# INF555-PC1
# — Structures de données abstraites pour la 3D —
# (Prise en main de Java pour l'Image)

Frank NIELSEN
nielsen@lix.polytechnique.fr

Septembre 2008

## Avant de commencer...

Tous les codes écris dans INF555 devront être disponibles dans une seule bibliothéque. Cela vous apprendra à gérer plusieurs fichiers et projets[1], et vous servira notamment pour votre projet examen. Donnez à chaque fichier le nom de votre package `package NomPrenom`. Veuillez toujours utiliser la documentation de l'API Java qui est fournie et disponible en ligne. (Installer une copie en local sur votre ordinateur. `C:\ProgramFiles\Java\jdk-6-doc\docs\technotes\guides\2d\index.html`)

## 1 Prise en main de Java pour le traitement d'images

### 1.1 Lire les pixels d'une image

En Java, les pixels d'une image (au format BMP/JPG ou PNG) doivent être capturés en utilisant la méthode `getPixels` de l'objet `PixelGrabber`. Chaque pixel est codé sur 32 bits (4 octets): ARGB où A est le canal pour la transparence.

En s'inspirant du code minimal ci-dessous, écrivez votre propre fonction statique pour récupérer le tableau de pixels dans votre classe bibliothéque (choisissez son nom).

Listing 1: Lire les pixels d'une image

```
// Read all pixels in a given image (Frank NIELSEN)
package INF555_NielsenFrank;

import java.awt.*;
import java.awt.image.*;
import java.net.*;

public class INF555_ReadImagePixels
{
public static void main(String [] args)
{
PixelGrabber pg;
String filenameimg="liftoff.jpg";
int[] raster=null;
int x,y,w=0,h=0;
```

[1]Sous les environnements de développement `Eclipse` ou `JCreator`. Je recommende personnellement ce dernier.

```
int red,green,blue,index;

Toolkit tk=Toolkit.getDefaultToolkit();
Image img = tk.getDefaultToolkit().getImage(filenameimg);

if (img!=null)
        {
         pg = new PixelGrabber(img, 0 , 0 , -1 , -1 , true);
         try { pg.grabPixels(); } catch (InterruptedException e) { }
     raster= (int[])pg.getPixels();
        w=pg.getWidth();
        h=pg.getHeight();
        System.out.println("Image width w="+w+" height="+h);
        }
  else
  {
  System.out.println("Image not found");
  System.exit(-1);
  }
// Extract all pixels (inefficient)
for (y = 0; y < h; y++) {
            for (x = 0; x < w ; x++) {
              index=y*w+x;

                red = (raster[y * w+ x] & 0xFF) ;
                green = ((raster[y *w+ x] & 0xFF00) >> 8);
                blue = ((raster[y *w + x] & 0xFF0000) >> 16);
            // Transparency attribute a is with >>24
           //System.out.println(red+" "+green+" "+blue);
            }
        }
}
}
```

### 1.1.1 Créer une image

Pour céer une image, on crée d'abord un object `BufferedImage` puis son contexte graphique `Graphics2D`. On peut alors *dessiner* dans l'image en utilisant la bibliothéque de Java (`fillRect`, `fillOval`), etc. Ensuite, on sauvegarde l'image en utilisant la méthode `write` de la classe `ImageIO`.

En s'inspirant du code minimal ci-dessous, écrivez votre propre fonction statique pour créer des images servant de test dans votre classe bibliothéque.

Listing 2: Création et sauvegarde d'une image

```
// INF555, CreateImage, Frank NIELSEN
package INF555_NielsenFrank;

import java.awt.*;
import java.awt.image.*;
import javax.imageio.ImageIO ;
import java.io.*;


public class INF555_CreateImage
{
```

```
public static void main(String [] args)
{
try{
    int width = 640;
        int height = 480;

        BufferedImage bufferedImage = new BufferedImage(width, height, BufferedImage
            .TYPE_INT_RGB);
        Graphics2D g2d = bufferedImage.createGraphics();

        // Draw graphics
        g2d.setColor(Color.white);
        g2d.fillRect(0, 0, width, height);
        g2d.setColor(Color.black);
        g2d.fillOval(0, 0, width, height);
        g2d.dispose();
   File f = new File ("result.jpg");
  ImageIO.write (bufferedImage, "jpeg", f);
}
catch(Exception e){}
      } // end of main
}
```

## 1.2 Modifier et sauvegarder une image

En reprenant le code pour lire les pixels, nous allons maintenant créer une autre image et calculer son gradient sur l'axe $x$ (différence pixel droit-pixel courant).

## 1.3 Applets

En Java, les applets sont des applications qui s'affichent grâce à la commande `appletviewer file.html` ou alors universellement sous un navigateur (Internet Explorer, Safari, Mozilla, etc.). Pour finir, nous allons créer une petite applet qui lit une image à partir d'un TAG HTML et l'affiche.

Listing 3: Applet qui affiche des points

```
package INF555_NielsenFrank;

import java.awt.*;
import java.applet.*;

class Point2D
{double x,y;
Point2D()
{this.x=300.0*Math.random();
this.y=300.0*Math.random();}}

public class INF555_AppletPoints extends Applet {
final static int n=100;
static Point2D [] set;
public void init() {
int i;
set=new Point2D[n];
for(i=0;i<n;i++)
set[i]=new Point2D();}
```

```
public void paint(Graphics g) {int i;
for(i=0;i<n;i++)
{
int xi, yi;
xi=(int)set[i].x; yi=(int)set[i].y;
g.drawRect(xi, yi,1,1);
}
g.drawString("INF555", 50, 60 );
}
}
```

Listing 4: Applet qui affiche une image

```
package INF555_NielsenFrank;

import java.awt.*;
import java.applet.*;



public class INF555_AppletImage extends Applet {
String filenameimg="liftoff.jpg";


public void init() {

URL url = INF555_ReadImagePixels.class.getResource(filenameimg);
Image img = Toolkit.getDefaultToolkit().getImage(url);
}

public void paint(Graphics g) {

  }
}
```

```
filenameimgurl=getParameter("imgurl");
```

# 2 Une applet pour le coloriage de dessins (*area flooding*)

On désire colorier une image qui est donnée sous formes de contours 4-connexes. Nous allons implanter deux méthodes: en utilisant pile (équivalent à un parcours depth first search dans le graphe d'adjacence des pixels) et en utilisant une queue (équivalent a breadth first search). Lequel est le plus performant? Quel sont les pires des cas?

## 2.1 Pile

Ecrire une version utilisant la pile comme structure de données.

## 2.2 Queue

Ecrire une version utilisant une file. Dans votre applet, affichez le nombre maximal d'élements dans la file (ou empilé dans la pile)

# 3 Union d'ensembles disjoints

## 3.1 Etiquettage des composantes connexes

On se donne une image binaire et l'on désire étiquetter toutes les composantes connexes de celles-ci en utilisant la structure de donnée de Tarjan: Union-find.

```
UnionFind(int n)
{int k;

parent=new int[n];
rank=new int[n];

System.out.println("Creating a UF DS for "+n+" elements");

for (k = 0; k < n; k++)
      {parent[k]   = k;
      rank[k] = 0;      }
}

//
// Find procedures
//
int Find(int k)
{
  while (parent[k]!=k ) k=parent[k];
    return k;}

//
// Assume x and y being roots
//
int UnionRoot(int x, int y)
{
  if ( x == y ) return -1;

      if (rank[x] > rank[y])
      {parent[y]=x; return x;}
      else
      { parent[x]=y;if (rank[x]==rank[y]) rank[y]++;return y;}
}
}
```

## 3.2 Segmentation d'images par fusion de régions

Maintenant on veut faire la segmentation d'images couleur en utilisant un prédicat pour décider si deux régions connexes doivent être fusionnées ou pas. Ce problème est rencontré en analyse d'images médicales dD (3D, 4D, etc.) ou encore pour segmenter des planches de dessin animés (afin de pouvoir ultérieurement créer des images intermédiaires). Par exemple, on peut choisir comme premier prédicat un seuil constant: Fusionner si et seulement si (ssi) la difference en valeur absolue des niveaux moyens de gris est inférieur à une valeur donnée.

Ecrire une fonction `MergePredicate` qui décide si l'on doit fusionner ou pas deux régions connexes. Afin de tester le prédicat, on ordonne les paires de pixels adjacents par ordre croissant de difference de couleur. Ecrire le programme de segmentation.

Utiliser le prédicat ci-dessous avec $g = 256$ (niveaux de gris) et $Q = 32$. Comparer qualitativement les résultats obtenus.

```
boolean MergePredicate(int reg1, int reg2)
{
double dR, dG, dB;
double logreg1, logreg2;
double dev1, dev2, dev;

dR=(Ravg[reg1]-Ravg[reg2]);
dR*=dR;

dG=(Gavg[reg1]-Gavg[reg2]);
dG*=dG;

dB=(Bavg[reg1]-Bavg[reg2]);
dB*=dB;

logreg1 = min(g,N[reg1])*Math.log(1.0+N[reg1]);
logreg2 = min(g,N[reg2])*Math.log(1.0+N[reg2]);

dev1=((g*g)/(2.0*Q*N[reg1]))*(logreg1 + logdelta);
dev2=((g*g)/(2.0*Q*N[reg2]))*(logreg2 + logdelta);

dev=dev1+dev2;

return ( (dR<dev) && (dG<dev) && (dB<dev) );
}
```

# 4 Exercices supplémentaires (a finir chez soi)**

Ecrire les applets correspondants au Chapitre 2 de *Visual Computing*:

## 4.1 File de priorité et détection d'une intersection de segments de droites

## 4.2 Hachage géométrique pour l'appariements