta astro013.jpg 461 896 24 27 20

Operadores da API JAI: *histogram*

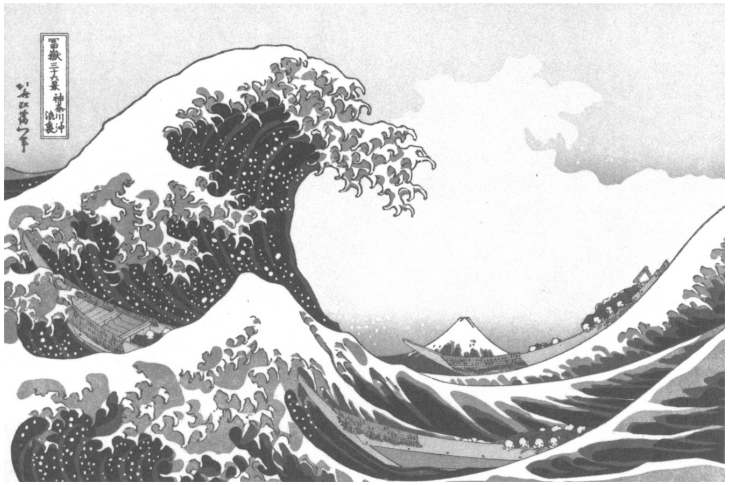
- Operador sem imagem resultante: calcula histogramas em uma imagem.
  - Histogramas são recuperadors como propriedades do `RenderedOp` resultante.

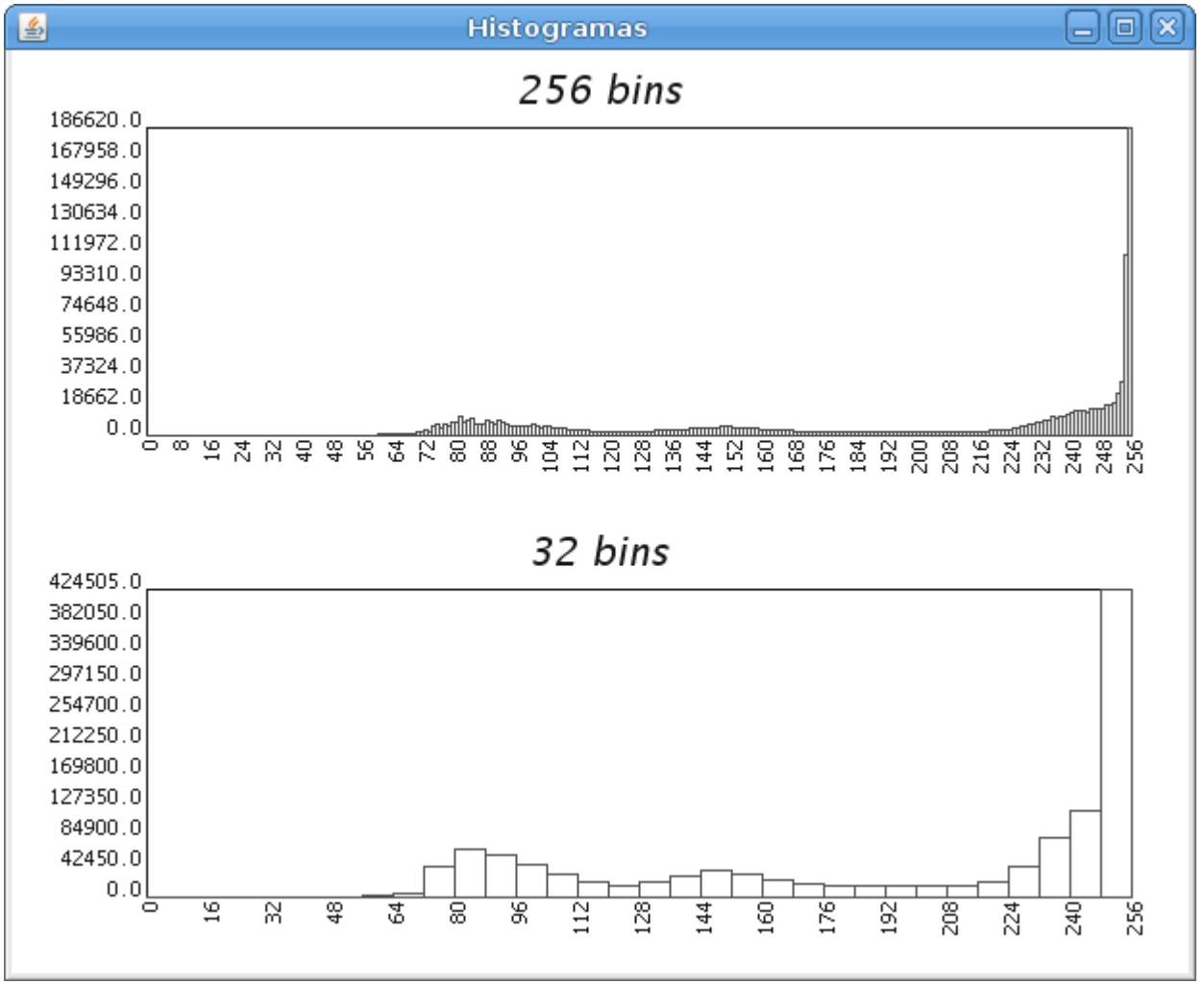
```
public static void main(String[] args)
  {
  PlanarImage image = JAI.create("fileload", args[0]);
  // Primeiro histograma com 256 bins.
  ParameterBlock pb1 = new ParameterBlock();
  pb1.addSource(image);
  pb1.add(null);
  pb1.add(1); pb1.add(1);
  pb1.add(new int[]{256});
  pb1.add(new double[]{0});  pb1.add(new double[]{256});
  PlanarImage dummyImage1 = JAI.create("histogram", pb1);
  Histogram histo1 =
    (Histogram)dummyImage1.getProperty("histogram");
```

Operadores da API JAI: *histogram*

```
// Segundo histograma com 32 bins.
ParameterBlock pb2 = new ParameterBlock();
pb2.addSource(image);
pb2.add(null);
pb2.add(1); pb2.add(1);
pb2.add(new int[]{32});
pb2.add(new double[]{0});  pb2.add(new double[]{256});
PlanarImage dummyImage2 = JAI.create("histogram", pb2);
Histogram histo2 =
  (Histogram)dummyImage2.getProperty("histogram");
// Exibimos os histogramas usando um componente específico.
JFrame f = new JFrame("Histogramas");
DisplayHistogram dh1 = new DisplayHistogram(histo1,"256 bins");
dh1.setBinWidth(2); dh1.setHeight(160); dh1.setIndexMultiplier(1);
DisplayHistogram dh2 = new DisplayHistogram(histo2,"32 bins");
dh2.setBinWidth(16); dh2.setHeight(160);dh2.setIndexMultiplier(8);
dh2.setSkipIndexes(2);
f.getContentPane().setLayout(new GridLayout(2,1));
f.getContentPane().add(dh1);  f.getContentPane().add(dh2);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.pack();  f.setVisible(true);
}
```

Operadores da API JAI: *histogram*





Aplicação: *Pan Sharpening*

- Alguns satélites tem bandas com resoluções diferentes.
- Podemos usar combinações de bandas (cores e pancromáticas) para obter melhor resolução espacial.



Aplicação: *Pan Sharpening*

```
PlanarImage iRed = JAI.create("fileload",args[0]);
PlanarImage iGreen = JAI.create("fileload",args[1]);
PlanarImage iBlue = JAI.create("fileload",args[2]);
PlanarImage panImage = JAI.create("fileload",args[3]);
ParameterBlock pb = new ParameterBlock();
pb.addSource(iRed);
pb.addSource(iGreen);
pb.addSource(iBlue);
PlanarImage rgbImage = JAI.create("bandmerge", pb);
```

```
pb = new ParameterBlock();
pb.addSource(rgbImage);
float scaleX = (1f*panImage.getWidth()/iRed.getWidth());
float scaleY = (1f*panImage.getHeight()/iRed.getHeight());
pb.add(scaleX);
pb.add(scaleY);
rgbImage = JAI.create("scale",pb);
```

Aplicação: *Pan Sharpening*

```
IHSColorSpace ihs = IHSColorSpace.getInstance();
ColorModel IHSColorModel =
    new ComponentColorModel(ihs,
                            new int []{8,8,8},
                            false,false,
                            Transparency.OPAQUE,
                            DataBuffer.TYPE_BYTE);
pb = new ParameterBlock();
pb.addSource(rgbImage);
pb.add(IHSColorModel);
RenderedImage imageIHS  = JAI.create("colorconvert", pb);
```

```
PlanarImage[] IHSBands = new PlanarImage[3];
for(int band=0;band<3;band++)
   {
   pb = new ParameterBlock();
   pb.addSource(imageIHS);
   pb.add(new int[]{band});
   IHSBands[band] = JAI.create("bandselect",pb);
   }
```

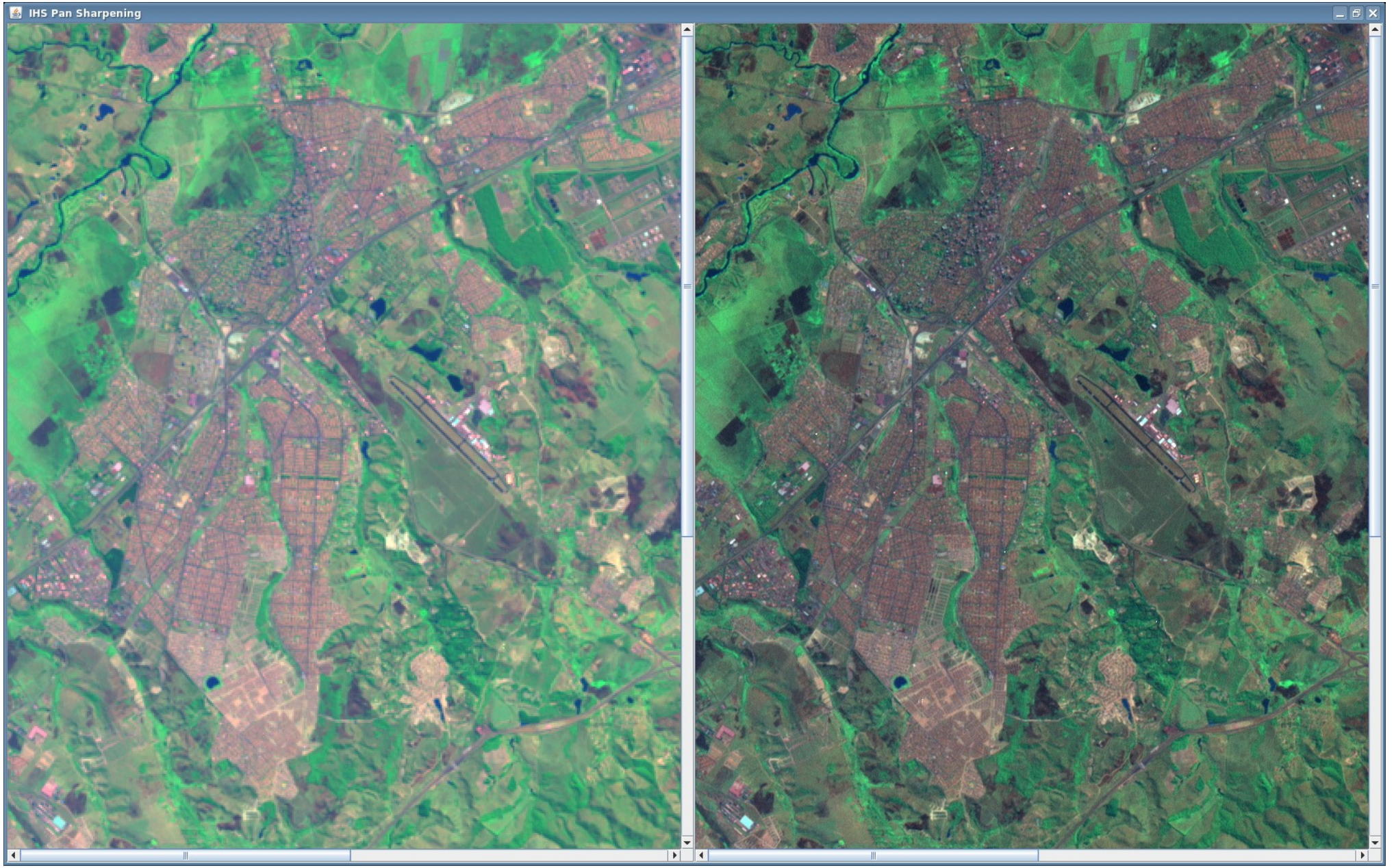
Aplicação: *Pan Sharpening*

```
ImageLayout imageLayout = new ImageLayout();
imageLayout.setColorModel(IHSColorModel);
imageLayout.setSampleModel(imageIHS.getSampleModel());
RenderingHints rendHints =
  new RenderingHints(JAI.KEY_IMAGE_LAYOUT,imageLayout);
pb = new ParameterBlock();
pb.addSource(panImage);
pb.addSource(IHSBands[1]);
pb.addSource(IHSBands[2]);
RenderedImage panSharpenedIHSImage =
   JAI.create("bandmerge", pb, rendHints);
```

```
pb = new ParameterBlock();
pb.addSource(panSharpenedIHSImage);
pb.add(rgbImage.getColorModel()); // RGB color model
PlanarImage finalImage = JAI.create("colorconvert", pb);
```

```
JFrame frame = new JFrame("IHS Pan Sharpening");
frame.add(new DisplayTwoSynchronizedImages(rgbImage,finalImage));
frame.pack();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```

Aplicação: *Pan Sharpening*



Aplicação: *Pan Sharpening*

Programa

- ***Dia 1:*** Introdução e aplicações. Representação de imagens digitais. Criando e armazenando imagens.
- ***Dia 2:*** Visualização de imagens.
- ***Dia 3:*** Manipulação de pixels e regiões. Operadores da API JAI.
- ***Dia 4:*** Outros operadores da API JAI. Implementação de algoritmos.

Mais informações em...

- http://www.lac.inpe.br/~rafael.santos
  - http://www.lac.inpe.br/~rafael.santos/piapresentacoes.jsp
  - http://www.lac.inpe.br/JIPCookbook/index.jsp
- http://www.lac.inpe.br/ELAC/index.jsp