Programmation graphique avancée, animations (suite)

Java 2D

Une animation

Inspirée de celle écrite par Arthur van Hoff qui se trouve à se trouve :
http://www.javaworld.com/jw-03-1996/animation/Example7Applet.html
écrite par Arthur van Hoff extrait de son article sur la programmation d'animations en Java disponible à :
http://www.javaworld.com/jw-03-1996/animation
J'ai ajouté le chargement par le MediaTracker, une réécriture de update() et paint().
On dispose des 2 images :

world.gif:

car.gif:

2 voitures avancent de droite à gauche sur l'image du monde.

Une animation (A. V. Hoff)

le fichier html contient :

```
<applet code=Example7Applet.class width=200 height=200>
<param name=fps value=20>
</applet>
```

applet d'animation

```
import java.awt.*;

public
class Example7Applet extends
java.applet.Applet implements Runnable {
    int frame;
    int delay;
    Thread animator;

    Dimension offDimension;
    Image offImage;
    Graphics offGraphics;

    Image world;
    Image car;

    /**
     * Initialisation de l'applet et calcul du delai
     * entre "trames".
     */
    public void init() {
        String str = getParameter("fps");
     int fps = (str != null) ? Integer.parseInt(str) : 10;
        delay = (fps > 0) ? (1000 / fps) : 100;

        tracker = new MediaTracker(this);
        world = getImage(getCodeBase(),
"world.gif");
        car = getImage(getCodeBase(), "car.gif");
       tracker.addImage(world, 0);
       tracker.addImage(car, 0);
    }
```

```
    /**
     * La methode start() de l'applet. On crée la
     * thread d'animation et on la lance.
     */

    public void start() {
        animator = new Thread(this);
        animator.start();
    }

    /**
     * Le corps de la thread.
     */
    public void run() {
        // stocker la date de lancement
        long tm = System.currentTimeMillis();
        while (Thread.currentThread() == animator) {
            // lance l'affichage de l'animation
            repaint();

            // Delai d'attente ajuste pour avoir la
            // meme attente entre chaque trame.
            try {
                tm += delay;
                Thread.sleep(Math.max(0, tm -
System.currentTimeMillis()));
            } catch (InterruptedException e) {
                break;
            }

            // numero de trame incremente pour
            // pouvoir afficher la trame
            // suivante.
            frame++;
        }
    }
```

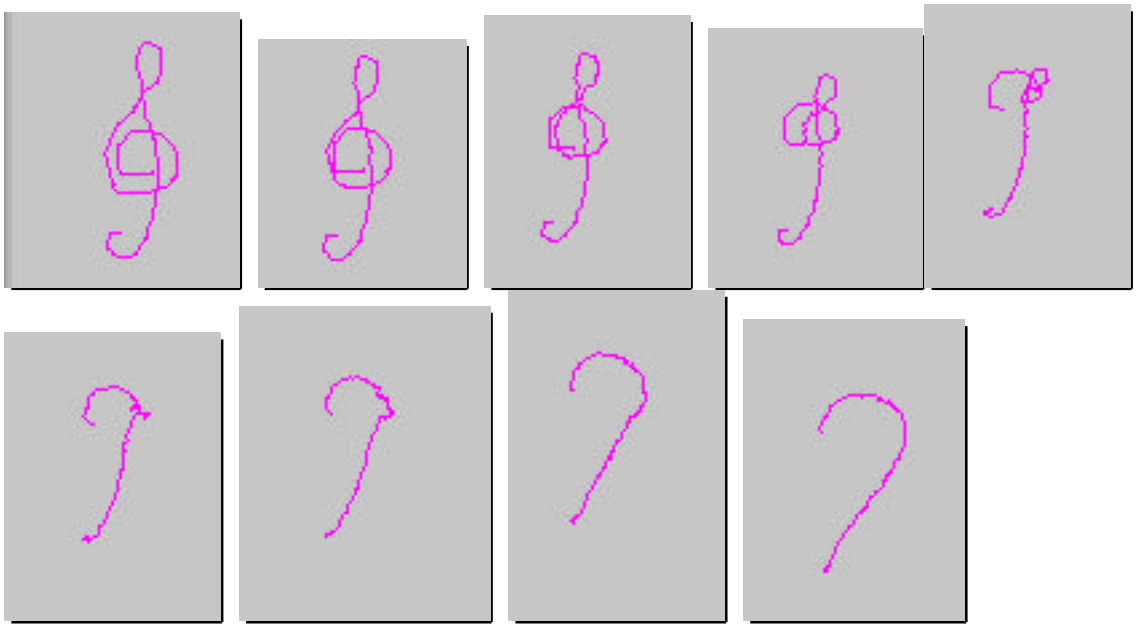
```
    /**
     * La methode stop() de l' applet.
     * Elle arrete la thread et desalloue
     * les entites necessaires au double buffering.
     */
    public void stop() {
        animator = null;
        offImage = null;
        offGraphics = null;
    }

    /** dessine la trame courante. */
    public void paint(Graphics g) {
        Dimension d = size();

// Cree le double buffering si ce n'est pas deja
fait.
        if ((offGraphics == null)
         || (d.width != offDimension.width)
         || (d.height != offDimension.height)) {
           offDimension = d;
           offImage = createImage(d.width, d.height);
           offGraphics = offImage.getGraphics();
       }

        // efface l'image precedente
        offGraphics.setColor(getBackground());
        offGraphics.fillRect(0, 0, d.width, d.height);
        offGraphics.setColor(Color.black);

        // prepare le dessin de la bonne trame
        paintFrame(offGraphics);

        // affiche la bonne trame a l'écran
        g.drawImage(offImage, 0, 0, null);
    }
```

```
    public void update(Graphics g) {
        paint(g);
    }
    /** la creation de la trame :
      * utilise le double buffering et le MediaTracker
      */
    public void paintFrame(Graphics g) {
        // Ne faire l'affichage que lorsque
        // toutes les images ont été chargées
        // au besoin provoquer le chargement
        // des images par status(..., true);
        if (tracker.statusID(0, true) ==
MediaTracker.COMPLETE) {
        Dimension d = size();
        int w = world.getWidth(this);
        int h = world.getHeight(this);
            g.drawImage(world, (d.width - w)/2,
(d.height - h)/2, this);
        w = car.getWidth(this);
        h = car.getHeight(this);
            w += d.width;
            //dessine la premiere voiture qui avance de
             // a la fois 5 pixels de droite à gauche
            g.drawImage(car, d.width - ((frame * 5) %
w), (d.height - h)/3, this);
            //dessine la seconde voiture :
            // elle avance de 7 pixels de droite à
gauche
            g.drawImage(car, d.width - ((frame * 7) %
w), (d.height - h)/2, this);
        }
        else { // dans le cas où les images n'ont
        // pas encore été chargées
g.drawString("Images en cours de chargement",
50, 50);
        }
    }
}
```

Morphing 2D en Java

Une applet écrite par Pat Niemeyer (pat@pat.net) utilisable à : http://www.ooi.com/tween/editor.html

On fait 2 dessins et le programme passe de l'un à l'autre par "morphing".



"... feel free to use it for your class but please don't redistribute it"

code morphing P. Niemeyer

L'applet utilise essentiellement 2 classes dans le paquetage `tween` : la classe `Editor` et la classe `Canvas`. L'applet présente tout d'abord un éditeur de dessin :

dont une partie du code est :

```
package tween;
•••
class Editor extends Panel {
        tween.Canvas canvas;
        Graphics canvasGr;
        int xpos, ypos, oxpos, oypos;
        private int [][] track = new int [2048][2];
        private int trackEnd = 0;
        private Vector tracks = new Vector();
        static boolean standalone;

        Editor() {
                •••
                add( "Center", canvas = new tween.Canvas() );
                Panel p = •••
                p.add( new Button("Clear") );
                p.add( new Button("Tween") );
                add( "South", p );
        }
```

```
final public boolean handleEvent( Event e ) {
        int x = e.x, y = e.y;
        if ( e.id == Event.MOUSE_DRAG ) {
                xpos = x; ypos = y;
                addPoint( x, y );
                if ( canvasGr == null )
                        canvasGr = canvas.getGraphics();
                canvasGr.setColor( Color.red );
                canvasGr.drawLine( oxpos, oypos,
oxpos=xpos, oypos=ypos );
                return true;
        } else
        if ( e.id == Event.MOUSE_DOWN ) {
                oxpos = x; oypos = y;
                trackEnd = 0;
                addPoint( x, y );
                return true;
        } else
        if ( e.id == Event.MOUSE_UP ) {
                int l;
                int [][] tr = new int [ trackEnd ][2];
                for ( int i=0; i< tr.length; i++) {
                        tr[i][0] = track[i][0];
                        tr[i][1] = track[i][1];
                }
                tracks.addElement( tr );
                return true;
        }
        return true;
    }
    public boolean action( Event e, Object o ) {
        ••• if ( o.equals("Tween") ) {
                canvas.setTracks( tracks );
                canvas.startTweening(); •••
        }
        return true;
    }

    private void addPoint( int x, int y ) {
        try {
                track[trackEnd][0] = x;
                track[trackEnd][1] = y;
                trackEnd++;
        } catch ( ArrayIndexOutOfBoundsException e2 ) {
                System.out.println("too many points!!!");
                trackEnd = 0;
        }
    }
}
```

morphing P. Niemeyer (suite)

L'éditeur est un code classique de "rubber band" avec les 3 cas à traiter de bouton souris : appui, déplacer bouton enfoncé, relachement. À chaque manipulation, le code remplit un tableau `track` (indice `trackEnd` incrémenté) qui va "contenir la courbe tracé par l'utilisateur".
Lorsque le tracé est fini ce tableau est alors mis dans le vecteur `tracks` qui constitue alors le premier élément du vecteur. Le tracé suivant sera le second élément du vecteur, (etc. car on peut mettre plusieurs tracés).
Un tracé est constitué d'un ensemble de points qui ont eux-mêmes deux coordonnées d'où la déclaration de tableau à deux indices de la forme :

```
track[numeroDuPoint][0 pour x, 1 pour y]
```

morphing P. Niemeyer (suite)

La seconde classe importante est `tween.Canvas` :

```
package tween;

import java.awt.*;
import java.util.Vector;
import java.io.*;

class Canvas extends java.awt.Canvas implements Runnable
{
	private Vector tracks = new Vector();
	private Thread runner;
	private Image drawImg;
	private Graphics drawGr;
	private boolean loop;

	void setTracks( Vector tracks ) {
		if ( runner != null )
			stopTweening();
		this.tracks = tracks;
	}
	synchronized public void startTweening( ) {
		if ( runner != null )
			stopTweening();
		if ( tracks == null || tracks.size() == 0 )
			return;
		runner = new Thread( this );
		runner.setPriority( runner.getPriority() + 1 );
		this.loop = false;
		runner.start();
	}
	synchronized public void stopTweening() {
		if ( runner == null )
			return;
		runner.stop();
		runner = null;
	}
	public void run() {
		do {
			tweenTracks();
		} while ( loop );
	}
```

```
        private void tweenTracks() {
                int n = tracks.size();
                for (int i=0; i< n-1; i++)  {
                        tweenTrack( (int [][])tracks.elementAt(i),
                                (int [][])tracks.elementAt(i+1) );
                }
                // draw the final track precisely...
                // (we've tweened "up to" it)
                clearGr();
                drawTrack( (int [][])tracks.elementAt(n-1),
Color.magenta );
        }

        private void tweenTrack(
                int [][] fromTrack, int [][] toTrack ) {
        // execute le morphing du dessin fromTrack au dessin
        // toTrack
                int tweens = averageDistance( fromTrack, toTrack
)/2;
                if ( tweens < 1 )
                        tweens = 1;
        // on cherche maintenant a construire une suite de
        // dessins de fromTrack a toTrack. Ces courbes sont
        // numerotees par tweenNo et on en construit autant que
        //  la "distance moyenne de fromTrack a toTrack.
                for ( int tweenNo=0; tweenNo < tweens;
tweenNo++) {
        // calcul du nombre de points de la courbes de numero
        // tweenNo
                        int len = (int)( fromTrack.length
                                + 1.0*(toTrack.length -
fromTrack.length)/tweens * tweenNo );
        // La courbe construite s'appelle tweenTrack
                        int [][] tweenTrack = new int [len][2];

                        for ( int i = 0; i < tweenTrack.length; i++ ) {
        // Recherche des indices dans les courbes
        // fromTrack et toTrack des points correspondants
        // au  point courant (d'indice i) a construire dans
        // tweenTrack.
                                int from =
(int)(1.0*fromTrack.length/tweenTrack.length * i);
                                int to =
(int)(1.0*toTrack.length/tweenTrack.length * i);
                                int x1 = fromTrack[from][0];
                                int x2 = toTrack[to][0];
                                int y1 = fromTrack[from][1];
                                int y2 = toTrack[to][1];
```

```
        // On place alors ce point d'indice i dans tweenTrack
                        tweenTrack[i][0] = (int)(x1 + 1.0*(x2-
x1)/tweens * tweenNo);
                        tweenTrack[i][1] = (int)(y1 + 1.0*(y2-
y1)/tweens * tweenNo);
                } // for ( int i = 0; ...
        // on dessine tweenTrack
                clearGr();
                drawTrack( tweenTrack, Color.magenta );
                try {
                        Thread.sleep( 1000/24 );
                } catch ( Exception e ) { }
        } // for ( int tweenNo=0 ...
    }

    private int averageDistance( int [][] track1, int [][] track2 )
    {
    /*
    On ne retient qu'un point sur 10 pour calculer la
     distance moyenne entre deux les deux courbes.
    */
        int averages = (track1.length + track2.length)/2/10;
        if ( averages < 3 )
                averages = 3;
        int t = 0;
        for ( int i=0; i< averages; i++ ) {
                int [] p1 = track1[ (int)( 1.0 * track1.length /
averages * i ) ];
                int [] p2 = track2[ (int)( 1.0 * track2.length /
averages * i ) ];
                int dx = p2[0] - p1[0];
                int dy = p2[1] - p1[1];
                t += (int)Math.sqrt( dx*dx + dy*dy );
        }
        return t/averages;
    }

    public void update( Graphics g ) {
        paint(g);
    }

    public void paint( Graphics g ) {
        if ( drawImg == null )
                return;
        g.drawImage(drawImg, 0, 0, null);
    }
```

```
        public void drawTrack(int [][] track, Color color ) {
                Graphics gr = getOffScreenGraphics();
                gr.setColor( color );
10
                for ( int i=0; i < track.length-1; i++ )
                        gr.drawLine( track[i][0], track[i][1],
                                track[i+1][0], track[i+1][1] );

                repaint();
        }


        private void clearGr() {
                getOffScreenGraphics().clearRect(0, 0,
size().width, size().height);
        }

        public Graphics getOffScreenGraphics() {
                if ( drawGr == null ) {
                        drawImg = createImage( size().width,
size().height );
                        drawGr = drawImg.getGraphics();
                }
                return drawGr;
        }
```

classe tween.Canvas P. Niemeyer

Dans l'éditeur après appui sur le bouton tween, la méthode `setTracks()` est lancée puis "l'animation morphing" par `startTweening()`.
`setTracks()` positionne le champ `tracks` qui est le vecteur des formes à "joindre" par morphing.
`startTweening()` crée la thread de morphing dont le corps est `tweenTracks()`.

`tweenTracks()` va faire le morphing entre les 2 formes finales en parcourant le vecteur des formes finales (on peut supposer que ce vecteur n'a que 2 éléments !!).

Morphing (P. Niemeyer)

Finalement c'est donc la méthode `private void tweenTrack(int [][] fromTrack, int [][] toTrack)` qui est la clé de tout l'ensemble.
Cette méthode cherche d'abord la distance moyenne entre les 2 courbes. Cette distance donne alors le nombre de courbes intermédiaires à dessiner. Il faut alors construire et dessiner chaque courbe intermédiaire. Pour une courbe donnée (i.e. pour une valeur de `tweenNo`), on construit cette courbe à `len` points. `len` est calculé de sorte à être proche du nombre de points de la courbe finale pour les dernières courbes et vice-versa pour les premières courbes. On repère les points correspondants aux deux courbes finales pour le point à construire, puis on construit ce point ce qui initialise les deux composantes de `tweenTrack[i]`.

Les dessins sont fait en double buffering et on dessine 24 courbes par secondes.

Bibliographie

http://www.javaworld.com/javaworld/jw-03-1996/jw-03-animation.html

http://java.sun.com:81/applets/Fractal/1.0.2/example1.html

Teach yourself Java in 21 days : Laura Lemay, Charles L.Perkins ; ed Sams.net traduit en français ed S&SM "Le programmeur Java"

Java, les animations et le multimédia

On trouve sur Internet certains sites proposant des programmes Java (souvent des applets) qui intègrent, le mouvement, le changement d'images, parfois les sons, etc. Malheureusement très souvent ces pages demande beaucoup de temps de chargement.

http://www.worldnet.net/~tomsoft/Java/View3D/View3D.html propose des objets 3D (avion, hélicoptère, dirigeable, etc.) à manipuler avec la souris. propose des effets d'ombres, des vues filaires, etc.

http://www.developer.com/directories/pages/dir.java.mm.anim.video.html
un catalogue remis à jour quotidiennement des meilleures applets d'animation et multimédia.

http://www.boutell.com/baklava/scones.html
le célèbre jeu "casse bouteille" spatial.

Bibliographie

Java, les animations et le multimédia (suite)

http://junior.apk.net/~jbarta/idiot/idiot.html
un bon gag : un bouton poussoir qui fuit quand on essaie de cliquer dessus (mais parfois on y arrive !).

http://www.best.com/~pvdl/rubik.html
le cube de Rubik en Java : magnifique !
plus généralement le site de Karl Hörnell à http://www.tdb.uu.se/~karl/ est très beau avec de magnifique applets à http://www.tdb.uu.se/~karl/brain.html entre autre un éditeur de partition et exécuteur de mélodies à : http://www.tdb.uu.se/~karl/java/bluenotes.html

http://javaboutique.internet.com/
Une belle "boutique" de programmes Java dans beaucoup de domaines y compris le multimédia.

Java 2D

ajoute de nouvelles fonctionnalités graphiques (manipulation de la couleur, transformations géométriques, ...) à `java.awt` pour les dessins, le texte, les images, etc. pour les applications comme pour les applets.
fait partie intégrante de Java 1.2. Était un ajout dans Java 1.1.

Système de coordonnées

En Java 2D le système de coordonnées 2D est :

appelé système de coordonnées utilisateur.

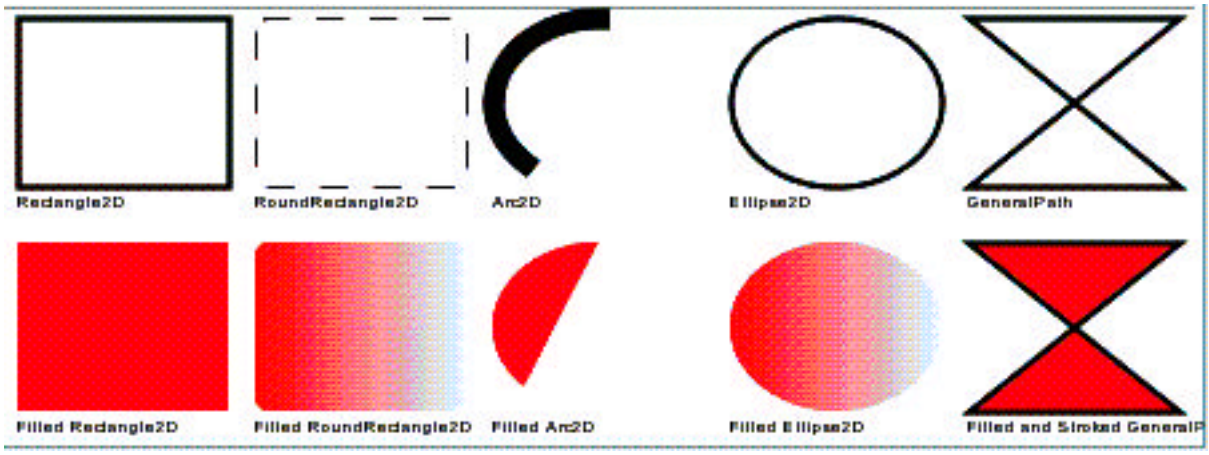
utilise la classe `java.awt.Graphics2D` classe dérivée de `java.awt.Graphics`.

En fait très souvent le code utilisant Java 2D est de la forme :

```
public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
...
}
```

Les figures proposées

Ce sont des classes du paquetage `java.awt.geom`. On trouve par exemple, `Arc2D`, `Ellipse2D`, `GeneralPath`, `Rectangle2D`, ...



Graphics2D

Les principaux attributs de cette classe sont :

- stroke qui contrôle le "pen style" (la manière de dessiner les contours : épaisseur, pointillé, type de jointure entre lignes, ...)
- paint qui contrôle le mode de remplissage : couleurs unies, motifs de remplissage, texture, ...
- composite indiquant le mode de fusion entre les couleurs i.e. pixel source et pixel destination (transparence, opacité, ...)
- transform définissant les transformations affines à faire sur le dessin (rotation, homothétie, ...) et ceci quel que soit le "type" de dessin (image, texte, ...)
- clip indiquant la zone de clipping
- font qui concerne les dessins (glyph) des caractères
- rendering hints ou suggestions de rendu qui traite de l'antialiasing, le traitement des images, ... Ce sont des suggestions qui peuvent être utilisées (meilleur rendu) ou ignorées (plus rapide à l'exécution).

Ces champs sont manipulables par des méthodes appropriées : setStroke(), setPaint(), setComposite(), setTransform(), setClip(), setFont(), setRenderingHints().

Un peu de code

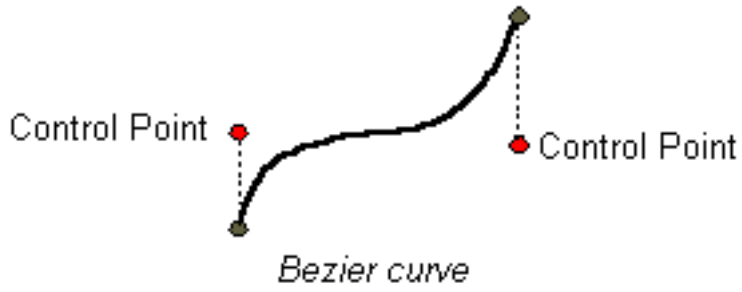
Ce sont ces méthodes et ces objets qui permettent d'obtenir :

par le code :

```
float dash1[] = {10.0f};
BasicStroke dashed = new BasicStroke(1.0f,
                                     BasicStroke.CAP_BUTT,
                                     BasicStroke.JOIN_MITER,
                                     10.0f, dash1, 0.0f);


g2.setStroke(dashed);
g2.draw(new RoundRectangle2D.Double(x, y, rectWidth,
                                    rectHeight, 10, 10));
```

Dessins avancés

Java 2D permet de dessiner des courbes de degré 2 ou plus (3 = courbes de Bézier par exemple)



La classe `java.awt.geom.GeneralPath` permet de construire des succession de courbes.
La classe `java.awt.geom.Area` permet de faire des opérations ensemblistes entre aires. (union, intersection, soustraction, ...).
Ces classes implémentent l'interface `java.awt.Shape` (utile pour le polymorphisme).

Traitement des images

La classe `java.awt.image.BufferedImage` dérivée de `java.awt.Image` implémente le double buffering pour les images. C'est cette classe qui permet de manipuler les images. On lui associe des objets "opérateurs sur les images" de la classe `BufferedImageOp`. Par exemple un code qui filtre une image est du style :

```
short[] threshold = new short[256];
for (int i = 0; i < 256; i++)
        threshold[i] = (i < 128) ? (short)0 : (short)255;

BufferedImageOp thresholdOp =
        new LookupOp(new ShortLookupTable(0, threshold),
        null);

BufferedImage destination = thresholdOp.filter(source, null);
```

Diverses Opérations sur les images

Convolution

Ces opérations consiste à changer la valeur d'un pixel en fonction des pixels environnants. Par exemple si la nouvelle valeur d'un pixel est une moyenne pondérée à l'aide la matrice (appelé kernel):

```
float ninth = 1.0f / 9.0f;

float[] blurKernel = {
        ninth, ninth, ninth,
        ninth, ninth, ninth,
        ninth, ninth, ninth
};

BufferedImageOp blur = new ConvolveOp(new Kernel(3, 3,
        blurKernel));
```

l'image apparaît nettement plus floue.
Une matrice comme :

```
float[] edgeKernel = {
        0.0f, -1.0f, 0.0f,
        -1.0f, 4.0f, -1.0f,
        0.0f, -1.0f, 0.0f
};
BufferedImageOp edge = new ConvolveOp(new Kernel(3, 3,
edgeKernel));
```

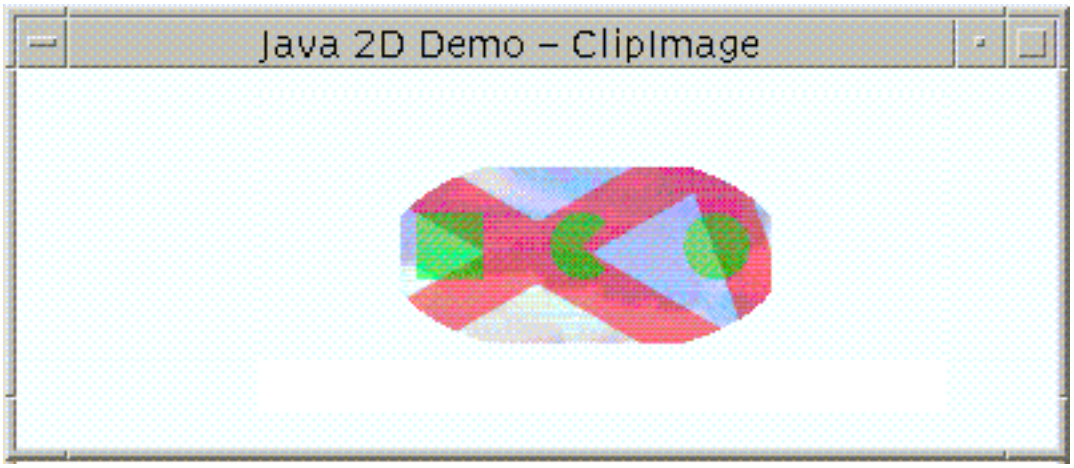
fait apparaître seulement les contours : le voir sur une couleur unie. Très utile en imagerie médicale.

Invertion

permet de faire apparaître le négatif d'une image.
Très souvent on combine plusieurs de ces opérations.

Clipping

À toute forme (Area) peut être associée une zone de clipping qui indique sa partie réellement affichée. Cette zone de clipping peut être une forme géométrique ou un texte :

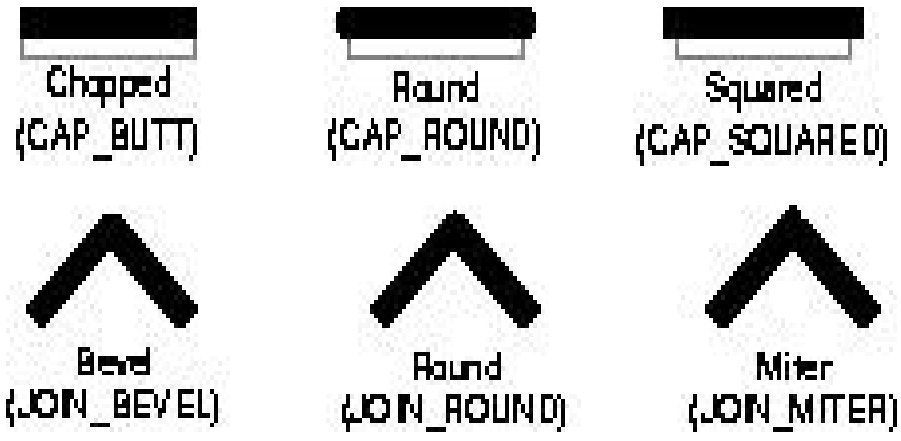


Diverses techniques abordées par Java 2D

l'antialiasing

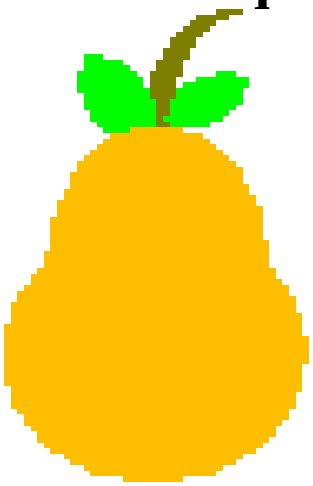
Si on dessine seulement les points appartenant à la courbe, la rupture nette de couleur fait apparaître un "hachage" : aliasing.
En traitant aussi les pixels qui voisins de la courbe, celle ci apparaît plus fluide : c'est le traitement de l'antialiasing.

Attributs de fin de segment



Combinaisons d'aires

Création de nouvelles formes géométriques



La figure ci dessus construite par programmation Java 2D est constitué de :
- l'union d'un cercle est d'une ellipse pour le corps
- l'intersection de 2 cercles verts pour chacune des feuilles
- la soustraction de 2 cercles pour la tige

voir le tutorial Java 2D à
http://java.sun.com/products/jdk/1.2/docs/guide/2d/spec/j2d-bookTOC.doc.html
ou
http://www.javasoft.com/docs/books/tutorial/2d/

Code Java 2D

En Java 1.1 on écrit :

```
public void paint(Graphics g) {
        g.setColor(Color.red);
        g.fillRect(300, 300, 200, 100);
}
```

qui dessine un rectangle rouge.

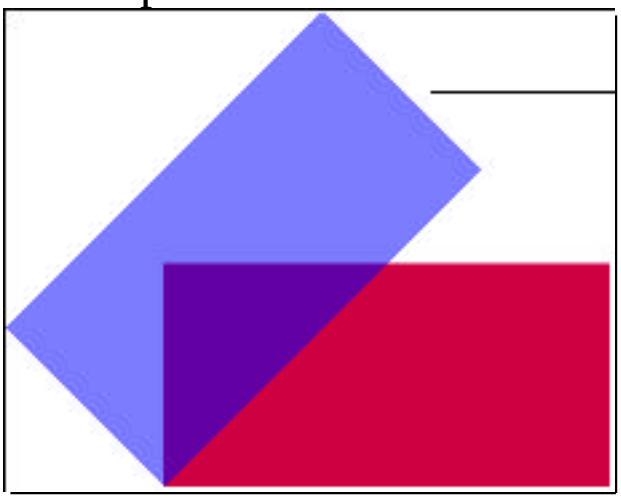
En Java 1.2, on écrit :

```
public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;

        g2d.setColor(Color.red);

        GeneralPath path = new
GeneralPath(GeneralPath.EVEN_ODD);
        path.moveTo(300.0f, 400.0f); // coin bas gauche
        path.lineTo(500.0f, 400.0f); // ligne vers la droite
        path.lineTo(500.0f, 300.0f); // ligne vers le haut
        path.lineTo(300.0f, 300.0f); // ligne vers la gauche
        path.closePath(); // on referme le trace

        g2d.fill(path); // on remplit en rouge.
}
```

on dessine en fait un chemin (objet de la classe `GeneralPath`) et après l'avoir refermé, on le remplit. Le mode de remplissage est indiqué dans le constructeur du chemin.
On peut évidemment dessiner des rectangles qui sont maintenant des objets de classe qui implémentent l'interface `Shape`.

manipulations avancées

transformations affines (translation, rotation), transparence :

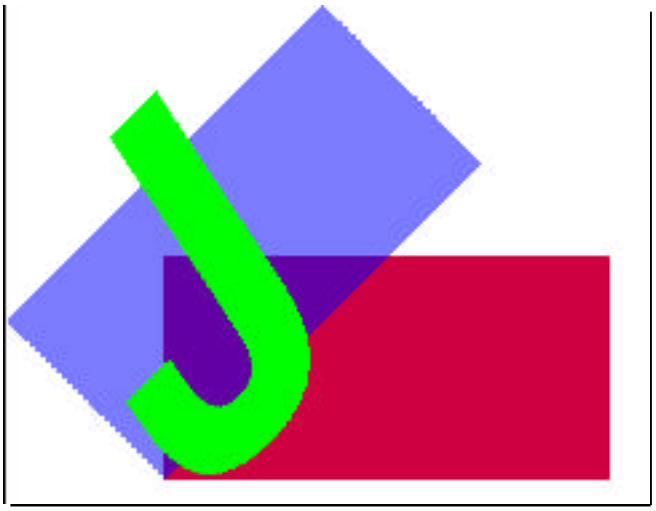


```
public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.setColor(Color.red);
        GeneralPath path = new
GeneralPath(GeneralPath.EVEN_ODD);
        path.moveTo(0.0f, 0.0f); // lower left corner
        path.lineTo(200.0f, 0.0f); // lower right corner
        path.lineTo(200.0f, -100.0f); // upper right corner
        path.lineTo(0.0f, -100.0f); // upper left corner
        path.closePath(); // close the rectangle
        AffineTransform at = new AffineTransform();
        at.setToTranslation(300.0, 400.0);
        g2d.transform(at);
        g2d.fill(path);

        // second rectangle
        g2d.setColor(Color.blue); // il sera bleu
        // positionne la composante alpha de transparence a
        // 50% de transparence
        AlphaComposite comp =
AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
0.5f);
        g2d.setComposite(comp);
        // rotation de -PI/4 radians (i.e. -45°).
        // L'orientation est celle des aiguilles d'une montre !!
        at.setToRotation(-Math.PI/4.0));
        g2d.transform(at);
        // affichage du rectangle bleu
        g2d.fill(path);
}
```

Le texte

Le texte est traité comme un dessin. Par exemple pour dessiner la lettre J en police Helvetica grasse oblique, après une rotation de 45° "à gauche" sur les rectangles précédent, il suffit d'ajouter :

```
// construire l'objet police Helvetica-BoldOblique de 200 points
 Font myFont = new Font("Helvetica-BoldOblique",
                   Font.PLAIN, 200);

 // afficher le caractere `J' en vert, opaque
 // apres rotation de 45°
 // a gauche en utilisant le gc precedent.

g2d.setColor(Color.green);
g2d.setComposite(AlphaComposite.getInstance(AlphaCompo
site.SRC_OVER, 1.0f));
g2d.drawString("J", 0f, 20f);
```



Les images

De même que pour les dessins et les figures, on peut appliquer des transformations affines sur des images et écrire :

```
Image image = applet.getImage(url);
AffineTransform at = new AffineTransform();
at.rotate(Math.PI/4.0);
g2d.drawImage(image, at, this);
```

Les pixels des images peuvent avoir une valeur de transparence distincts 2 à 2 comme :



Bibliographie Java 2D

Livres

Computer Graphics principles and practice. Foley, van Dam, Feiner, Hughes; ed Addison Wesley. ISBN 0-201-84840-6.

URL Remarquables

La page de départ pour la technologie Java 2D à :
http://java.sun.com/products/java-media/2D/index.html

Démonstrations de programmes Java 2D à partir de cette page.

http://java.sun.com/products/jdk/1.2/docs/guide/2d/spec/j2d-title.fm.html

Le tutorial Java 2D.

http://www.javasoft.com/docs/books/tutorial/2d/

ou un autre à
http://java.sun.com/products/jdk/1.2/docs/guide/2d/spec/j2d-bookTOC.doc.html

Mailing list : java2d-interest

mailing list concernant Java2D. Pour y souscrire, envoyer un mail à javamedia-request@sun.com avec subscribe java2d-interest dans le corps du message.

archive de ces mails accessible à : http://java.sun.com/products/java-media/mail-archive/2D/index.html

SDK pour Java 2D

Tout est inclus dans Java 1.2.

Java 1.2 : http://java.sun.com/products/jdk/1.2/docs/index.html

Plusieurs articles Java 2D de Bill Day and Jonathan Knudsen à http://www.javaworld.com/javaworld/jw-07-1998/jw-07-media.html, http://www.javaworld.com/javaworld/jw-08-1998/jw-08-media.html, http://www.javaworld.com/javaworld/jw-09-1998/jw-09-media.html.