Traitement d'images en Java avec JAI

Fabien Chaillou – Ingénieurs 2000

19 février 2008

Sommaire

- **Comment est construite une image ?**
- Différents traitements à l'aide de JAI
- Autres fonctionnalités de JAI
- Démos
- Conclusion
- Bibliographie

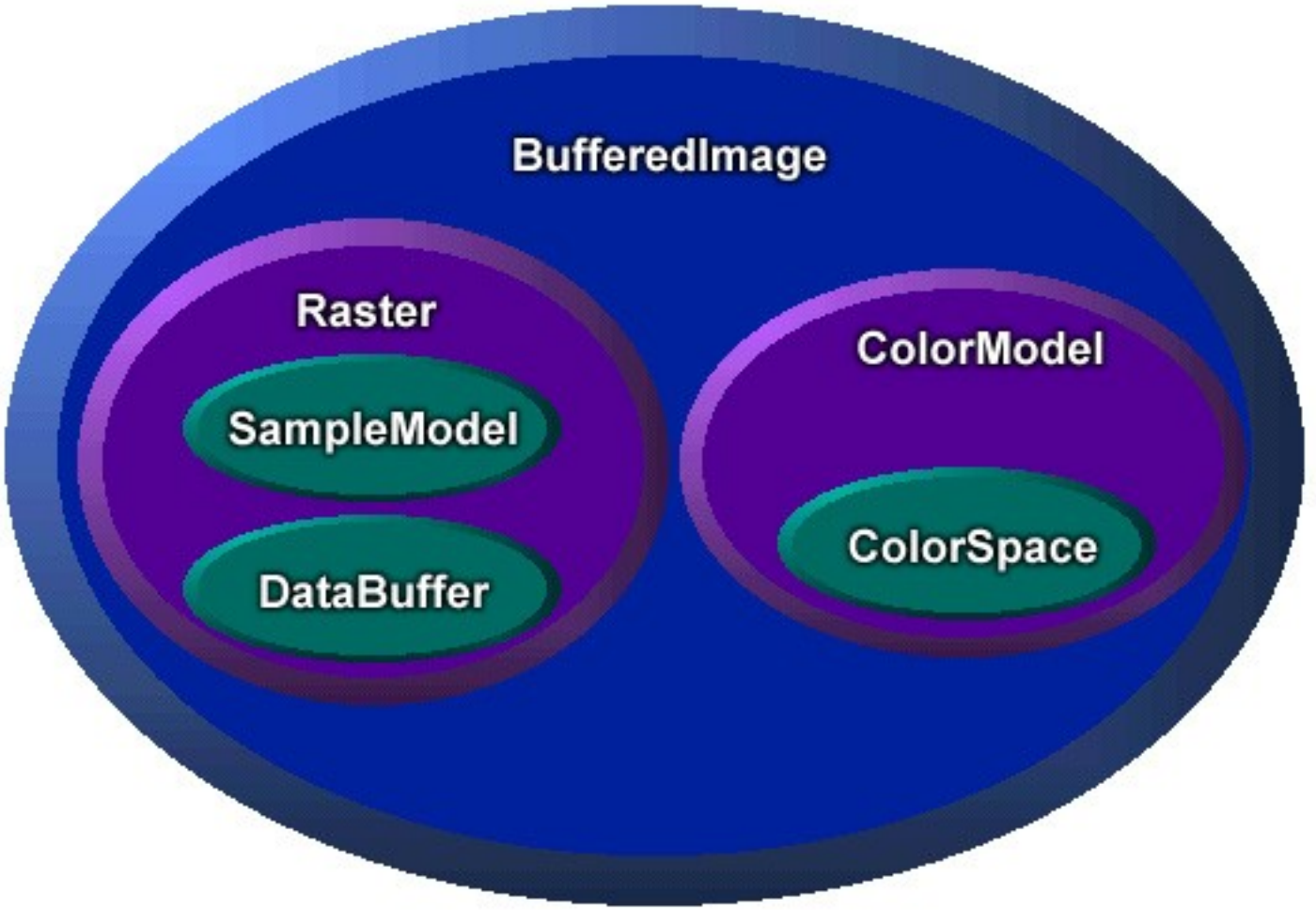
Qu'est ce qu'une image ?

- Un élement visuel d'une certaine taille
- Unité de mesure : le pixel
  - De l'anglais **pic**ture's **el**ements
- Résolution :
  - Nombre de pixels en largeur
  - Nombre de pixels en hauteur
- Chaque pixel est représentée par une couleur

Définition JAVA d'une Image

- Java 1.0 : la classe Image
  - Problèmes : peu de détails sur le contenu précis
- A partir de Java 1.2 (Java2)
  - Introduction de l'API Java2D :
    - *interfaces RenderedImage et RenderableImage.*
    - *Classe BufferedImage*
- Java Advanced Imaging
  - Mêmes interfaces que Java2D
  - Implémentations différentes : PlanarImage et TiledImage

Dissection d'une Image (1)



Dissection d'une Image (2)

- Deux composants inter-dépendants
  - Raster : contient les données de l'image
    - *DataBuffer contient les données binaires de l'image*
    - *SampleModel permet d'extraire un pixel du DataBuffer*
  - ColorModel
    - *Associer la valeur d'un pixel à une couleur*

Images ”standard”

- Image RGB(A)
  - Chaque pixel sur 3 octets (4 octets si alpha)
  - RGB(A) => Red, green, blue, (alpha)
  - Chaque octet représente une valeur entre 0 et 255 pour la couleur associée (16,7 millions de couleurs)
- Image en niveau de gris
  - Palette de 256 niveaux de gris
  - Chaque pixel sur un octet dont la valeur est l'index du niveau de gris dans la palette

Sommaire

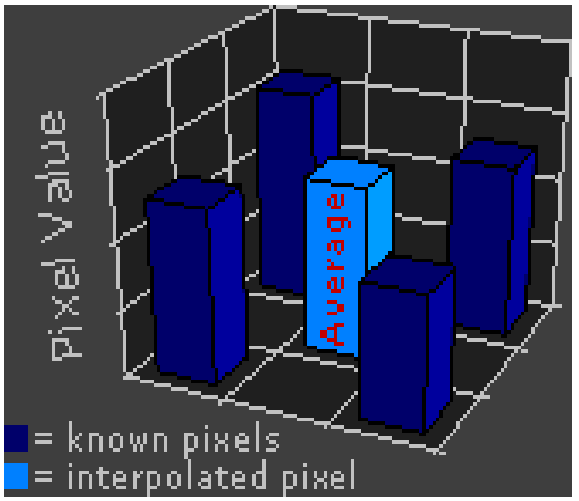
- Comment est construite une image ?
- **Différents traitements à l'aide de JAI**
- Autres fonctionnalités de JAI
- Démos
- Conclusion
- Bibliographie

Différents types de traitements

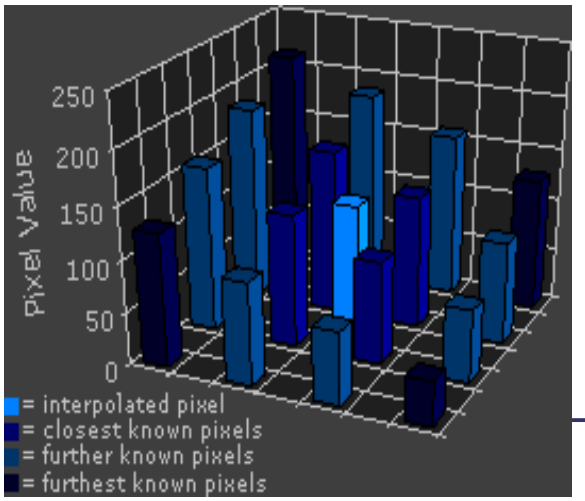
- Transformations géométriques
- Opérations Point à Point
- Opérations locales
- Opérations mixant plusieurs images

Transformations géométriques

- Redimensionner l'image
- Plusieurs techniques :
  - Nearest neighbor interpolation :
    - *Si diminution de taille prendre un pixel sur deux*
    - *Si augmentation de taille, recopier deux fois le même pixel*
  - Bilinear interpolation



Bicubic interpolation



Implémentation

```
PlanarImage source =
readSourceImage(”source.png”);
float scaleX = 0.5f, scaleY = 0.5f;
float translationX = 0f, translationY = 0f;
RenderingHints hints = new RenderingHints();
RenderedOp scaleOp =
   ScaleDescriptor.create(source,
        scaleX, scaleY,
        translationX, translationY,
        Interpolation.getInstance(
          Interpolation.INTERP_BILINEAR),
        hints);
PlanarImage result = scaleOp.getRendering();
```

Opérations point à point

- Traitement pixel par pixel
  - Modification de la teinte
- Exemples :
  - Passage d'une image en niveau de gris
    - *A partir d'une image RGB : faire la moyenne des valeurs de chaque couleur.*
  - Modification du contraste
    - *Amplifier la différence de valeur entre chaque pixel.*
    - *Multiplier la valeur du pixel par une constante.*

Implémentations (1)

```
PlanarImage source = readSourceImage(”source.png”);
double[][] bandMerge = {{0.33, 0.33, 0.33, 0}};
RenderingHints hints = new RenderingHints();
hints.put(RenderingHints.KEY_RENDERING,
          RenderingHints.VALUE_RENDER_QUALITY);
RenderedOp op = BandCombineDescriptor.create(
     source, bandMerge, hints);

PlanarImage result = op.getRendering();
```

Implémentation (2)

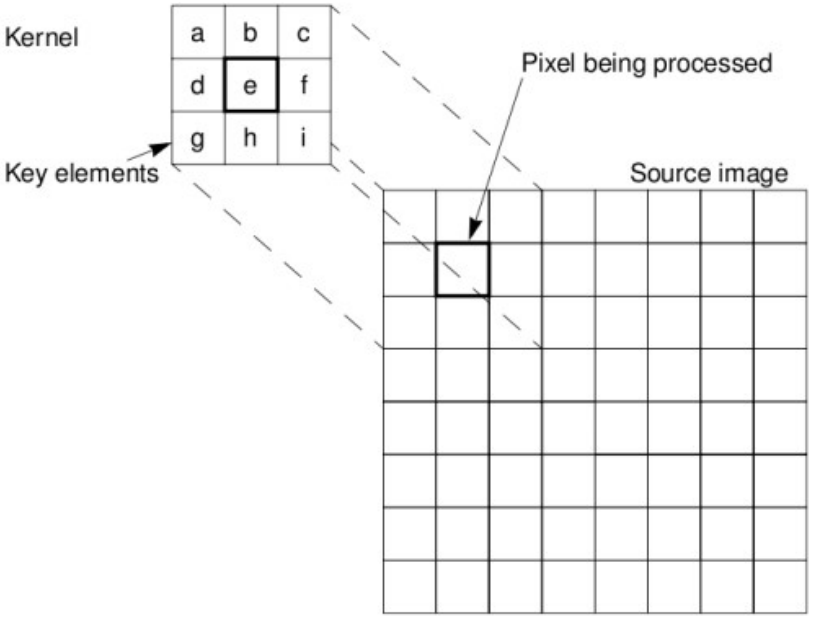
```
PlanarImage source = readSourceImage(”source.png”);
RenderingHints hints = new RenderingHints();
byte[][] datas = new byte[3][256];
for (int i = 0; i < 256; i++) {
   datas[0][i] = clampValue(i - 50);
   datas[1][i] = clampValue(i - 50);
   datas[2][i] = clampValue(i - 50);
}
LookupTableJAI table = new LookupTableJAI(datas);
RenderedOp op = LookupDescriptor.create(
              source, table, hints);

PlanarImage result = op.getRendering();
```

Opérations locales

- Opérations sur les pixels d'une image en relation avec les pixel alentours.
- Utilisation d'une matrice de convolution
- Permet de nombreux effets et traitements
- Relativement lourd à calculer
  - Nécéssite de travailler sur une copie de l'image
  - Dépendant de la taille de la matrice de convolution

Matrice de convolution

- Principe
  - Matrice carrée de taille impaire
  - Pondère la valeur de chaque pixel



Opérations possibles

- Dépendent du ”Kernel” choisi pour la convolution
  - Flou
  - Affinement de l'image
  - Detection de contours

Implémentation

```
PlanarImage source = readSourceImage(”source.png”);
RenderingHints hints = new RenderingHints();
float[] matrix = new float[]{ 1/9f, 1/9f, 1/9f,
                              1/9f, 1/9f, 1/9f,
                              1/9f, 1/9f, 1/9f };

KernelJAI kernel = new KernelJAI( 3, 3, matrix);
RenderedOp op = ConvolveDescriptor.create(source,
                    kernel, hints);

PlanarImage result = op.getRendering();
```

Opérations sur plusieurs images

- Mélange les valeurs des pixels entre deux images
- Permet de par exemple ajouter un élément d'une image vers une autre
- Images de même taille

Sommaire

- Comment est construite une image ?
- Différents traitements à l'aide de JAI
- **Autres fonctionnalités de JAI**
- Démo
- Conclusion
- Bibliographie

Region Of Interest

- Traitement sur une zone d'une image
- Ne se limite pas à une zone rectangulaire
- S'appuie sur l'inteface Shape de Java2D

Chaîne de rendu

- Ensemble de traitements sur une ou plusieurs images
- Graphe acyclique dirigé combinant un ensemble de RenderedOp
- Calcul du résultat au moment du rendu
- Chaines réutilisables

Exemple de chaine

- Réduction d'une image puis flou :

```
PlanarImage source = readSourceImage(”source.png”);
RenderingHints hints = new RenderingHints();
RenderedOp scaleOp =
   ScaleDescriptor.create(source,
        0.5f, 0.5f,  0f , 0f,
        Interpolation.getInstance(
          Interpolation.INTERP_BILINEAR), hints);
float[] matrix = new float[]{ 1/9f, 1/9f, 1/9f,
                              1/9f, 1/9f, 1/9f,
                              1/9f, 1/9f, 1/9f };
KernelJAI kernel = new KernelJAI( 3, 3, matrix);
RenderedOp op = ConvolveDescriptor.create(scaleOp,
                     kernel, hints);

PlanarImage result = op.getRendering();
```

Traitement Client-Serveur

- Déporter les traitements sur un serveur distant
- Utilisation de RMI pour la communication réseau
- Accès transparents aux opérations à distances
  - RemoteRenderedImage
  - RemoteRenderedOp
- Bonus : serialization des images

Démos

Conclusion

- API mature de gestion d'image
- S'appuie sur le JDK, facilite la prise en main
- Librairie se base sur du code natif donc traitement plus rapide
- Par contre API parfois peu ”objet”
  - Anti-design Pattern God avec la classe JAI

Perspectives Future

- Reflexion sur une refonte de JAI avec création d'une nouvelle JSR pour écrire JAI 2.0

- Traitement d'image utilisant la carte graphique :

  - Utilisation des architectures programmables des GPU directement en Java

Bibliographie

- Programming in Java Advanced Imaging : http://java.sun.com/products/java-media/jai/forDevelopers/jai1_0_1guide-unc/
- JAI Javadoc : http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs
- Interpolation :http://www.cambridgeincolour.com/tutorials/image-interpolation.htm
- Définitions : http://www.culture.gouv.fr/documentation/joconde/fr/partenaires/AIDEMUSEES/museofiche1.htm

Questions ?