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.

Criando Novos Operadores

- Qual tipo de operação estamos implementando?
  - `SourcelessOpImage`: operadores sem imagens de entrada.
  - `PointOpImage`: pixels da saída dependem dos mesmos pixels da entrada.
  - `AreaOpImage`: pixels da saída dependem de área ao redor dos da entrada.
  - `GeometricOpImage`: pixels da saída podem depender de todos da entrada.
  - `StatisticsOpImage`: operadores que calculam estatísticas sobre imagem de entrada.

Criando Novos Operadores

- Receita de bolo (relativamente) simples para imagens renderizadas.

1. Criar classe que herda de `XXXOpImage`.
   - Como `XXXOpImage` é abstrata, devemos implementar métodos que fazem o processamento (quase sempre `computeTile`).
2. Criar classe que implementa `RenderedImageFactory`.
   - Implementa método `create`.

Criando Novos Operadores

3. Criar classe que implementa `OperationDescriptor` ou herda de `OperationDescriptorImpl`, que descreve os parâmetros e valores *default* do operador.
4. Registrar o novo operador junto ao `OperationRegistry` e `RIFRegistry` (podemos criar método para registro na classe do passo anterior).

Novo Operador: segmenta3

- Segmentador por limiar semelhante a `binarize`, mas com 2 limiares.
- Classe que herda de `PointOpImage`.
  - Contém construtor para inicializar atributos e método `computeTile`.

Passo 1

```
public class Segmenta3OpImage extends PointOpImage
  {
  private RenderedImage source;
  private int threshold1,threshold2;

  public Segmenta3OpImage(RenderedImage source,int th1,int th2,
                          ImageLayout layout,RenderingHints hints,
                          boolean b)
    {
    super(source,layout,hints,b);
    this.source = source;
    this.threshold1 = th1;  this.threshold2 = th2;
    }
```

Novo Operador: segmenta3

Passo 1

```
public Raster computeTile(int x,int y)
  {
  Raster r = source.getTile(x,y);
  int minX = r.getMinX();
  int minY = r.getMinY();
  int width = r.getWidth();
  int height = r.getHeight();
  // Criamos um WritableRaster da região sendo considerada.
  WritableRaster wr =
      r.createCompatibleWritableRaster(minX,minY,width,height);
  for(int l=0;l<r.getHeight();l++)
    for(int c=0;c<r.getWidth();c++)
      for(int b=0;b<r.getNumBands();b++)
        {
        int p = r.getSample(c+minX,l+minY,b);
        if (p < threshold1) p = 0;
        else if (p > threshold2) p = 255;
        else p = 127;
        wr.setSample(c+minX,l+minY,b,p);
        }
  return wr;
  }
```

Novo Operador: segmenta3

Passo 2

```
public class Segmenta3RIF implements RenderedImageFactory
  {
  public RenderedImage create(ParameterBlock paramBlock,
                              RenderingHints hints)
    {
    RenderedImage source = paramBlock.getRenderedSource(0);
    int threshold1 = paramBlock.getIntParameter(0);
    int threshold2 = paramBlock.getIntParameter(1);
    ImageLayout layout = new ImageLayout(source);
    return new Segmenta3OpImage(source,threshold1,threshold2,
                                layout,hints,false);
    }
  }
```

Novo Operador: segmenta3

Passo 3

```
public class Segmenta3Descriptor extends OperationDescriptorImpl
  {
  private static final String opName = "segmenta3";
  private static final String vendorName =
       "Hypothetical Image Processing Lab";
  private static final String[][] resources =
   {
    {"GlobalName", opName},
    {"LocalName",  opName},
    {"Vendor",     vendorName},
    {"Description","A simple 3-level image segmentation operator"},
    {"DocURL",     "http://www.lac.inpe.br/~rafael.santos"},
    {"Version",    "1.0"},
    {"arg0Desc",   "First Threshold Value"},
    {"arg1Desc",   "Second Threshold Value"}
    };
  private static final String[] supportedModes = {"rendered"};
```

Novo Operador: segmenta3

Passo 3

```
private static final String[] paramNames = {"1st threshold",
                                            "2nd threshold"};
private static final Class[] paramClasses = {Integer.class,
                                             Integer.class};
private static final Object[] paramDefaults = {new Integer(85),
                                               new Integer(170) };
private static final Range[] validParamValues =
  {
  new Range(Integer.class, Integer.MIN_VALUE, Integer.MAX_VALUE),
  new Range(Integer.class, Integer.MIN_VALUE, Integer.MAX_VALUE)
  };
private static final int numSources = 1;
private static boolean registered = false;
```

Novo Operador: segmenta3

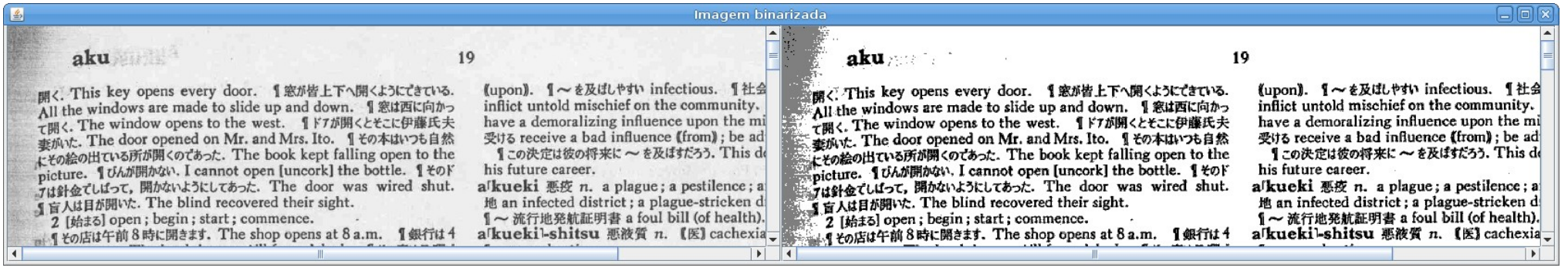
Passo 3/4

```
public Segmenta3Descriptor()
  {
  super(resources,supportedModes,numSources,paramNames,
        paramClasses,paramDefaults,validParamValues);
  }

public static void register()
  {
  if (!registered)
    {
    OperationRegistry op =
      JAI.getDefaultInstance().getOperationRegistry();
    Segmenta3Descriptor desc = new Segmenta3Descriptor();
    op.registerDescriptor(desc);
    Segmenta3RIF rif = new Segmenta3RIF();
    RIFRegistry.register(op,opName,vendorName,rif);
    registered = true;
    }
  }
```

Novo Operador: segmenta3

```
public static void main(String[] args)
  {
  Segmenta3Descriptor.register();
  PlanarImage imagem = JAI.create("fileload", args[0]);
  ParameterBlock p = new ParameterBlock();
  p.addSource(imagem);
  p.add(new Integer(120));
  p.add(new Integer(200));
  PlanarImage resultado = JAI.create("segmenta3",p);
  JFrame frame = new JFrame("Imagem binarizada");
  frame.add(new DisplayTwoSynchronizedImages(imagem,resultado));
  frame.pack();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setVisible(true);
  }
```



Novo Operador: contapixels

- Conta número de pixels com cor semelhante a um parâmetro (com tolerância).
- Classe que herda de `StatisticsOpImage`.
  - Contém construtor para inicializar atributos e vários métodos para acumular estatísticas.

Passo 1

```
public class ContaPixelsOpImage extends StatisticsOpImage
  {
  private Color target;
  private Float tolerance;
  private Long count;

  public ContaPixelsOpImage(RenderedImage source,
                            Color target,Float tolerance)
    {
    super(source,null,source.getMinX(),source.getMinY(),1,1);
    this.target = target;       this.tolerance = tolerance;
    count = null;
    }
```

Novo Operador: contapixels

Passo 1

```
protected void accumulateStatistics(String name, Raster raster,
                                    Object stats)
  {
  if (count == null) count = new Long(0);
  int r,g,b;
  for(int l=0;l<raster.getHeight();l++)
    for(int c=0;c<raster.getWidth();c++)
      {
      int x = raster.getMinX()+c;
      int y = raster.getMinY()+l;
      r = raster.getSample(x,y,0);
      g = raster.getSample(x,y,1);
      b = raster.getSample(x,y,2);
      float dist = (target.getRed()-r)*(target.getRed()-r)+
                   (target.getGreen()-g)*(target.getGreen()-g)+
                   (target.getBlue()-b)*(target.getBlue()-b);
      if (dist<=tolerance*tolerance) count++;
      }
  }
```

Novo Operador: contapixels

Passo 1

```
protected Object createStatistics(String arg0)
  {
  if (count == null) count = new Long(0);
  return count;
  }

protected String[] getStatisticsNames()
  {
  return new String[]{"count"};
  }

public Object getProperty(String name)
  {
  if (count == null) super.getProperty(name);
  return count;
  }
```

Novo Operador: contapixels

Passo 2

```
public class ContaPixelsRIF implements RenderedImageFactory
  {
  public RenderedImage create(ParameterBlock paramBlock,
                              RenderingHints hints)
    {
    RenderedImage source = paramBlock.getRenderedSource(0);
    Color target = (Color)paramBlock.getObjectParameter(0);
    Float tolerance = (Float)paramBlock.getObjectParameter(1);
    return new ContaPixelsOpImage(source,target,tolerance);
    }
  }
```

Novo Operador: contapixels

Passo 3

```
public class ContaPixelsDescriptor extends OperationDescriptorImpl
  {
  private static final String opName = "contapixels";
  private static final String vendorName =
                              "Hypothetical Image Processing Lab";
  private static final String[][] attributes =
    {
      {"GlobalName",   opName},
      {"LocalName",    opName},
      {"Vendor",       vendorName},
      {"Description",  "A simple RGB pixel counting operator"},
      {"DocURL",       "http://www.lac.inpe.br/~rafael.santos"},
      {"Version",      "1.0"},
      {"arg0Desc",     "Target value (color used for similarity)"},
      {"arg1Desc",     "Tolerance value"},
    };
  private static final String[] modes = {"rendered"};
```

Novo Operador: contapixels

Passo 3

```
  private static final int numSources = 1;
  private static final String[] paramNames = {attributes[6][0],
                                              attributes[7][0]};
  private static final Class[] paramClasses = {Color.class,
                                               Float.class};
  private static final Object[] paramDefaults =
                                { new Color(0,0,0),new Float(0) };
  private static boolean registered = false;

  public ContaPixelsDescriptor()
    {
    super(attributes,modes,numSources,paramNames,
          paramClasses,paramDefaults,null);
    }
```

Novo Operador: contapixels

Passo 4

```
public static void register()
  {
  if (!registered)
    {
    OperationRegistry op =
      JAI.getDefaultInstance().getOperationRegistry();
    ContaPixelsDescriptor desc = new ContaPixelsDescriptor();
    op.registerDescriptor(desc);
    ContaPixelsRIF rif = new ContaPixelsRIF();
    RIFRegistry.register(op,opName,vendorName,rif);
    registered = true;
    }
  }
```

Novo Operador: contapixels

```
public static void main(String[] args)
  {
  ContaPixelsDescriptor.register();
  PlanarImage input = JAI.create("fileload", args[0]);
  int r = Integer.parseInt(args[1]);
  int g = Integer.parseInt(args[2]);
  int b = Integer.parseInt(args[3]);
  float t = Float.parseFloat(args[4]);
  Color color = new Color(r,g,b);
  ParameterBlock p = new ParameterBlock();
  p.addSource(input);
  p.add(color);
  p.add(t);
  PlanarImage output = JAI.create("contapixels",p);
  Long count = (Long)output.getProperty("count");
  System.out.println("Existem "+count+
          " pixels com cores semelhantes a "+color);
  }
```

Extras: I/O

Que formatos são suportados?

```
public static void main(String[] args)
   {
   String[] iFormatos = ImageIO.getReaderMIMETypes();
   System.out.println("Leitura: ");
   for(String f:iFormatos)
      {
      System.out.print(f+" ");
      }
   System.out.println("\nGravação: ");
   String[] oFormatos = ImageIO.getWriterMIMETypes();
   for(String f:oFormatos)
      {
      System.out.print(f+" ");
      }
   System.out.println();
   }
```

```
Leitura:
image/jpeg image/png image/x-png image/vnd.wap.wbmp image/gif image/bmp
Gravação:
image/png image/jpeg image/x-png image/vnd.wap.wbmp image/bmp image/gif
```

Compactando Imagens

- Forma mais simples: abrir imagem em formato menos compactado e salvar em formato mais compactado.
- Lembrar sempre que existe perda de informações com JPEG e GIF!

- Método mais inteligente: controlar a compactação.

Como compactar mais?

```
public static void main(String[] args) throws IOException
  {
  // Load the image (it is hard-coded here to make the code
  // simpler).
  String imageFile = "/tmp/folhas.tif";
  BufferedImage i = ImageIO.read(new File(imageFile));
  showImage("Original Image", i);
  // Show results with different compression ratio.
  compressAndShow(i, 0.5f);
  }
```

Veja mais em **http://www.lac.inpe.br/JIPCookbook**

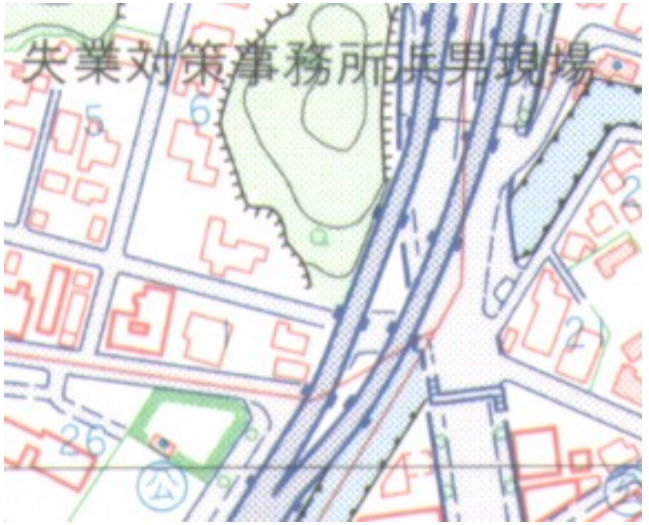
Como compactar mais?

```
public static void compressAndShow(BufferedImage image, float quality)
  throws IOException
  {
  // Get a ImageWriter for jpeg format.
  Iterator<ImageWriter> writers = ImageIO.getImageWritersBySuffix("jpeg");
  if (!writers.hasNext())
    throw new IllegalStateException("No writers found");
  ImageWriter writer = (ImageWriter) writers.next();
  // Create the ImageWriteParam to compress the image.
  ImageWriteParam param = writer.getDefaultWriteParam();
  param.setCompressionMode(ImageWriteParam.MODE_EXPLICIT);
  param.setCompressionQuality(quality);
  // The output will be a ByteArrayOutputStream (in memory)
  ByteArrayOutputStream bos = new ByteArrayOutputStream(32768);
  ImageOutputStream ios = ImageIO.createImageOutputStream(bos);
  writer.setOutput(ios);
  writer.write(null, new IIOImage(image, null, null), param);
  ios.flush(); // otherwise the buffer size will be zero!
  // From the ByteArrayOutputStream create a RenderedImage.
  ByteArrayInputStream in = new ByteArrayInputStream(bos.toByteArray());
  RenderedImage out = ImageIO.read(in);
  int size = bos.toByteArray().length;
  showImage("Compressed to " + quality + ": " + size + " bytes", out);
  }
```

Veja mais em **http://www.lac.inpe.br/JIPCookbook**

Como compactar mais?

```
private static void showImage(String title,RenderedImage image)
  {
  JFrame f = new JFrame(title);
  f.getContentPane().add(new DisplayJAI(image));
  f.pack();
  f.setVisible(true);
  f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
```



Original

0.5

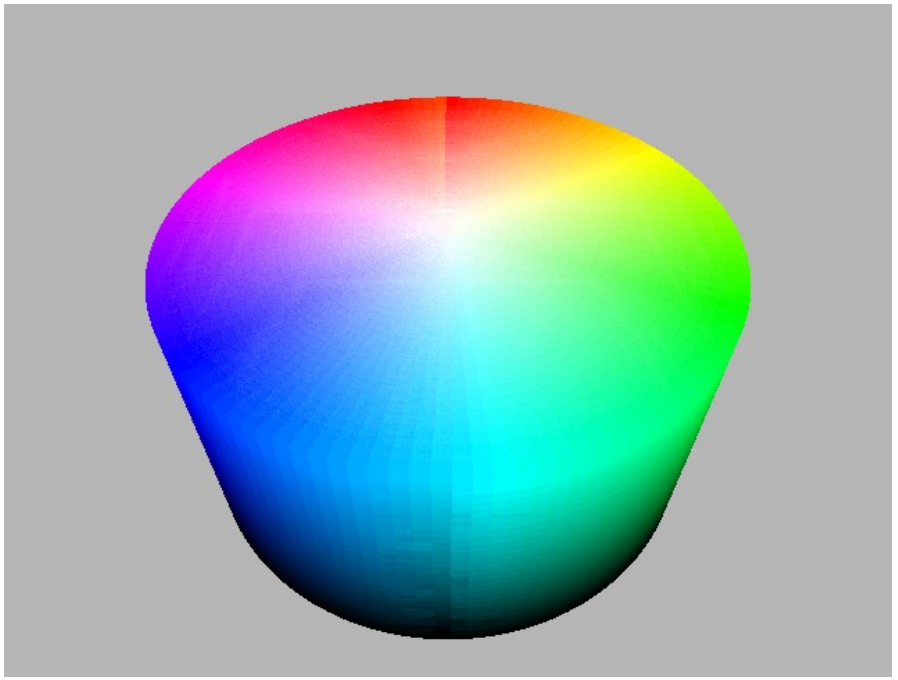
0.1

Veja mais em **http://www.lac.inpe.br/JIPCookbook**

Extras: Pixels

Mini-aplicação: Conversão RGB ↔ IHS

- IHS: *Intensity, Hue and Saturation*.
- Cores representadas por:
  - Intensity: brilho percebido da cor, 0 a 100% (preto = 0%).
  - Hue (croma): cor “bruta”, 0 a 359 graus (0 graus = vermelho).
  - Saturation: intensidade da cor, 0 a 100% (branco = 100%).
- Variantes: HSV, HSB, HSL, etc.



Mini-aplicação: Conversão RGB ↔ IHS

- Integração de imagens de diferentes sensores
  - RGB → IHS, substitui banda I por banda de maior resolução, converte novamente IHS → RGB
- Manipulação de contraste e brilho
  - RGB → IHS, manipula brilho e contraste da banda I, converte novamente IHS → RGB

- Nosso exemplo: converte RGB → IHS, substitui I e S por 100% constantes, reconverte para RGB.

Mini-aplicação: Conversão RGB ↔ IHS

```
public class RGBtoIHS
  {
  public static void main(String[] args)
    {
    PlanarImage imagem = JAI.create("fileload", args[0]);
    // Converte para o modelo de cores IHS.
    IHSColorSpace ihs = IHSColorSpace.getInstance();
    ColorModel modeloIHS =
       new ComponentColorModel(ihs,
                               new int []{8,8,8},
                               false,false,
                               Transparency.OPAQUE,
                               DataBuffer.TYPE_BYTE) ;
    ParameterBlock pb = new ParameterBlock();
    pb.addSource(imagem);
    pb.add(modeloIHS);
    RenderedImage imagemIHS  = JAI.create("colorconvert", pb);
```

Mini-aplicação: Conversão RGB ↔ IHS

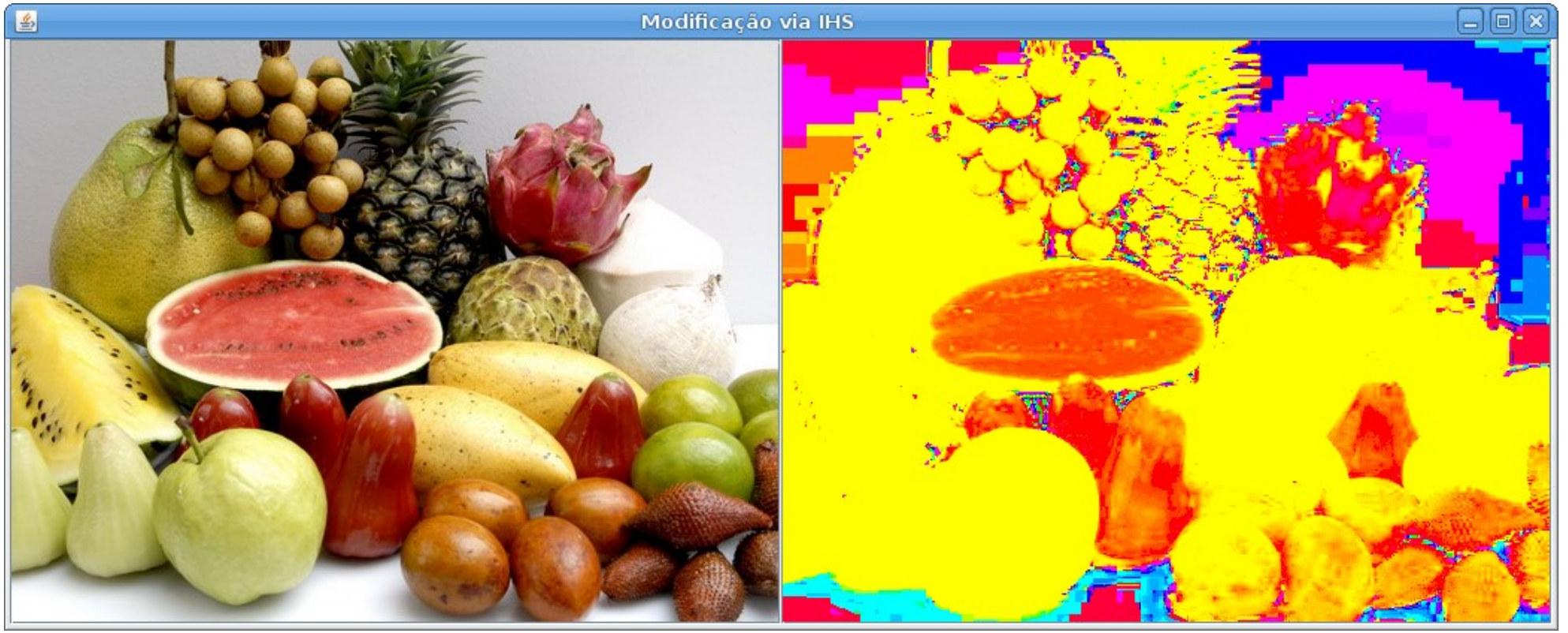
```
// Extraímos as bandas I, H e S.
RenderedImage[] bandas = new RenderedImage[3];
for(int band=0;band<3;band++)
  {
  pb = new ParameterBlock();
  pb.addSource(imagemIHS);
  pb.add(new int[]{band});
  bandas[band] = JAI.create("bandselect",pb);
  }
// Criamos bandas constantes para as bandas I e S.
pb = new ParameterBlock();
pb.add((float)imagem.getWidth());
pb.add((float)imagem.getHeight());
pb.add(new Byte[]{(byte)255});
RenderedImage novaIntensidade = JAI.create("constant",pb);
pb = new ParameterBlock();
pb.add((float)imagem.getWidth());
pb.add((float)imagem.getHeight());
pb.add(new Byte[]{(byte)255});
RenderedImage novaSaturação = JAI.create("constant",pb);
```

Mini-aplicação: Conversão RGB ↔ IHS

```
// Juntamos as bandas H e as I e S constantes.
// Devemos passar um RenderingHint que indica que o modelo
// de cor IHS será usado.
ImageLayout imageLayout = new ImageLayout();
imageLayout.setColorModel(modeloIHS);
imageLayout.setSampleModel(imagemIHS.getSampleModel());
RenderingHints rendHints =
  new RenderingHints(JAI.KEY_IMAGE_LAYOUT,imageLayout);
pb = new ParameterBlock();
pb.addSource(novaIntensidade);
pb.addSource(bandas[1]);
pb.addSource(novaSaturação);
RenderedImage imagemIHSModificada =
  JAI.create("bandmerge", pb, rendHints);
// Convertemos de volta para RGB.
pb = new ParameterBlock();
pb.addSource(imagemIHSModificada);
pb.add(imagem.getColorModel()); // Imagem original era em RGB!
RenderedImage imagemFinal = JAI.create("colorconvert", pb);
```

Mini-aplicação: Conversão RGB ↔ IHS

```
    JFrame frame = new JFrame("Modificação via IHS");
    frame.add(new DisplayTwoSynchronizedImages(imagem,imagemFinal));
    frame.pack();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
    }
}
```



Extras: Display

Mini-aplicação: Visualizador com Ícone

- Mostra imagens grandes com pequeno ícone e *pan*.

```
public class DisplayThumbnail extends DisplayJAI
                              implements MouseMotionListener,MouseListener
  {
  private PlanarImage originalImage;
  private float scale;
  private int imageXTiles,imageYTiles;
  private int imageTileWidth,imageTileHeight;
  private int imageWidth,imageHeight;
  private int visibleRegionWidth,visibleRegionHeight;
  private int thumbWidth,thumbHeight;
  // The size of the border around the thumbnail.
  private final int border = 10;
  // The scaled viewport (dimensions are scaled/translated by the border).
  private Rectangle2D scaledViewport;
  private Color viewportColor;
  // Colors to be used when the mouse is/isn't over the viewport.
  private static Color viewportOn = new Color(120,255,120);
  private static Color viewportOff = new Color(0,192,0);
  // Coordinates obtained when we click (press) the mouse button to start
  // dragging the viewport.
  private int lastX,lastY;
  // Those coordinates represent the region where we can safely drag the
  // viewport without "falling outside" the image boundaries.
  private int minValidX,minValidY,maxValidX,maxValidY;
```

Mini-aplicação: Visualizador com Ícone

```
public DisplayThumbnail(PlanarImage image,float scale,
                        int width,int height)
  {
  this.scale = scale;
  originalImage = image;
  visibleRegionWidth = width;     visibleRegionHeight = height;
  // Get some stuff about the image.
  imageXTiles = image.getNumXTiles();
  imageYTiles = image.getNumYTiles();
  imageTileWidth = image.getTileWidth();
  imageTileHeight = image.getTileHeight();
  imageWidth = image.getWidth();    imageHeight = image.getHeight();
  // Must create a thumbnail image using that scale.
  ParameterBlock pb = new ParameterBlock();
  pb.addSource(image);
  pb.add(scale);  pb.add(scale);  pb.add(0.0F);  pb.add(0.0F);
  pb.add(new InterpolationNearest());
  PlanarImage thumbnail = JAI.create("scale", pb, null);
  // Let's get the width and height of the thumbnail.
  thumbWidth = thumbnail.getWidth();
  thumbHeight = thumbnail.getHeight();
```

Mini-aplicação: Visualizador com Ícone

```
    // Now let's add a border.
    pb = new ParameterBlock();
    pb.addSource(thumbnail);
    pb.add(new Integer(border)); pb.add(new Integer(border));
    pb.add(new Integer(border)); pb.add(new Integer(border));
    pb.add(new BorderExtenderConstant(new double[]{0,0,128}));
    thumbnail = JAI.create("border",pb);
    // Shift the image to the original position
    pb = new ParameterBlock();
    pb.addSource(thumbnail); pb.add(1.0f*border); pb.add(1.0f*border);
    thumbnail = JAI.create("translate",pb,null);
    // Use this thumbnail as the image for the DisplayJAI component.
    set(thumbnail);
    // We'd like to listen to mouse movements.
    addMouseMotionListener(this);
    addMouseListener(this);
    // Initially the scaled viewport will be positioned at border,border.
    scaledViewport =
      new Rectangle2D.Float(border,border,width*scale,height*scale);
    // We assume that the mouse is off the viewport.
    viewportColor = viewportOff;
    }
```

Mini-aplicação: Visualizador com Ícone

```
public synchronized void paintComponent(Graphics g)
  {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;
  // Paint the tile grid.
  g2d.setColor(Color.YELLOW);
  g2d.setComposite(AlphaComposite.getInstance(
                                AlphaComposite.SRC_OVER,0.5f));
  float x1,x2,y1,y2;
  // Vertical tiles' boundaries.
  x1 = x2 = border;
  y1 = border; y2 = border+thumbHeight;
  for(int tx=0;tx<=imageXTiles;tx++)
    {
    g2d.drawLine((int)x1,(int)y1,(int)x2,(int)y2);
    x1 += imageTileWidth*scale;
    x2 += imageTileWidth*scale;
    }
```

Mini-aplicação: Visualizador com Ícone

```
    // Horizontal tiles' boundaries.
    x1 = border; x2 = border+thumbWidth;
    y1 = y2 = border;
    for(int ty=0;ty<=imageYTiles;ty++)
      {
      g2d.drawLine((int)x1,(int)y1,(int)x2,(int)y2);
      y1 += imageTileHeight*scale;
      y2 += imageTileHeight*scale;
      }
    // Paint a red border.
    g2d.setColor(Color.RED);
    g2d.drawRect(border,border,thumbWidth,thumbHeight);
    // Paint the viewport.
    g2d.setColor(viewportColor);
    g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,1f));
    Stroke stroke = new BasicStroke(2f);
    g2d.setStroke(stroke);
    g2d.draw(scaledViewport);
    }
```

Mini-aplicação: Visualizador com Ícone

```
public void mouseMoved(MouseEvent e)
  {
  int x = e.getX(); int y = e.getY();
  // Ignore events outside the border.
  if ((x < border) || (y < border) ||
      (x > border+thumbWidth) || (y > border+thumbHeight))
    return;
  // Are we inside the viewport rectangle ?
  if (scaledViewport.contains(x,y)) viewportColor = viewportOn;
  else viewportColor = viewportOff;
  // Hopefully it will repaint only the needed section.
  Rectangle repaintBounds =
    new Rectangle((int)scaledViewport.getX()-5,
                  (int)scaledViewport.getY()-5,
                  (int)scaledViewport.getWidth()+10,
                  (int)scaledViewport.getHeight()+10);
  repaint(repaintBounds);
  }
```

Mini-aplicação: Visualizador com Ícone

```
public void mousePressed(MouseEvent e)
  {
  // Store the new dragging starting points.
  lastX = e.getX(); lastY = e.getY();
  // Calculate the new window w/ viewport movements.
  minValidX = lastX-(int)scaledViewport.getX()+border;
  minValidY = lastY-(int)scaledViewport.getY()+border;
  maxValidX = border+thumbWidth-(int)scaledViewport.getWidth()+
                  (lastX-(int)scaledViewport.getX());
  maxValidY = border+thumbHeight-(int)scaledViewport.getHeight()+
                  (lastY-(int)scaledViewport.getY());
  }
public void mouseDragged(MouseEvent e)
  {
  int x = e.getX(); int y = e.getY();
  if (x > maxValidX) x = maxValidX-1;
  if (x < minValidX) x = minValidX;
  if (y > maxValidY) y = maxValidY-1;
  if (y < minValidY) y = minValidY;
  if ((x >= minValidX) && (y >= minValidY) &&
      (x <= maxValidX) && (y <= maxValidY))
    {
    updateLocation(x, y);  lastX = x; lastY = y;
    }
  }
```

Mini-aplicação: Visualizador com Ícone

```
public void updateLocation(int x,int y)
  {
  // Store the approximate region where the viewport was before the change.
  Rectangle initBounds =
    new Rectangle((int)scaledViewport.getX()-5,
                  (int)scaledViewport.getY()-5,
                  (int)scaledViewport.getWidth()+10,
                  (int)scaledViewport.getHeight()+10);
  // Recalculate new position for the viewport, based on mouse coordinates.
  double origX = scaledViewport.getX()+x-lastX;
  double origY = scaledViewport.getY()+y-lastY;
  // Reposition the viewport.
  scaledViewport.setFrame(origX,origY,
        scaledViewport.getWidth(),scaledViewport.getHeight());
  // Store the approximate region where the viewport is after the change.
  Rectangle finalBounds =
    new Rectangle((int)scaledViewport.getX()-5,
                  (int)scaledViewport.getY()-5,
                  (int)scaledViewport.getWidth()+10,
                  (int)scaledViewport.getHeight()+10);
  // Repaint only that section.
  repaint(finalBounds.union(initBounds));
  }
```

Mini-aplicação: Visualizador com Ícone

```
public PlanarImage getImage()
  {
  // Get the boundaries in the original image coordinates.
  float fromX  = (float)Math.round((scaledViewport.getX()-border)/scale);
  float fromY  = (float)Math.round((scaledViewport.getY()-border)/scale);
  float width  = (float)Math.round(scaledViewport.getWidth()/scale);
  float height = (float)Math.round(scaledViewport.getWidth()/scale);
  // Fix rounding errors to avoid exceptions on the crop.
  fromX = Math.min(fromX,(imageWidth-visibleRegionWidth));
  fromY = Math.min(fromY,(imageHeight-visibleRegionHeight));
  // Create a ParameterBlock with information for the cropping.
  ParameterBlock pb = new ParameterBlock();
  pb.addSource(originalImage);
  pb.add(fromX); pb.add(fromY); pb.add(width); pb.add(height);
  // Create the output image by cropping the input image.
  PlanarImage output = JAI.create("crop",pb,null);
  // Translate the image origin.
  pb = new ParameterBlock();
  pb.addSource(output);
  pb.add(-fromX);  pb.add(-fromY);
  // Create the output image by translating itself.
  return JAI.create("translate",pb,null);
  }
```

Mini-aplicação: Visualizador com Ícone

```
public Rectangle getCroppedImageBounds()
  {
  int fromX  = (int)Math.round((scaledViewport.getX()-border)/scale);
  int fromY  = (int)Math.round((scaledViewport.getY()-border)/scale);
  int width  = (int)Math.round(scaledViewport.getWidth()/scale);
  int height = (int)Math.round(scaledViewport.getWidth()/scale);
  return new Rectangle(fromX,fromY,width,height);
  }

public Rectangle getViewportBounds()
  {
  Rectangle temp = scaledViewport.getBounds();
  temp.setBounds((int)temp.getX()-border,(int)temp.getY()-border,
                 (int)temp.getWidth(),(int)temp.getHeight());
  return temp;
  }
```

Mini-aplicação: Visualizador com Ícone

```
public class DisplayThumbnailApp extends JFrame implements MouseMotionListener
  {
  private DisplayThumbnail dt;   private DisplayJAI dj; private JLabel world,view;

  public DisplayThumbnailApp(PlanarImage image,int dWidth,int dHeight)
    {
    super("Interactive Thumbnail Example");
    SpringLayout layout = new SpringLayout();
    Container contentPane = getContentPane();
    contentPane.setLayout(layout);
    dt = new DisplayThumbnail(image,0.05f,dWidth,dHeight);
    dt.addMouseMotionListener(this);
    dj = new DisplayJAI(dt.getImage());
    dj.setPreferredSize(new Dimension(dWidth,dHeight));
    dj.setMinimumSize(new Dimension(dWidth,dHeight));
    dj.setMaximumSize(new Dimension(dWidth,dHeight));
    JPanel borderDisplay = new JPanel(new BorderLayout());
    borderDisplay.add(dj);
    borderDisplay.setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED));
    JLabel worldL = new JLabel("World: ");    world = new JLabel("");
    JLabel viewL = new JLabel("Thumb: ");     view = new JLabel("");
    contentPane.add(dt); contentPane.add(borderDisplay);
    contentPane.add(worldL);   contentPane.add(world);
    contentPane.add(viewL);    contentPane.add(view);
    (organização do layout removida)
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setResizable(false);     pack();      setVisible(true);
    }
```

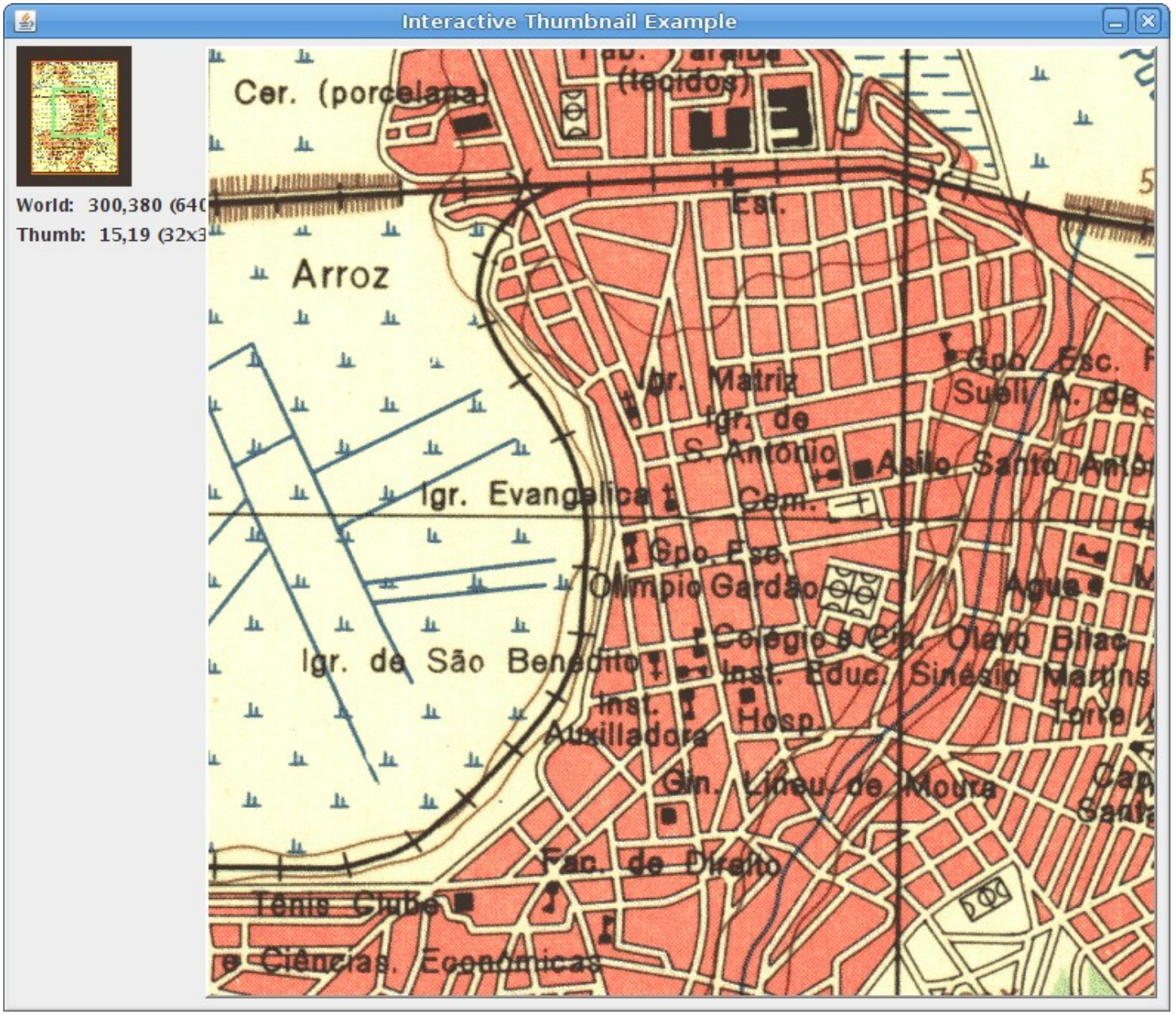
Mini-aplicação: Visualizador com Ícone

```
  public void mouseDragged(MouseEvent e)
    {
    dj.set(dt.getImage());
    // Gets some information about the viewport and cropped image.
    Rectangle crop = dt.getCroppedImageBounds();
    Rectangle viewp = dt.getViewportBounds();
    // Change the labels' contents with this information.
    world.setText(""+crop.x+","+crop.y+
       " ("+crop.width+"x"+crop.height+")");
    view.setText(""+viewp.x+","+viewp.y+
       " ("+viewp.width+"x"+viewp.height+")");
    }

  public void mouseMoved(MouseEvent e) { }

  public static void main(String[] args)
    {
    PlanarImage image = JAI.create("fileload", args[0]);
    new DisplayThumbnailApp(image,640,640);
    }
```

Mini-aplicação: Visualizador com Ícone



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...

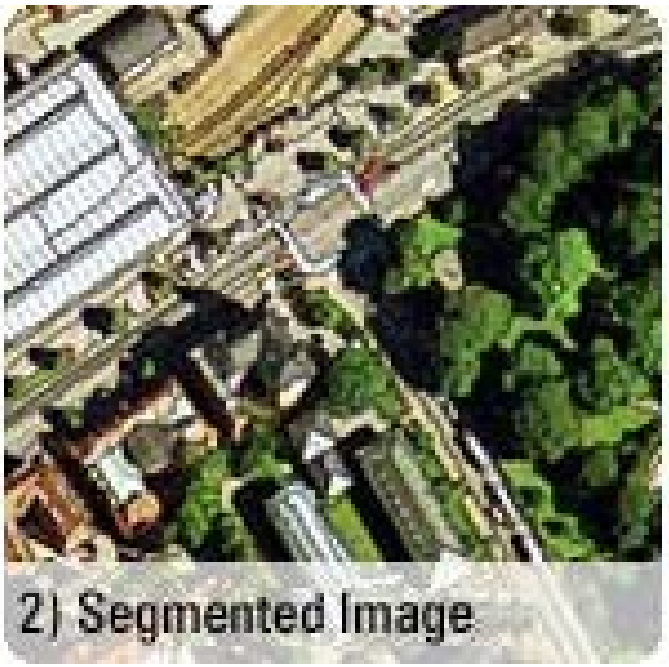
- http://www.lac.inpe.br/~rafael.santos
  - http://www.lac.inpe.br/~rafael.santos/piapresentacoes.jsp
  - http://www.lac.inpe.br/JIPCookbook/index.jsp
- http://www.lac.inpe.br/ELAC/index.jsp

Outros Tópicos

Tópicos para Pesquisa e Desenvolvimento

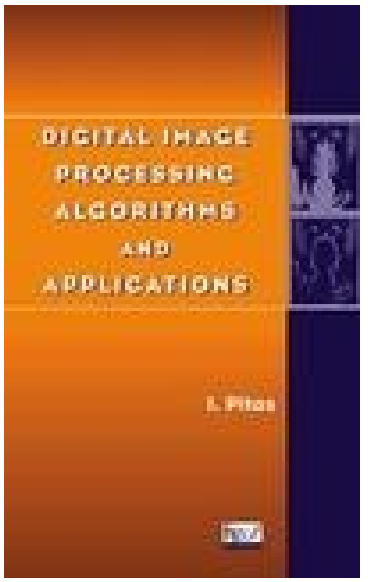
- Processamento por pixels x processamento por regiões
  - Imagens de alta resolução
  - Novos algoritmos de classificação
  - Modelagem de conhecimento
  - Modelos de mistura
  - Inteligência artificial

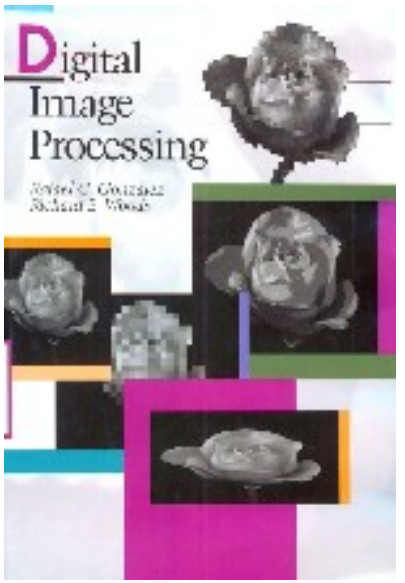
http://gras.ku.dk/software/ecognition.htm



Para saber mais...

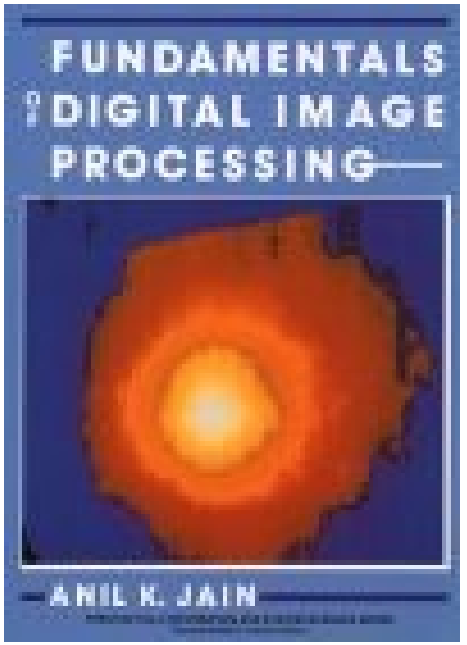
- **http://www.lac.inpe.br/JIPCookbook**
- Introductory Digital Image Processing: A Remote Sensing Perspective (*John R. Jensen*)
- Digital Image Processing Algorithms and Applications (*Ioannis Pitas*)
- Digital Image Processing (*Rafael C. Gonzalez, Richard E. Woods*)

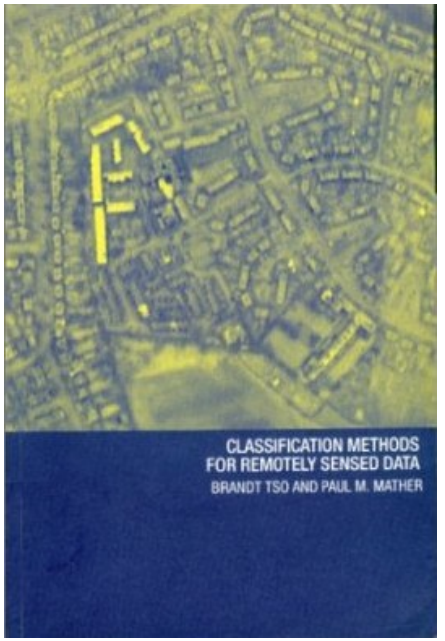


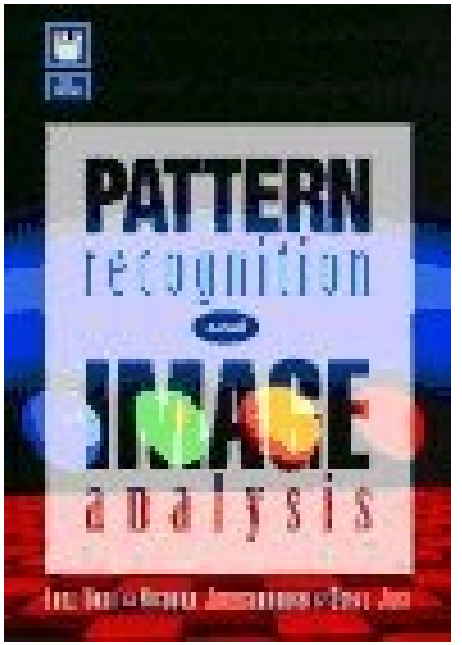


Para saber mais...

- Fundamentals of Digital Image Processing (*Anil K. Jain*)
- Classification Methods for Remotely Sensed Data (*Brandt Tso, Paul M. Mather*)
- Pattern Recognition and Image Analysis (*Earl Gose, Richard Johnsonbaugh, Steve Jost*)







Para saber mais...

- Fuzzy Algorithms: With Applications to Image Processing and Pattern Recognition (*Zheru Chi, H. Yan, Z.R. Chi, Hong Yan, Tuan Pham*)
- The Pocket Handbook of Image Processing Algorithms In C (*Harley R. Myler, Arthur R. Weeks*)
- Intelligence: The Eye, the Brain, and the Computer (*Martin A. Fischler, Oscar Firschein*)