Java Advanced Imaging

MIMUC Medientechnik SS 2003

Übersicht

- Wozu Bildbearbeitung mit Frameworks?
- Warum JAI?
- Verzahnung mit Java2D (AWT)
- JAI Packages und Typen
- Unterstützte Codecs und Bildformate
- BufferedImage, Raster, ColorModel (AWT)
- Rendergraphen
- Aufbau der Bildklassen
- Anzeigen eines gespeicherten Bildes
- Parameterblock
- Operationen
- Look-Up-Table
- Speichern eines Bildes
- Quellangabe

Wozu Bildbearbeitung mit Frameworks?

- Rechner/Monitore können Bilder heute schnell verarbeiten/anzeigen
  - Dadurch ist Analyse- und Bearbeitungsmöglichkeit am PC geschaffen
- Frameworks stellen bereit
  - Bildmanipulationsmöglichkeiten
  - Mathematische Operationen
  - Bildanzeigemöglichkeit
  - Bild-Codecs
- Einsatzgebiete
  - Bildanalyse- und Bildanzeigesysteme
    - Objektpositionierung, Objekttracking, Qualitätsprüfung
  - Bildbearbeitungssysteme
    - Medien, Werbeindustrie, Grafik-Design, Zeichenprogramme
  - Applikationen
    - Programme, Computerspiele
  - World Wide Web
    - Webseiten

Warum JAI?

- Pro
  - Plattformunabhängig
  - Geringer Speicherplatz
    - Schnelle Internetübertragung
  - Applets im WWW
    - Bildverarbeitende Web-Applikationen
  - Als Frontend einer anderen Programmiersprache
    - Diese übernimmt komplexere Operationen
    - Client-Server-Architektur
  - Komplexe Bildmanipulationen durchaus praktikabel
    - Großer Leistungsumfang zur Bildverarbeitung
    - Versucht alle Bedürfnisse zu erfüllen
    - Extrem Erweiterbar
- Contra
  - Langsam
    - nicht zuletzt wegen Swing

Verzahnung mit Java2D (AWT)

- JAI baut auf abstrakten Definitionen in Java2D auf
  - Benutzt Java2D Interfaces RenderableImage und RenderedImage
  - Java2D Klassen SampleModel, DataBuffer und Raster übernommen
- Gegenüberstellung im Bereich Bilder
  - Java2D für kleine Aufgaben
    - Zeichnen, Skalieren, Rotieren
  - JAI besser strukturiert als Java2D und erweiterbar
  - JAI implementiert einige schnellere Algorithmen
  - JAI unterstützt Multi-Threading
  - Komplexität des Codes bei JAI geringer, wegen Operatoren
- Anmerkungen
  - JAI ist in der Entwicklung => evtl. buggy
  - Sun will in JAI mehr Operatoren implementieren
  - JAI muss Bilder schneller verarbeiten
  - Verglichen mit C++ und dessen Bildverarbeitungsklassen ist sowohl Java2D als auch JAI geschwindigkeitsmäßig unterlegen

JAI Packages und Typen

- javax.media.jai.* (Gesamtpackage)
  - javax.media.jai.JAI
    - Fabrikklasse für Operationen durch statische Methoden (nicht instanzierbar)
  - javax.media.jai.PlanarImage
    - Bildklasse deren Objekte Bilddaten im Speicher verwalten
  - javax.media.jai.operator.*
    - Package der Vielzahl an Bildoperatoren
  - javax.media.jai.widget.*
    - Package mit Klassen für Anzeigemöglichkeit
    - javax.media.jai.widget.ScrollingImagePanel
      - Erbt von java.awt.ScrollPane, erweitert für JAI
      - Deprecated
    - javax.media.jai.widget.ImageCanvas
      - Erbt von java.awt.Canvas, erweitert für JAI
      - Deprecated
  - com.sun.media.jai.codec.*
    - Package der Encoding- und Decodingtypen

Unterstützte Codecs und Bildformate

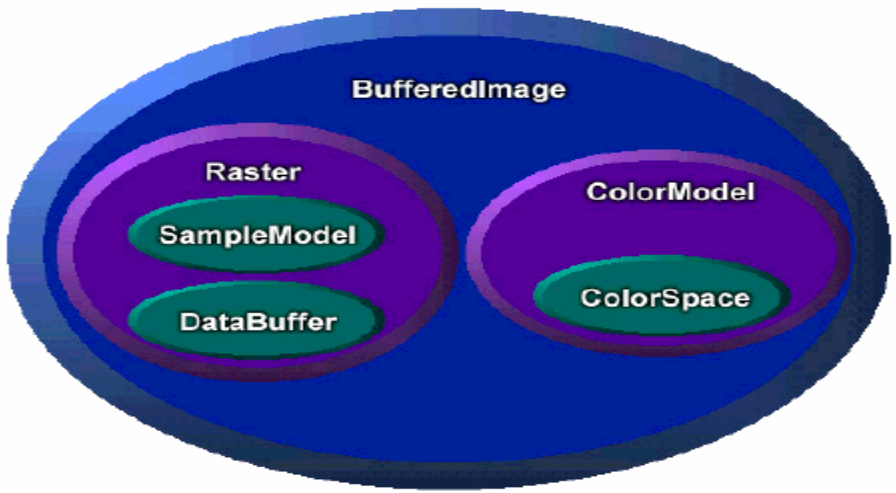
- JAI kann folgende Bildformate rendern und codiert speichern
  - BMP => Microsoft Windows Bitmap (unkomprimiert)
  - TIFF => Tag Image File Format
  - PNG => Portable Network Graphics
  - PNM => Portable aNy Map (beinhaltet: PBM, PGM, PPM)
  - JPEG => Joint Photographic Experts Group (verlustbehaftet komprimiert)
  - GIF => Graphics Interchange Format
- Näheres zu JPEG in JAI
  - Beim speichern Art/Komprimierung des JPEG bestimmen:
    - Mit/ohne JFIF-Header
    - Diskrete Kosinus Transformation (DCT) Parameter
      - Quantisierungstabellen, Horizontal- Vertikalsubsampling, Kompressionsqualität

BufferedImage, Raster, ColorModel (AWT) (1)

- Objekte aus AWT, auf die JAI aufbaut und evtl. zurückgibt
- BufferedImage
  - Erbt wie alle Bildobjekte von java.awt.Image
  - Verwaltet ein Bild im Speicher
  - Verwendung zum Double-Buffering (Kopie)
  - Unterstützt verschiedene Farbtiefen und Farbbänder
- ...hat ein ColorModel-Objekt
  - Klasse zur Übersetzung Pixelwerte in Farbkomponenten (RGB(A))
  - Nötig bei: Anzeigen, Drucken, auf anderes Bild zeichnen
  - Anzahl, Anordnung, Interpretation von Farbkomponenten in ColorSpace-Objekt
  - Pixels als 32-Bit Integer oder Array von DataBuffer-Typen (Transfertyp)
    - TYPE_BYTE, TYPE_INT, TYPE_USHORT, TYPE_SHORT, TYPE_FLOAT, TYPE_DOUBLE

BufferedImage, Raster, ColorModel (AWT) (2)

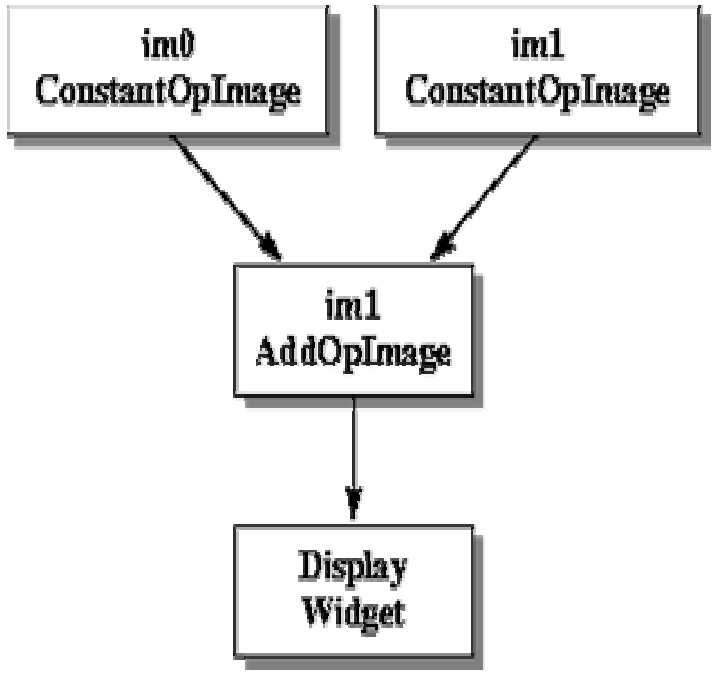
- ...hat ein Raster-Objekt
  - Beschreibt 2D-Array rechteckiger Anordnung von Pixels
  - Hat einen DataBuffer (siehe oben)
    - speichert Sample-Daten
  - Hat ein SampleModel
    - beschreibt wie gesuchter Samplewert aus DataBuffer gelesen und mittels ColorModel in Farbwert umgewandelt wird



Rendergraphen

- **JAI erleichtert Applikationserstellung durch Aufbau von Operationsgraphen**
  - **Dies erfolgt im Rendered- und Renderable Layer (Render-Graph und Renderable-Graph)**
  - **Operationen nicht sofort ausführen, erst zum Zeitpunkt des Renderns**
  - **Graphen aus Knotenobjekten, verhalten sich wie ein Bild und ein Operator**
    - **RenderedOp, RenderableOp**
- **Beispiel**

```
ParameterBlock pb0 = new ParameterBlock();
ParameterBlock pb1 = new ParameterBlock();
... // hier werden in pbX Daten zur Manipulationen
... // für die Bilder geschrieben
RenderedOp imO = JAI.create("const", pb0);
RenderedOp im1 = JAI.create("const", pb1);
im1 = JAI.create("add", imO, im1);
```



Aufbau der Bildklassen (1)

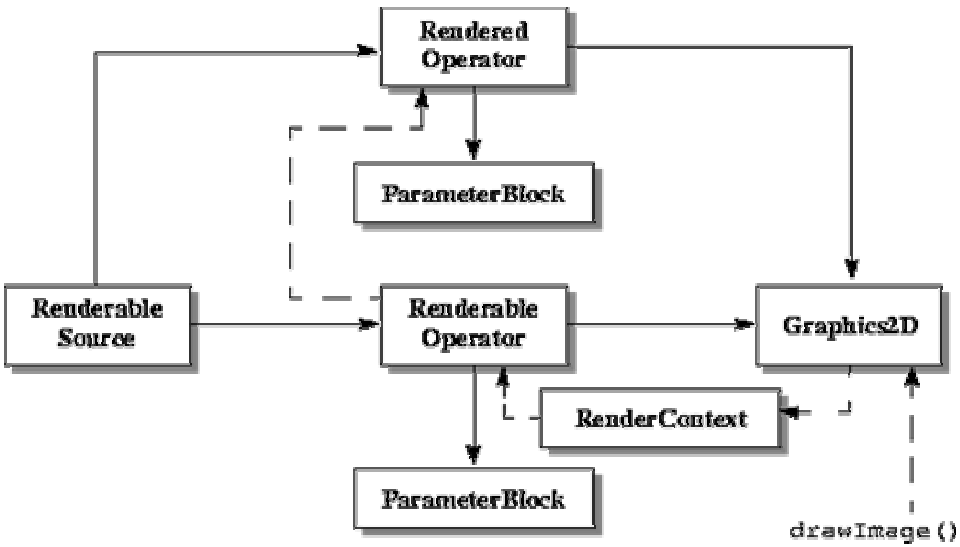
- JAI unterstützt 3 Modelle mit Java2D Grundlage
  - Producer/Consumer (Push) Model, AWT Basismodel
  - „Immediate mode“ Model, weiterentwickeltes AWT-Model
  - Pipeline (Pull) Model, neu von JAI
- Pull Model / JAI Model
  - Gliederung in 2 Schichten, für verschiedene Bildverarbeitungsmechanismen
  - Rendered Layer
    - Klassen- und Interfacenamen enthalten „rendered“
    - Grundlage: java.awt.image.RenderedImage
      - Nach Bedürfnissen des Kontexts gerendertes Bild
      - JAI Pendant: PlanarImage (später mehr)
    - Weiteres Objekte: RenderedOp
      - Instanzen sind Knoten im Render-Graphen

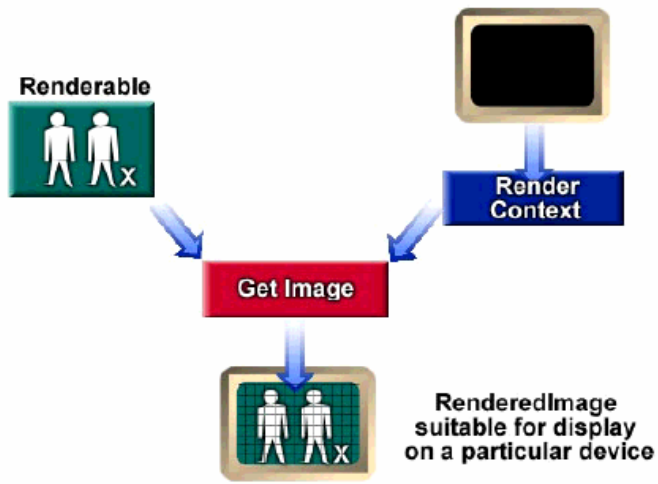
Aufbau der Bildklassen (2)

- Renderable Layer
  - Klassen- und Interfacenamen enthalten „renderable“
  - Grundlage: java.awt.RenderableImage
    - Haltung in benutzerdefiniertem Koordinatensystem
  - Unabhängig vom Renderprozess
  - Bildquellen die mehrfach oder in verschiedenen Kontexten vorkommen
  - Bietet Operatoren mit renderunabhängigen Parametern
    - Können untereinander referenziert werden => Pipelines
    - Bild aus Operatorenkette „heranziehen“ zu Anzeige/Datei

Aufbau der Bildklassen (3)

- **Objekte der RenderableLayer**
  - **RenderableOp**
    - **Instanzen sind Knoten im Renderable-Graphen**
    - **Erzeugt durch JAI.createRenderable()**
  - **Beispiel: Aufruf**
    - Graphics2D.drawImage(BufferedImage img, BufferedImageOp op, int left, int top)

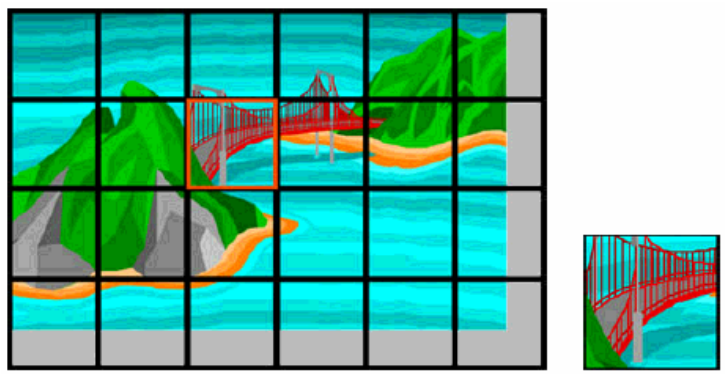




Aufbau der Bildklassen (4)

- **PlanarImage**
  - **JAI Hauptklasse zur 2D-Bild Beschreibung**
  - **Verbindung zw. Quelle und Ziel von Objekten in Render-Graph beibehalten**
- **TiledImage**
  - **Tile = wörtl. Übersetzt „Kachel/Fliese“**
  - **Beinhaltet Tiles**
    - **Ergebnis regelmäßiger Aufteilung der Bilddaten**
    - **Gitternetz von Teilbildern**
    - **Nach Erzeugung alle leer**
    - **Tileanforderung bewirkt Initialisierung mit entsprechenden Daten des Quell-PlanarImage**
    - **Auch Erhalten willkürlicher ROI (Region of Interest) aus Quell-PlanarImage**
- **Beispiel**

```
PlanarImage pi = JAI.create("fileload", “C:\\test.bmp“);
TiledImage ti = new TiledImage(pi, 50, 50);
Raster r = ti.getTile(3, 4);
ti = ti.getSubImage(150, 100, 50, 50);
```



Anzeigen eines gespeicherten Bildes (1)

- **Zum Laden eines unterstützten Bildformates gibt es 2 Wege**
  - **Mit der statischen create( )-Methode der Klasse JAI, Format wird erkannt**
  - **Entweder direkt**
    - **Funktioniert, obwohl eigentlich RenderedOp zurückkommt**
    - **Ergebnis in Bildobjekt oder Rendergraphknotenobjekt**

```
File f = new File(C:\\test.jpg);
if (f.exists() && f.canRead())
    PlanarImage pi = JAI.create("fileload", "C:\\test.jpg");
```

  - **Oder über einen Stream**
    - **Wenn man an den Header-Daten des Bildes interessiert ist (lesbar)**

```
try {
    FileSeekableStream fss = new FileSeekableStream("C:\\test.jpg");
    PlanarImage pi = JAI.create("stream", fss);
} catch(IOException e) {}
```

Anzeigen eines gespeicherten Bildes (2)

- **Anzeigen**
  - **Z.B. auf javax.media.jai.widget.ImageCanvas (deprecated)**

```
JFrame frm = new JFrame();
ImageCanvas ic = new ImageCanvas(pi); // oder: ScrollingImagePanel
frm.getContentPane().add(ic);
frm.setSize(pi.getWidth(), pi.getHeight());
frm.setVisible(true);
```

  - **Alternative: Zeichnen**

```
public class MyCanvas extends JComponent {
    private PlanarImage pi;
    public MyCanvas(PlanarImage pi) {
      super();
      this.pi = pi;
      setSize(pi.getWidth(), pi.getHeight());
    }
    public void paintComponent(Graphics g) {
      Graphics2D g2D = (Graphics2D) g;
      AffineTransform at = new AffineTransform(); // Transformationen möglich...
      g2D.drawRenderedImage(pi, at);
    }
}
```

Parameterblock

- In der JAI API Operation durch Namen (String) und Parameter beschrieben
  - Meist mit Bildquelle und weiteren Parametern
  - Übergabe aller benötigten Daten in Objekt der Klasse ParameterBlock...
    - Reihenfolge der Parameter Operationsspezifisch
    - Bildquelle unabhängig davon
  - ...an JAI.create("Operationsname“, pb)
    - Erzeugt neuen Operationsknoten anhand übergebenem ParameterBlock
- Beispiel

```
PlanarImage pi = JAI.create("fileload“, "C:\\test.jpg“);
ParameterBlock pb = new ParameterBlock();
pb.addSource(pi);
pb.add(2.0F);
pb.add(2.0F);
pi = JAI.create("scale", pb);
```

Operationen (1)

- In JAI gibt es eine Vielzahl von Operatoren
  - Von einfachen Ladeoperationen bis zur diskreten Fouriertransformation
  - Hieran wird von Sun ständig weiterentwickelt
- Operatoren-Kategorien
  - Punkt-Operatoren
    - Eingangspixelwert durch Operation zu Ausgangspixelwert
    - Add, And, Or, Xor, Divide, Invert, Lookup, Composite, Constant, Threshold
  - Flächen-Operatoren
    - Veränderung der Bildfläche durch Filtern bestimmter Pixels (Umgebungen)
    - Border, Convolve
  - Geometrie-Operatoren
    - Geometrische Transformationen die Umpositionierung der Pixel zur Folge haben
    - Lineare: Translation, Rotation, Skalierung
    - Nicht lineare: Warp-Transformationen
    - Ändern Lage, Größe, Form eines Bildes
    - Rotate, Scale, Translate, Warp

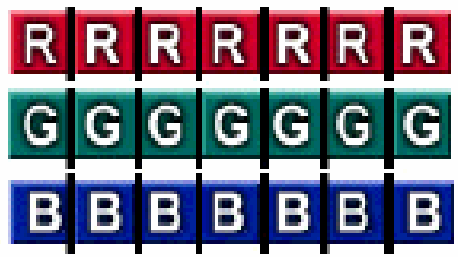
Operationen (2)

- Farbquantisierungs-Operatoren
  - Auch bekannt als Dithering
- Quantisierungsfehler gering halten bei Operationen auf:
  Monochrombilder (Farbtiefe < 8 Bit), Farbbilder (Farbtiefe < 24 Bit)
  - ErrorDiffusion, OrderedDither
- Datei-Operatoren
  - Lesen und Schreiben
  - Fileload, Filestore, Encode, JPEG, usw.
- Frequenz-Operatoren
  - Räumlich orientierte- in frequenzorientierte Bilder übersetzen (Fourier Transformation)
  - DCT, Conjugate
- Statistik-Operatoren
  - Zum Analysieren des Inhalts von Bildern
  - Extrema, Histogram
- Kantenextraktions-Operatoren
  - Gradient der Kanten im Bild zum Vorschein bringt
  - GradientMagnitude
- Weitere...

Look-Up-Table

- **Die Klasse LookupTableJAI dient Pixelwertetabellen zu erzeugen**
  - **Es gibt Singlebanded und Multibanded Konstruktoren**
  - **Singlebanded erhalten Array[0...255], Information für einen Farbwert**
  - **Multibanded erhalten 2D-Array[0..2][0...255], 3 Bänder für RGB**
  - **Farbwert des Quellpixels = Index für zugehöriges Band**
  - **Wert bei Index = Farbwert des zugehörigen Zielpixels**
  - **0 <= Werte in Arrays <= 255 (durch clamp( )-Methode)**
  - **Verwendung: Helligkeit, Farbkanäle, Kontrast**
- **Beispiel**

```
byte lutArray[][] = new byte[3][256];
for (int i = 0; i < 256; i++) {
    lut[0][i] = (byte) i;
    lut[1][i] = (byte) 0;
    lut[2][i] = (byte) 0;
}
LookupTableJAI lut = new LookupTableJAI(lutArray);
ParameterBlock pb = new ParameterBlock();
pb.addSource(pi);
pb.add(lut);
pi = JAI.create("lookup", pb);
```



Speichern eines Bildes

- **Speichern geht auch über statische JAI create( )-Methode**
  - **Wieder direkt oder als Stream zum Manipulieren des Headers**
  - **Encode-Format wird durch String mitgegeben**
  - **Für alle Formate auch *EncodeParameter (* = Format) Objekt, Interessant bei JPEG...**
    - **Erhält Encoding-Parameter zum Speichern**
    - **Übergabe Entweder an ImageEncoder Objekt oder JAI.create(“encode“, Source, Stream, “Format“, *EncodeParameter)**

- **Beispiele**
  - **1)**
  - **2)**

```
JAI.create("filestore", pi, “C:\\test.png”, “PNG”);
```

```
JPEGEncodeParam jep = new JPEGEncodeParam();
jep.setQuality(0.1F);
...  // hier: Horizontal-/Vertikalsubsampling, Quantisierungstabellen setzen (2D-Arrays)
jep.setWriteJFIFHeader(true);
try {
   FileOutputStream fos = new FileOutputStream(“C:\\test.jpg”);
   // JAI.create(“encode”, pi, fos, “JPEG”, jep);
   ImageEncoder ie = ImageCodec.createImageEncoder("JPEG", fos, jep);
   ie.encode(pi);
   fos.close();
} catch (IOException e) {}
```

Quellangabe

- Sun Microsystems: Programing in Java Advanced Imaging
  - http://java.sun.com/products/java-media/jai/forDevelopers/jai1_0_1guide-unc/JAITOC.fm.html
- Sun Microsystems: Java Advanced Imaging API Documentation
  - http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/index.html
- Sun Microsystems: Java Advanced Imaging Tutorial
  - http://developer.java.sun.com/developer/onlineTraining/javaai/jai/index.html
- JavaOne: Developing Applications with the JavaTM Advanced Imaging API
  - http://servlet.java.sun.com/javaone/javaone99/pdfs/e661.pdf
- Seminararbeit: Bildverarbeitung mit Java2D gegen JAI
  - http://www.fbilkt.fhkarlsruhe.de/.../Seminararbeit%20%20Bildverarbeitung%20Java2D%20Java%20Advanced%20I.doc
- Bildverarbeitungskurs
  - http://www.staff.fh-vorarlberg.ac.at/hv/semester5/ lecture/bv-steilkurs.pdf

Vielen Dank für Eure Aufmerksamkeit !!!