
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

- ***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.

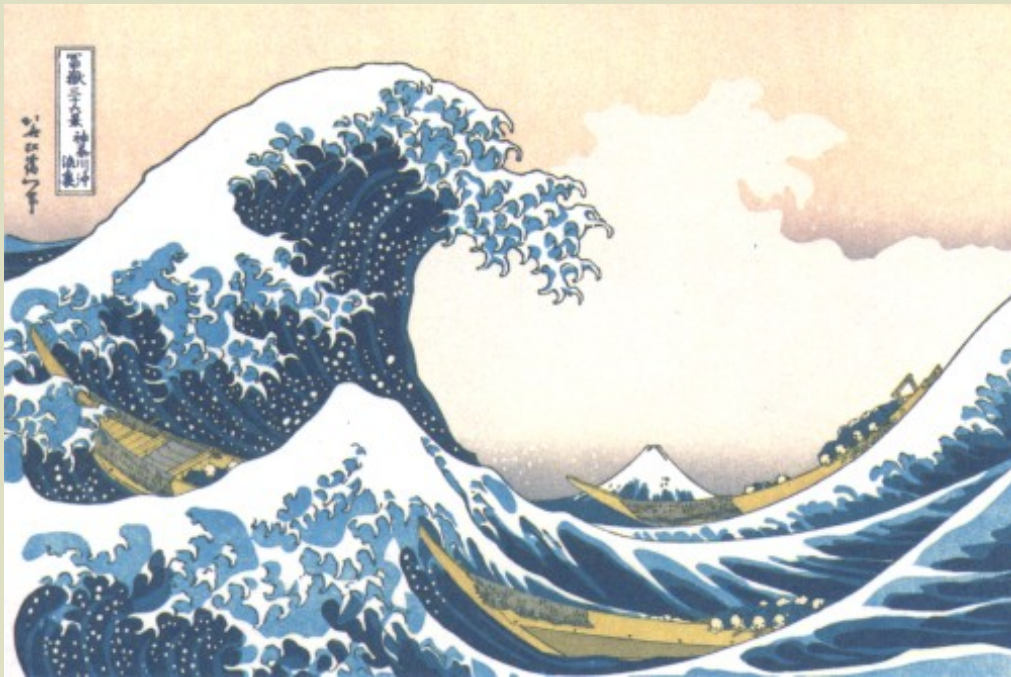
- Apresentar conceitos, técnicas e exemplos básicos de aplicação de processamento de imagens digitais.
- Implementações em Java opcionalmente com a API JAI (Java Advanced Imaging).
- Parte reduzida do livro on-line *Java Image Processing Cookbook* (<http://www.lac.inpe.br/JIPCookbook/index.jsp>).
- **Código!**

Introdução

- **Sensoriamento Remoto:**
 - Geologia (estudo da composição da superfície)
 - Agricultura (determinação da cobertura vegetal)
 - Engenharia Florestal (idem)
 - Cartografia (mapeamento da superfície)
 - Meteorologia
- Medicina e Biologia
- Astronomia (macro) e Física (micro)
- Produção e Controle de Qualidade
- Segurança e Monitoramento
- Documentos, Web, etc.

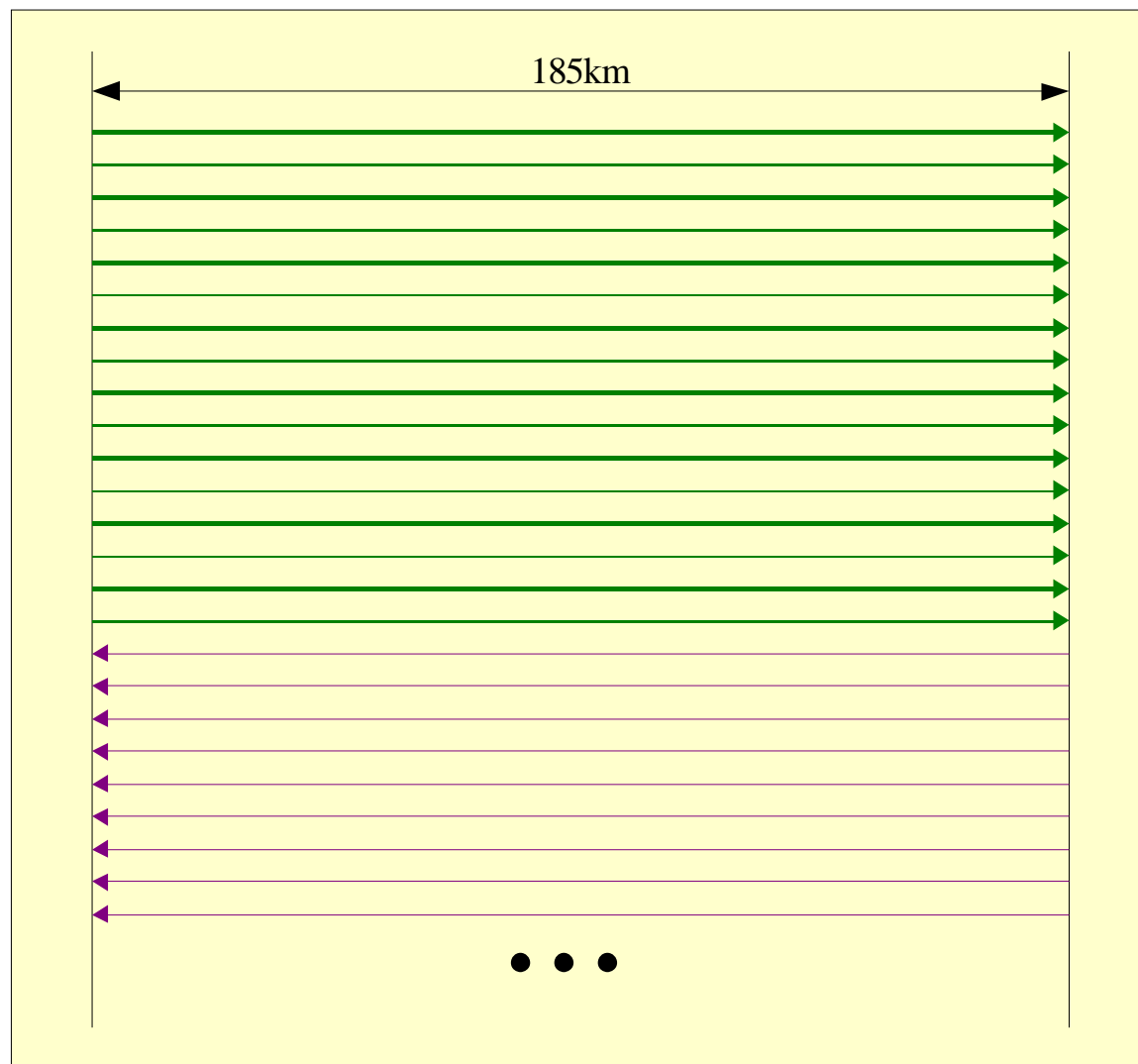
Imagens Digitais

- Imagem = matriz de pixels.
- Pixel = medida, conjunto de medidas ou índice para tabela de valores.
- Metadados: dados adicionais sobre a imagem.



- Sensor(es) medem uma determinada característica em um ponto de um objeto
- Vários pontos são usados para criar uma imagem
 - Pontos são geralmente distribuídos regularmente (*resolução espacial*)
 - Atributos ou características são digitalizadas ou discretizadas (*resolução espectral*)
- Uma imagem é uma matriz regular onde cada elemento pode ter vários atributos associados (bandas)

- Sensor TM (*Thematic Mapper*) do satélite Landsat
 - Dezesesseis sensores paralelos bidirecionais
 - Resolução de 30 metros por pixel
 - 6 bandas (mais uma de outra resolução)
 - Imagem que cobre faixa de 185 km de largura
 - Cada pixel é discretizado com valores entre 0 e 255



Tipos mais comuns

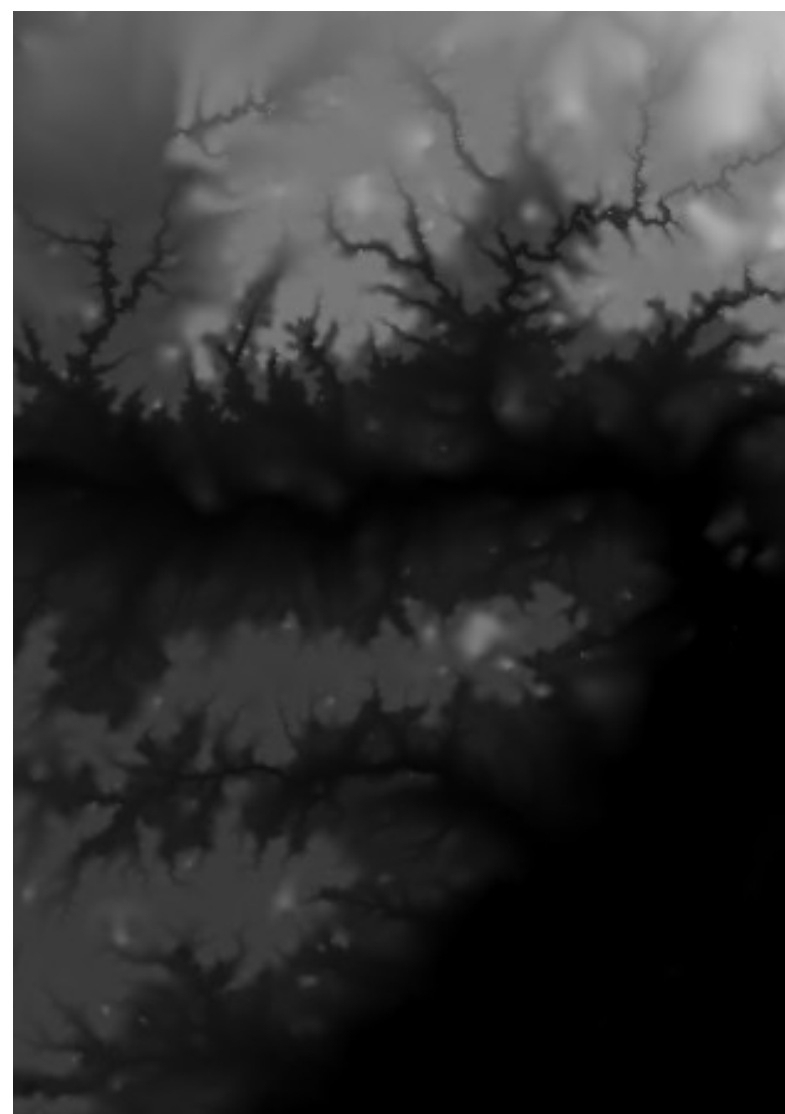
- Câmera Digital
 - 3264x2448 elementos sensores
 - Resolução: não se aplica
 - 3 bandas
 - Cada pixel é discretizado com valores entre 0 e 255

- Scanner
 - Array móvel de elementos sensores
 - Resolução: 2400 DPI ou mais
 - 3 bandas
 - Discretização variável

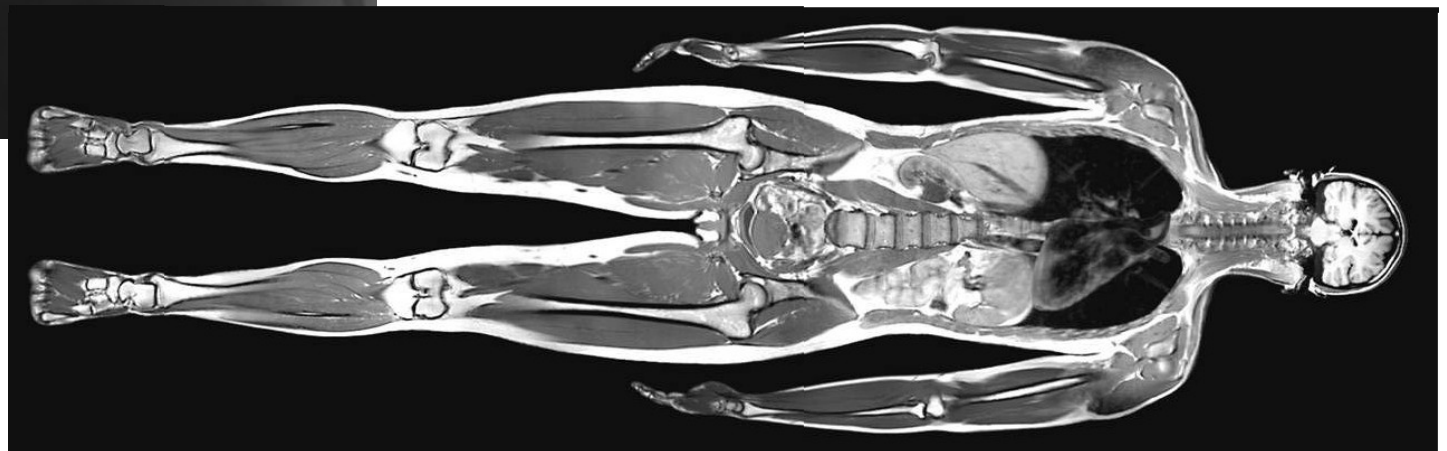
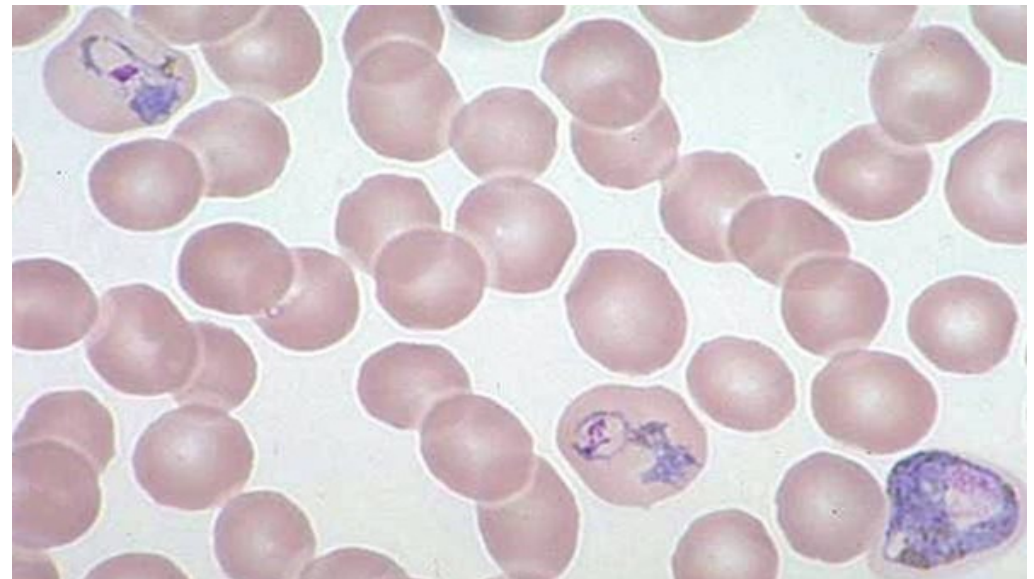
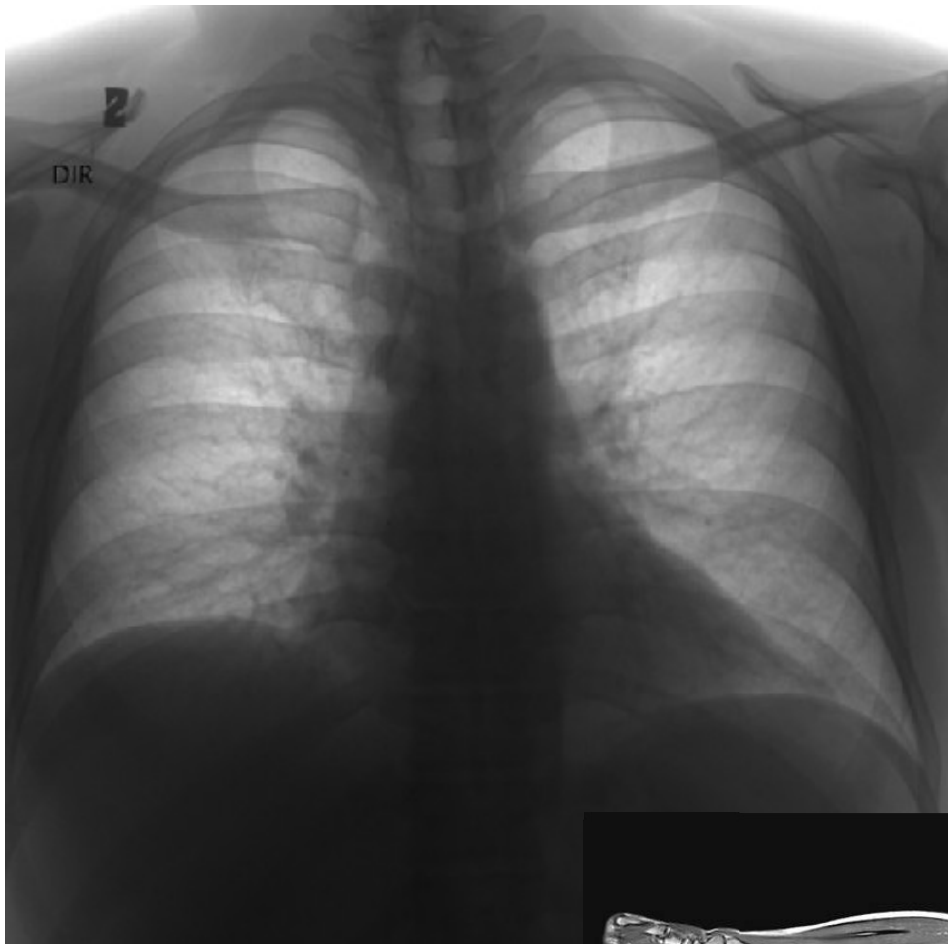


- Não somos limitados à imagens como as de câmeras e scanners!
 - Pixels podem ter mais que três valores associados a eles.
 - Pixels podem ter valores fora do tradicional intervalo $[0, 255]$.
 - Pixels não precisam representar valores inteiros ou positivos!
- Exemplos:
 - Imagens multispectrais e hiperspectrais.
 - Imagens de modelos de terreno, médicas (Raio-X), etc.

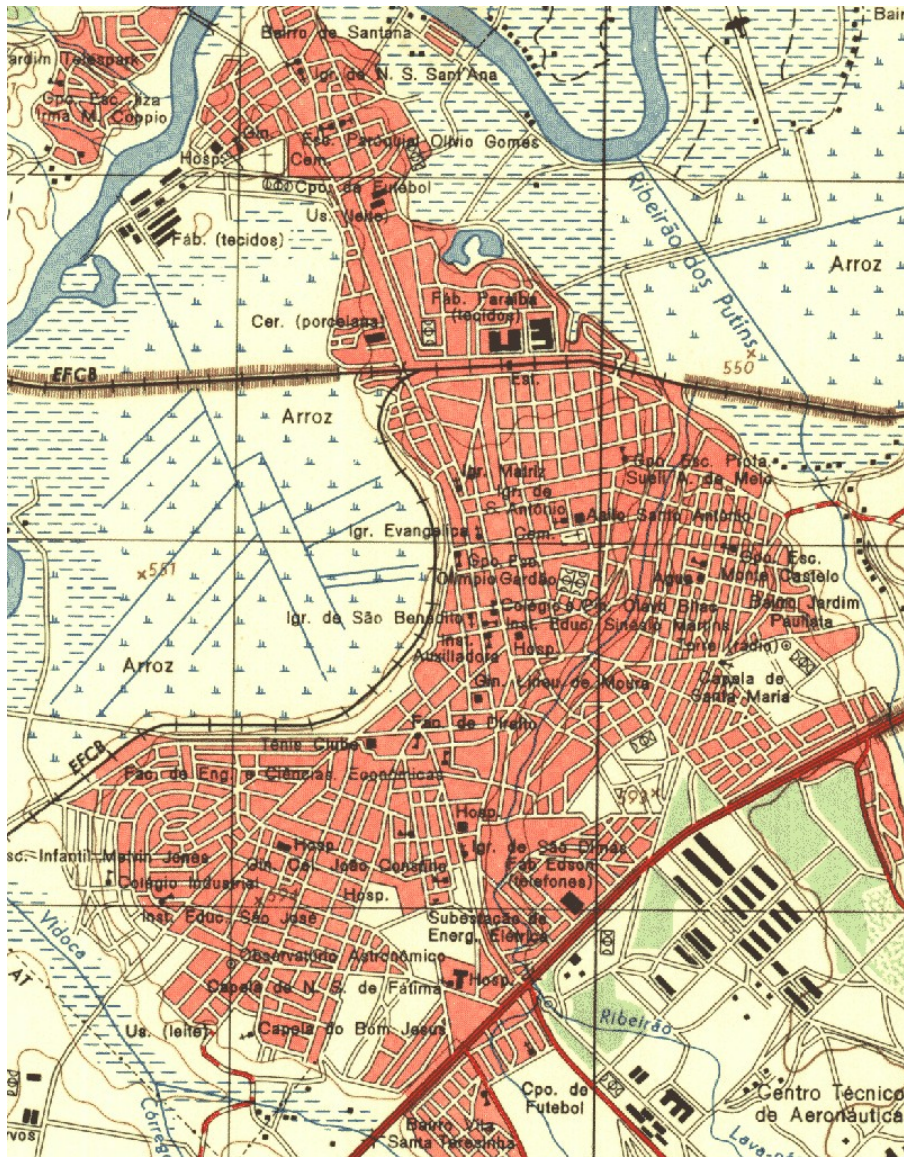
Outros Tipos de Imagens Digitais



Outros Tipos de Imagens Digitais



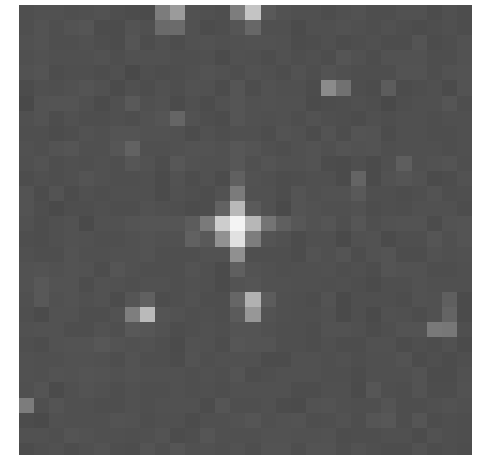
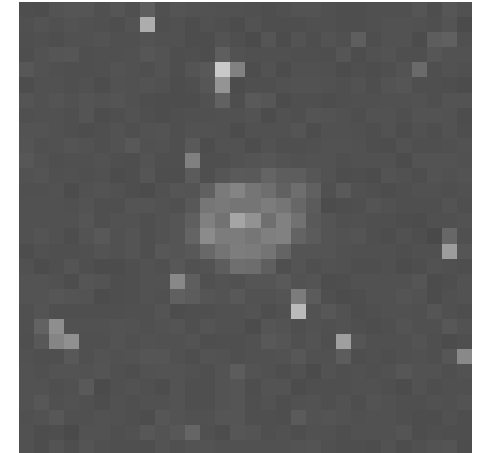
Outros Tipos de Imagens Digitais



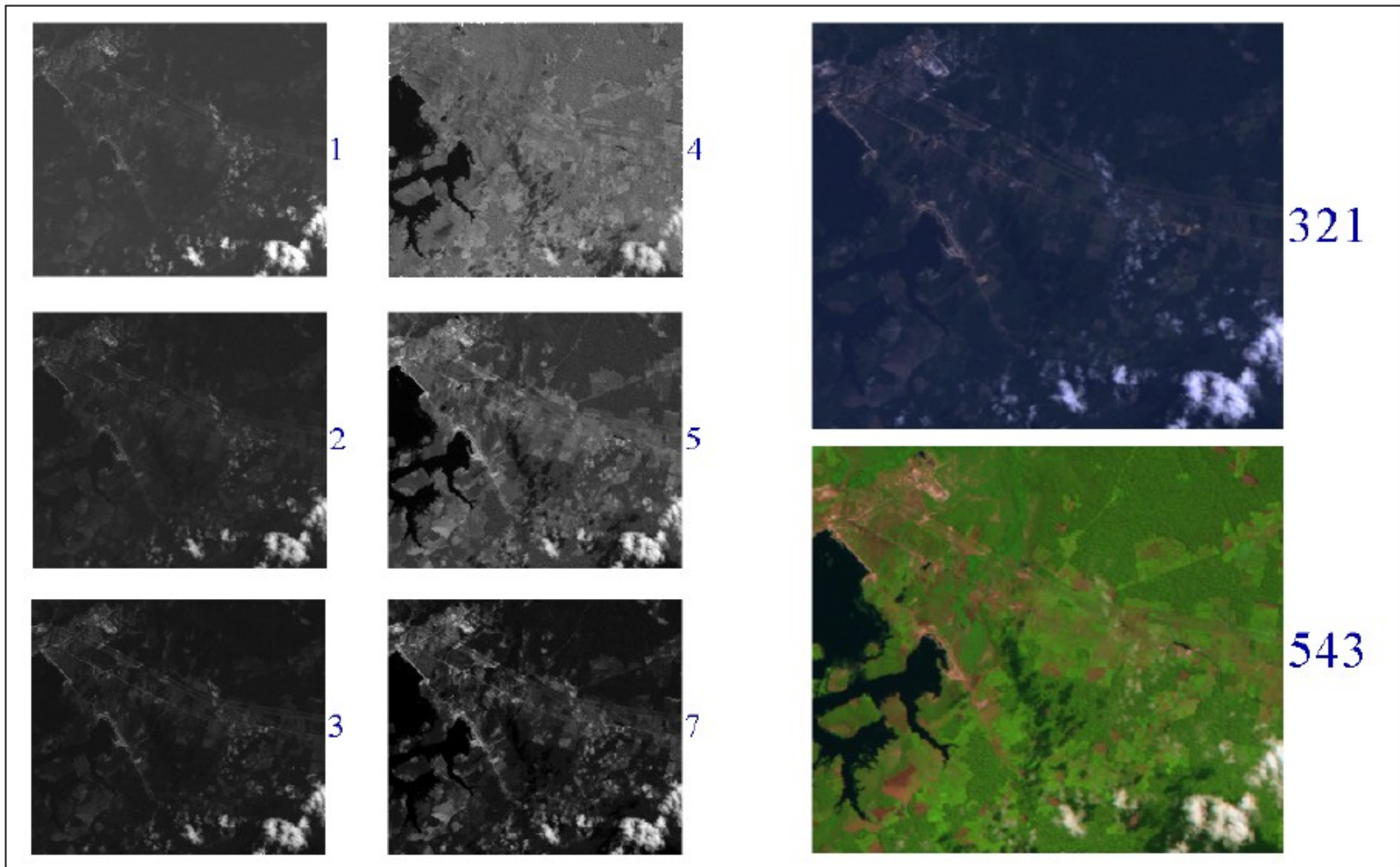
平かな 日常用語 - 216

 そのま、	 先日お話の	 すみません	 しのごよく	 さぞかし	 御無沙汰	 御心配なく
----------	-----------	-----------	-----------	----------	----------	-----------

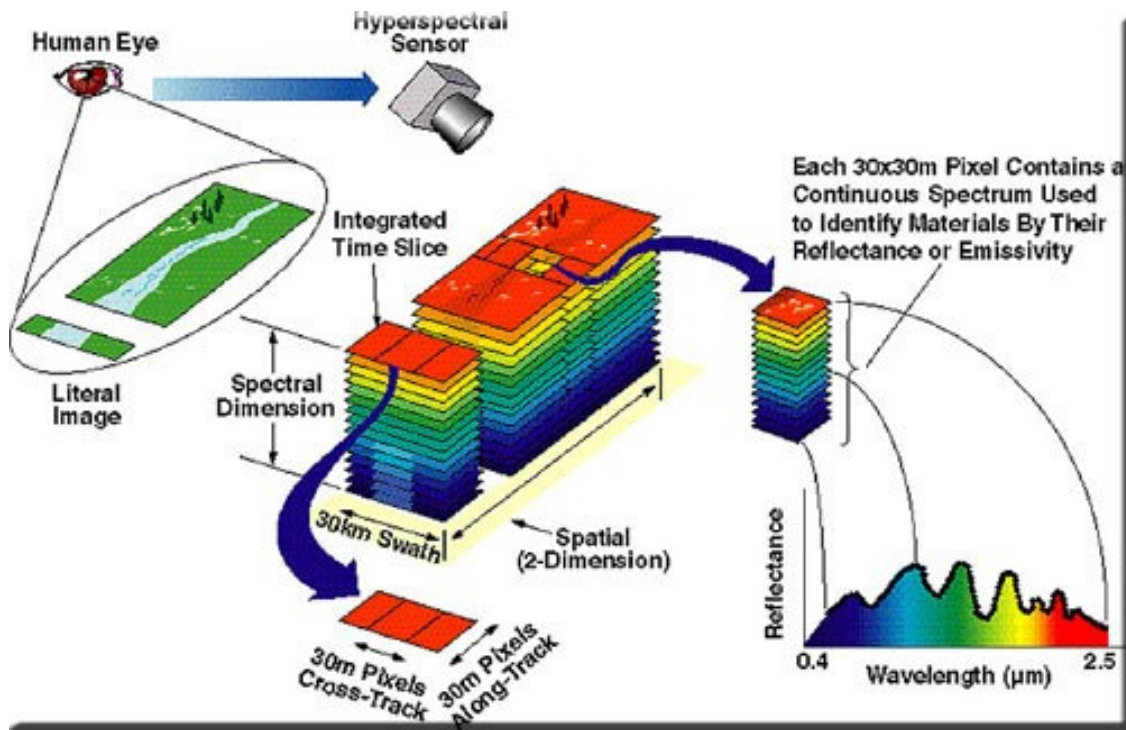
Outros Tipos de Imagens Digitais



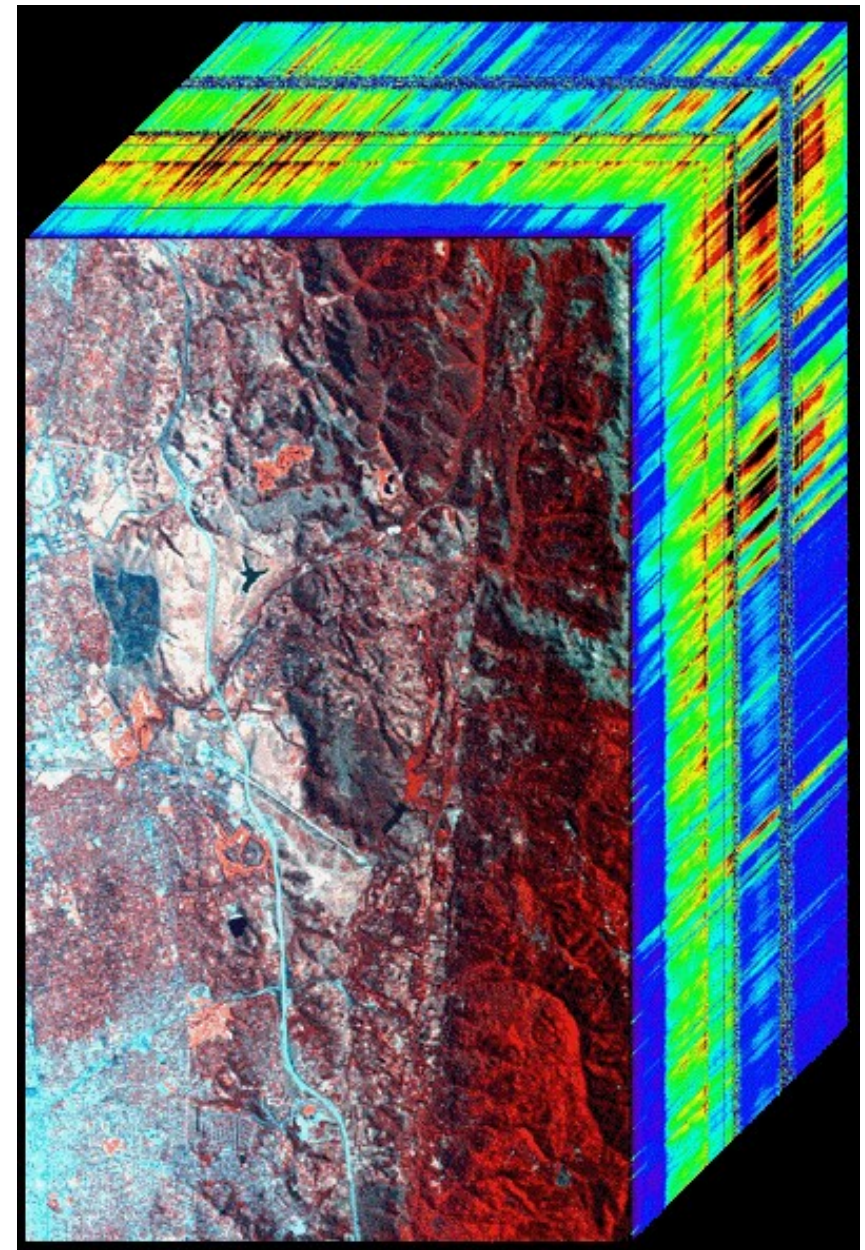
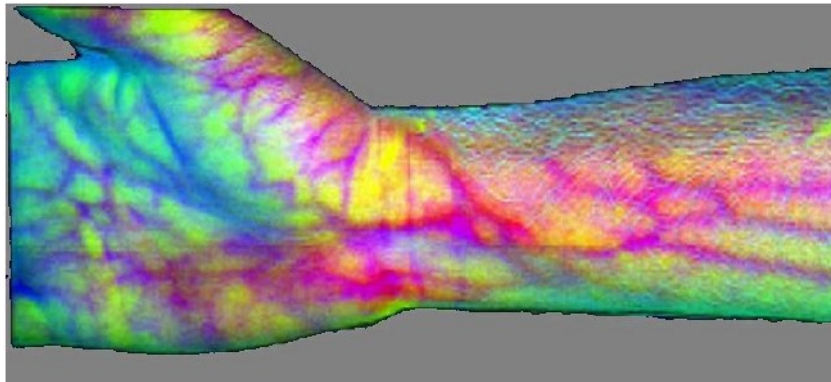
Imagens Digitais: Multiespectrais



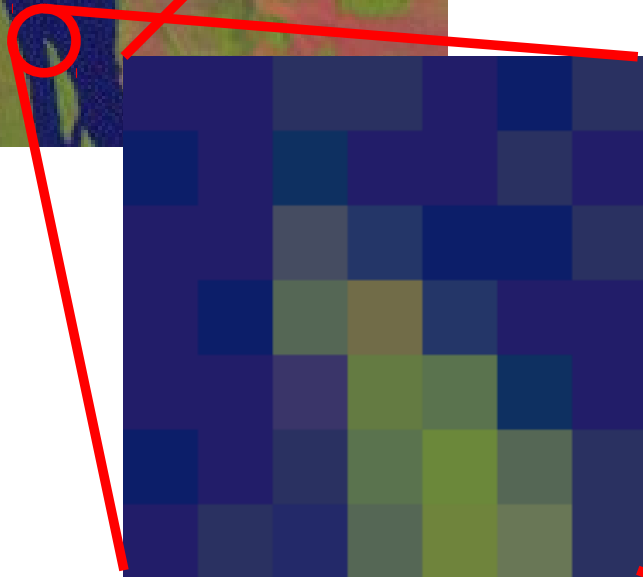
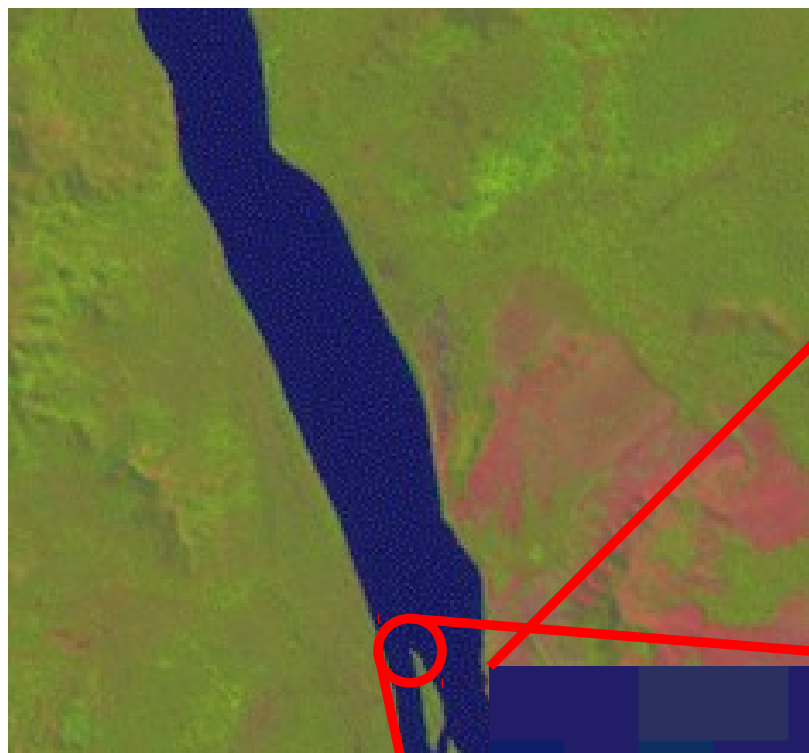
Imagens Digitais: Hiperespectrais



<http://www.cossa.csiro.au/hswwww/Overview.htm>



Imagens e Pixels



34	34	42	42	34	12	42
29	29	49	49	29	30	49
105	105	97	97	105	105	97
12	34	14	34	34	42	34
30	29	48	29	29	49	29
105	105	97	105	105	97	105
34	34	69	36	12	12	42
29	29	76	54	30	30	49
105	105	97	104	105	105	97
34	12	85	113	36	34	34
29	30	103	108	54	29	29
105	105	85	72	104	105	105
34	34	58	100	90	14	34
29	29	53	123	115	48	29
105	105	105	66	78	97	105
12	34	42	90	107	85	42
30	29	49	115	136	103	49
105	105	97	78	58	85	97
34	42	35	85	111	105	42
29	49	41	103	132	119	49
105	97	105	85	60	86	97

- Qual é o tamanho (lado) de um pixel ?
 - NOAA AVHRR (meteorológico): 1.1 km
 - Landsat RBV: 80 m
 - Landsat TM: 30 m para bandas 1-5 e 7, 120 m para banda 6 (térmica)
 - CBERS-2: 260m para WFI, 20m para CCD.
 - SPOT: 20 m para multiespectral, 10 m para pancromático
 - IKONOS: 1 m
 - QuickBird: 60 cm
 - Aéreas (obtidas de vôos com aviões): depende
 - Fotografias digitais: ???

- Menor resolução → Menos detalhes
 - Alguns alvos ou objetos são parcialmente menores do que um pixel: não podem ser identificados facilmente
- Maior resolução → Mais detalhes
 - Alvos ou objetos compostos de vários pixels contíguos (regiões)
 - Quanto maior a resolução, maiores estas regiões
- Mesma área coberta, com maior resolução → Mais pixels (mais informação para processar, imagens maiores)
- Alguns algoritmos e metodologias devem ser aplicados diferentemente dependendo da resolução!

- Quantas cores ou níveis de cinza diferentes um pixel pode assumir ?
 - Mais comum: usar oito bits para cada pixel em cada banda
 - Oito bits = um byte: valor máximo do pixel é 2^8-1 (entre 0 e 255)
 - Outros valores comuns: 12 bits por pixel (entre 0 e 4095)
- Frequentemente imagens são compostas de bandas:
 - Uma banda, 8 bits/pixel: 256 níveis de cinza diferentes
 - Três bandas, 8 bits/pixel: 256^3 cores diferentes (aprox. 16 milhões)
 - Seis bandas, 8 bits/pixel: 256^6 combinações diferentes (aprox. 281 trilhões)

- Reamostragem:
 - Redução do número de pixels para representar uma imagem
 - Causa a redução dos requisitos para processamento
- Quantização:
 - Redução do número de níveis de cinza (ou cores) de uma imagem
 - Pode levar a uma redução do número de bits necessário para representar a imagem → compressão
- Reamostragem e quantização causam a perda de informação
 - Podem ser necessárias...

Efeitos da Reamostragem



Efeitos da Quantização



- Preciso saber Java?
 - Ajuda e muito!
 - Experiência com C++, C#, outras linguagens pode ajudar.
- Todo o código está no livro on-line (<http://www.lac.inpe.br/JIPCookbook>), completo e comentado.

- Popularidade e flexibilidade de Java.
- Temos APIs para representação, visualização e I/O simples de imagens como parte do JSE.
- Temos a API *Java Advanced Imaging* para operações muito mais poderosas, flexíveis e complexas!
- E a questão da performance?
 - Melhor do que esperado!
 - Não estou preocupado com *real time*.
 - Mais valor à clareza e simplicidade de código.

- Vantagens

- Linguagem já conhecida.
- Baixo custo.
- Performance adequada.
- Elegância e clareza de código.
- APIs abrangentes e flexíveis.
- JSE e JEE.

- Desvantagens

- Não é exatamente *WORA!*
- Aplicações nativas sempre tem uma vantagem.
- Problemas potenciais para algumas aplicações.

- Java (Swing) tem classes e operadores básicos.
- *Java Advanced Imaging*
 - API adicional (*download* separado).
 - Projeto do java.net – público mas não totalmente aberto.
- Muitos operadores específicos para processamento de imagens.
- Execução postergada e cadeias de operadores.
- Representação mais poderosa e flexível de imagens (*tiles*).
- Alguns operadores acelerados (implementação nativa).
- **Dúvida: terá apoio da Oracle?**

- JAI não executa imediatamente algumas operações!
- Operações são descritas com operadores e parâmetros, execução é atrasada até que a *tile* em questão seja necessária.
- Alguns operadores não usam *tiles*....
- Algumas operações forçam a execução imediata.

RenderedImage

ColorModel

ColorSpace

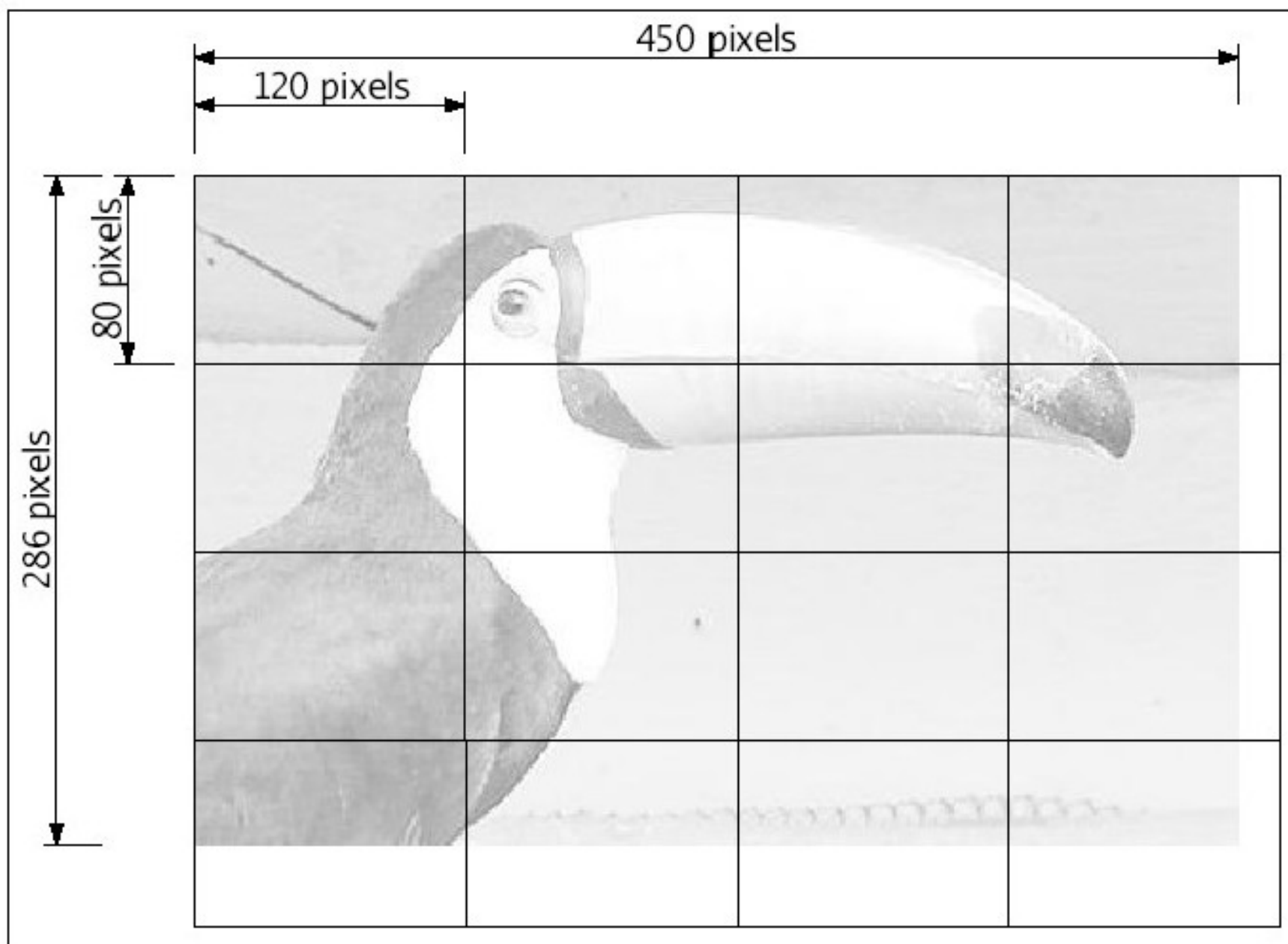
Raster

SampleModel

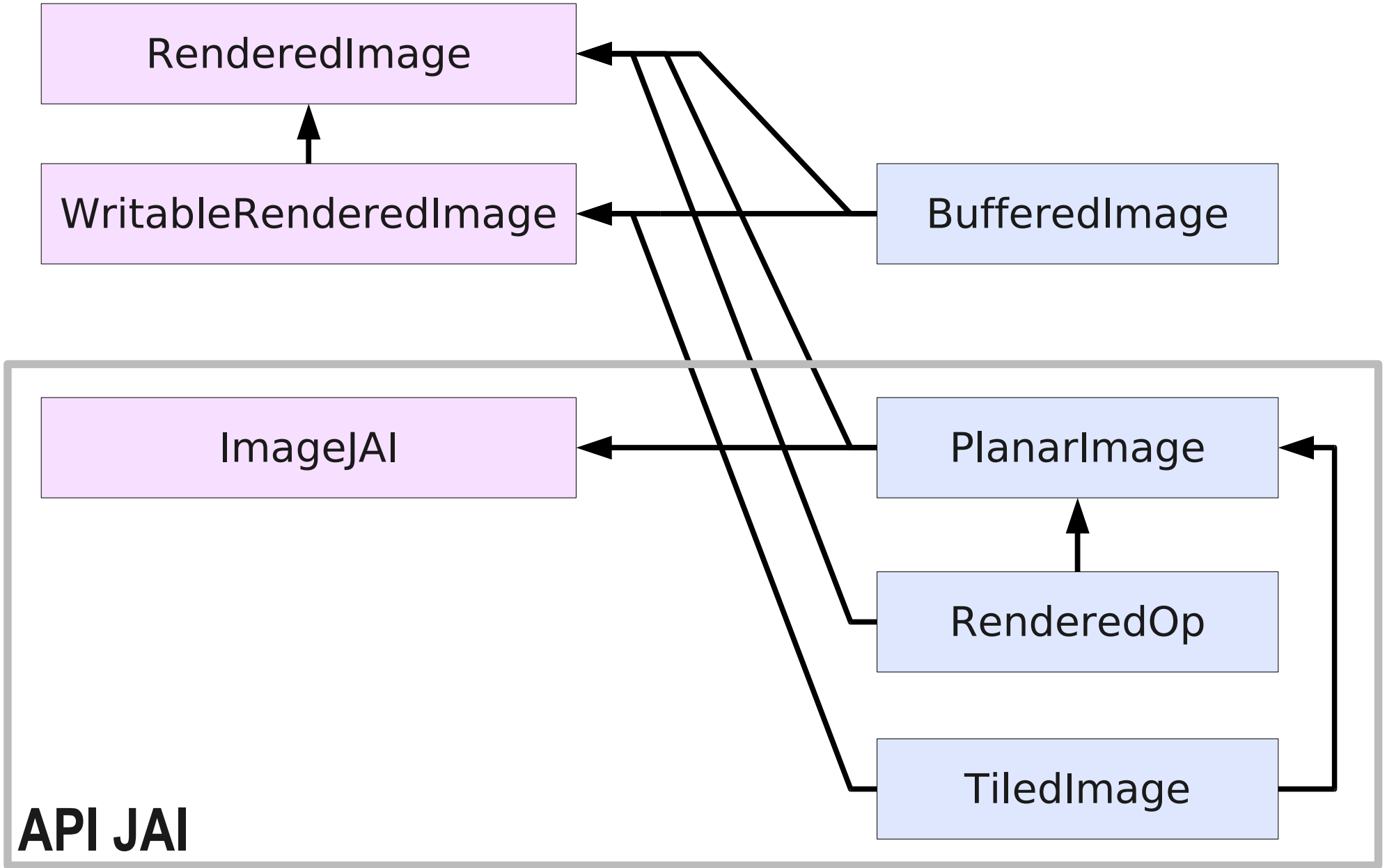
DataBuffer

- Formato de representação na memória é **diferente** de formato de arquivo!
- Existem limitações mútuas.

Representação de Imagens: TiledImage (JAI)



Representação de Imagens



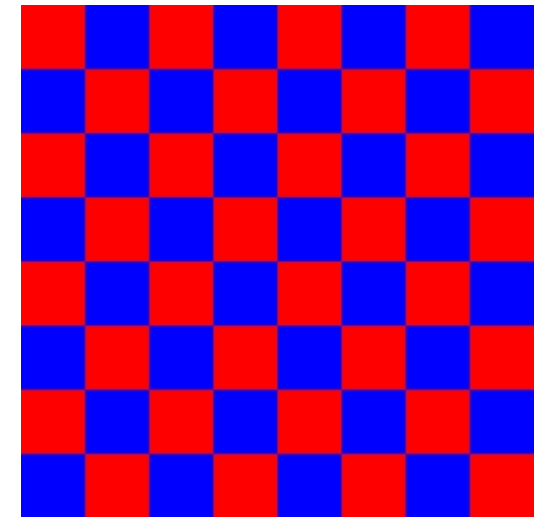
API JAI

1. Criamos instância de `BufferedImage`.
2. Criamos instância de `WritableRaster` associada à `BufferedImage`.
3. Manipulamos os pixels do `WritableRaster`.

Criando Imagens (sem JAI)



```
public static void main(String[] args) throws IOException
{
    int width = 256;
    int height = 256;
    BufferedImage image = new
        BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);
    WritableRaster raster = image.getRaster();
    int[] cor1 = new int[]{255,0,0};
    int[] cor2 = new int[]{0,0,255};
    int cont=0;
    for(int h=0;h<height;h++)
        for(int w=0;w<width;w++)
            {
                if (((w/32)+(h/32)) % 2) == 0)
                    raster.setPixel(w,h,cor1);
                else raster.setPixel(w,h,cor2);
            }
    ImageIO.write(image, "PNG", new File("checkerboard.png"));
}
```

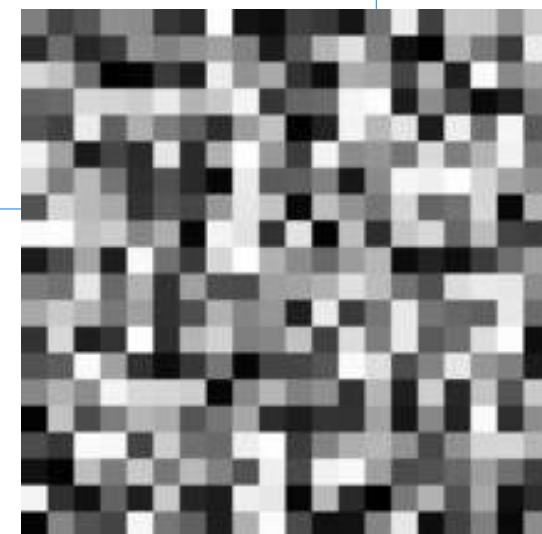


1. Criamos instância de `SampleModel` usando `RasterFactory`.
2. Criamos um `TiledImage` com este `SampleModel`.
3. Criamos um `WritableRaster` a partir da `TiledImage`.
4. Manipulamos os pixels do `WritableRaster`.

Criando Imagens (com JAI)



```
int width = 640; int height = 640;
SampleModel sampleModel =
    RasterFactory.createBandedSampleModel(DataBuffer.TYPE_BYTE,
        width,height,1);
TiledImage tiledImage =
    new TiledImage(0,0,width,height,0,0,sampleModel,null);
WritableRaster wr = tiledImage.getWritableTile(0,0);
for(int h=0;h<height/32;h++)
    for(int w=0;w<width/32;w++)
    {
        int[] fill = new int[32*32]; // A block of pixels...
        Arrays.fill(fill,(int)(Math.random()*256));
        wr.setSamples(w*32,h*32,32,32,0,fill);
    }
JAI.create("filestore",tiledImage,
    "jaigl.png","PNG");
```



Para imagens com tiles é um pouco mais complicado...

1. Criamos instância de `SampleModel` usando `RasterFactory`.
2. Criamos um `TiledImage` com este `SampleModel`.
3. Para cada *tile*:
 1. criamos um `WritableRaster` a partir da `TiledImage`.
 2. Manipulamos os pixels do `WritableRaster`.

Criando Imagens (com JAI)



```
int width = 483; int height = 483;
int tWidth = 64; int tHeight = 64;
SampleModel sampleModel =
    RasterFactory.createBandedSampleModel(DataBuffer.TYPE_BYTE,
        tWidth, tHeight, 3);
ColorModel cm = TiledImage.createColorModel(sampleModel);
TiledImage tiledImage =
    new TiledImage(0, 0, width, height, 0, 0, sampleModel, cm);

// Create the colors.
int[] red = new int[] {255, 0, 0};
int[] green = new int[] {0, 255, 0};
int[] blue = new int[] {0, 0, 255};
int[] yellow = new int[] {255, 255, 0};
int[] black = new int[] {0, 0, 0};
```

Criando Imagens (com JAI)

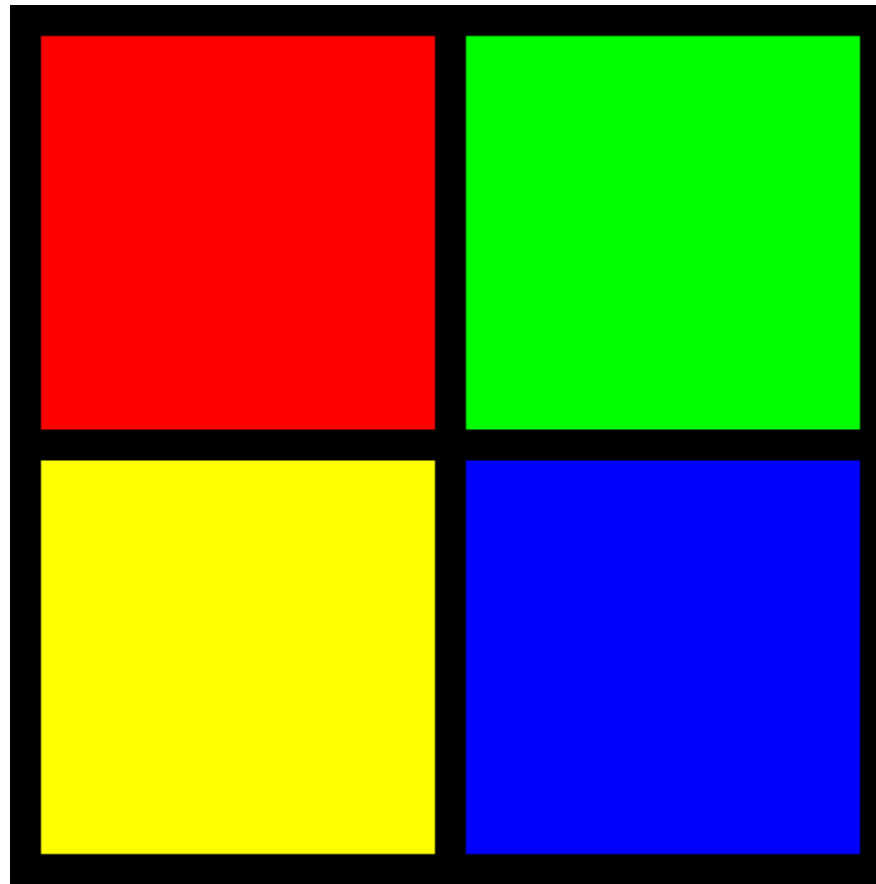


```
for(int th=tiledImage.getMinTileY();th<=tiledImage.getMaxTileY();th++)
  for(int tw=tiledImage.getMinTileX();tw<=tiledImage.getMaxTileX();tw++)
  {
    WritableRaster wr = tiledImage.getWritableTile(tw,th);
    for(int ih=0;ih<tHeight;ih++)
      for(int iw=0;iw<tWidth;iw++)
      {
        int w = wr.getMinX()+iw;
        int h = wr.getMinY()+ih;
        if ((w >= 17)&&(w < 17+216)&&(h >= 17)&&(h < 17+216))
          wr.setPixel(w,h,red);
        else if ((w >= 250)&&(w < 250+216)&&(h >= 17)&&(h < 17+216))
          wr.setPixel(w,h,green);
        else if ((w >= 17)&&(w < 17+216)&&(h >= 250)&&(h < 250+216))
          wr.setPixel(w,h,yellow);
        else if ((w >= 250)&&(w < 250+216)&&(h >= 250)&&(h < 250+216))
          wr.setPixel(w,h,blue);
        else wr.setPixel(w,h,black);
      }
  }
```

Criando Imagens (com JAI)



```
TIFFEncodeParam tep = new TIFFEncodeParam();  
tep.setWriteTiled(true);  
tep.setTileSize(tWidth,tHeight);  
JAI.create("filestore",tiledImage,"rgbtile.tiff","TIFF",tep);
```



- Sem JAI (BufferedImage):

```
public static void main(String[] args) throws IOException
{
    File f = new File(args[0]);
    BufferedImage image = ImageIO.read(f);
    System.out.println("Dimensões: "+
        image.getWidth()+"x"+image.getHeight()+" pixels");
}
```

- Com JAI (PlanarImage):

```
public static void main(String[] args) throws IOException
{
    PlanarImage image = JAI.create("fileload",args[0]);
    System.out.println("Dimensões: "+
        image.getWidth()+"x"+image.getHeight()+" pixels");
}
```


- Sem JAI (BufferedImage):

```
public static void main(String[] args) throws IOException
{
    int width = 256;
    int height = 256;
    BufferedImage image =
        new BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);
    ...
    ImageIO.write(image,"PNG",new File("checkerboard.png"));
}
```

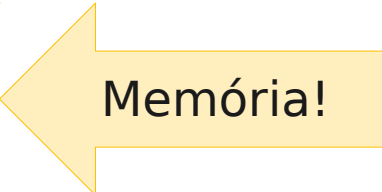
- Com JAI (PlanarImage):

```
public static void main(String[] args) throws IOException
{
    ...
    TiledImage tiledImage =
        new TiledImage(0,0,width,height,0,0,sampleModel,colorModel);
    ...
    JAI.create("filestore",tiledImage,"floatpattern.tif","TIFF");
}
```

Acesso a pixels (sem JAI)



```
public static void main(String[] args) throws IOException
{
    File f = new File(args[0]);
    BufferedImage imagem = ImageIO.read(f);
    Raster raster = imagem.getRaster();
    int[] pixel = new int[3];
    int brancos = 0;
    for(int h=0;h<imagem.getHeight();h++)
        for(int w=0;w<imagem.getWidth();w++)
        {
            raster.getPixel(w,h,pixel);
            if ((pixel[0] == 255) && (pixel[1] == 255) &&
                (pixel[2] == 255)) brancos++;
        }
    System.out.println(brancos+" pixels brancos");
}
```



Memória!

```
public static void main(String[] args) throws IOException
{
    File f = new File(args[0]);
    BufferedImage imagem = ImageIO.read(f);
    RandomIter iterator =
        RandomIterFactory.create(imagem, null);
    int[] pixel = new int[3];
    int brancos = 0;
    for(int h=0;h<imagem.getHeight();h++)
        for(int w=0;w<imagem.getWidth();w++)
            {
                iterator.getPixel(w,h,pixel);
                if ((pixel[0] == 255) && (pixel[1] == 255) &&
                    (pixel[2] == 255)) brancos++;
            }
    System.out.println(brancos+" pixels brancos");
}
```

- Existem também `RectIter` e `RookIter`.

- *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.

- <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>