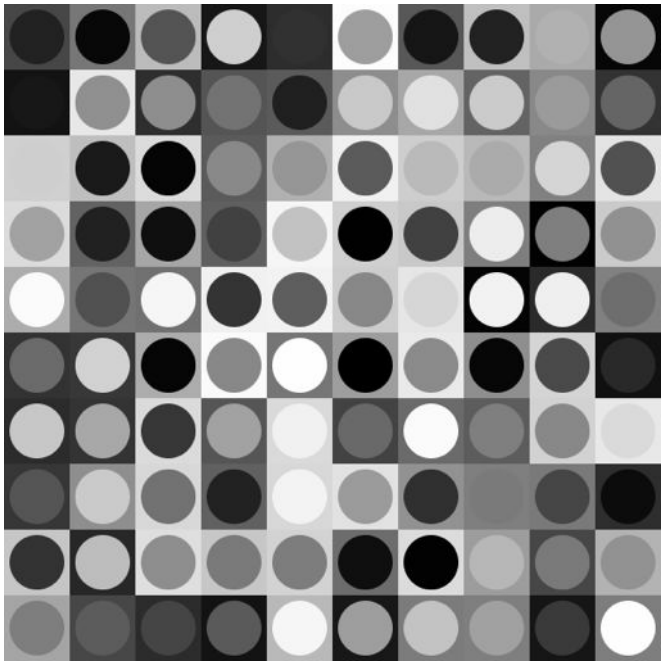
CNAM - NSY116 - oct. 2014

Techniques graphiques 2D

Pierre Cubaud <cubaud @ cnam.fr>



- Dessin bitmap
- Tracé de droites et cercles
- Courbes paramétrées
- Remplissage
- Transformations géométriques

(1) Dessin bitmap

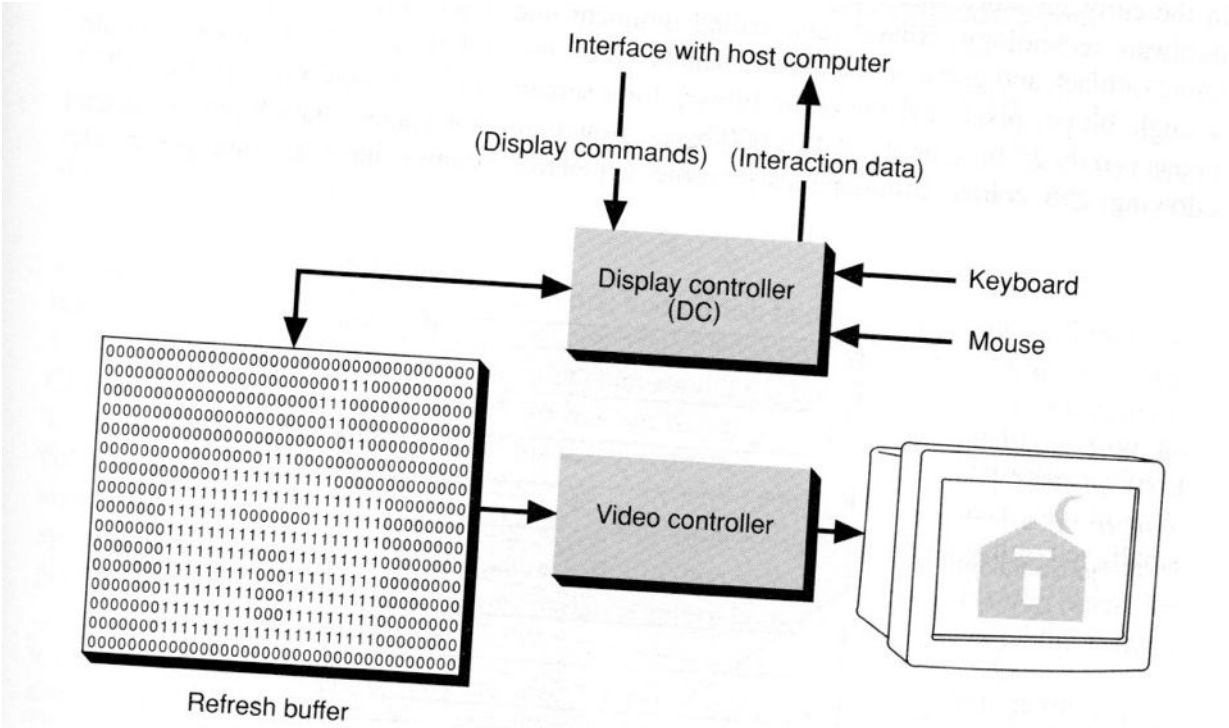
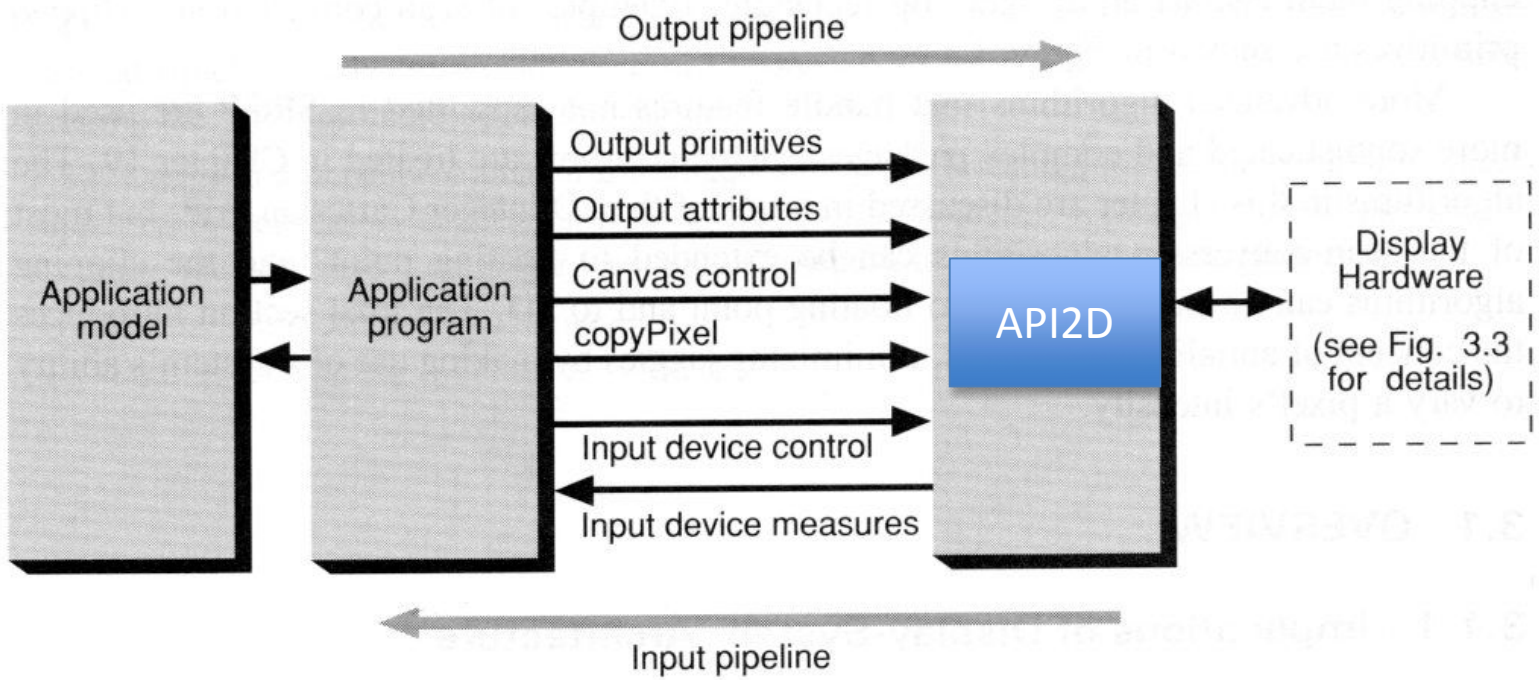


Fig. 1.2 Architecture of a raster display.

[Foley et al., p.11]

Organisation API Graphique



[Foley et al., p.68]

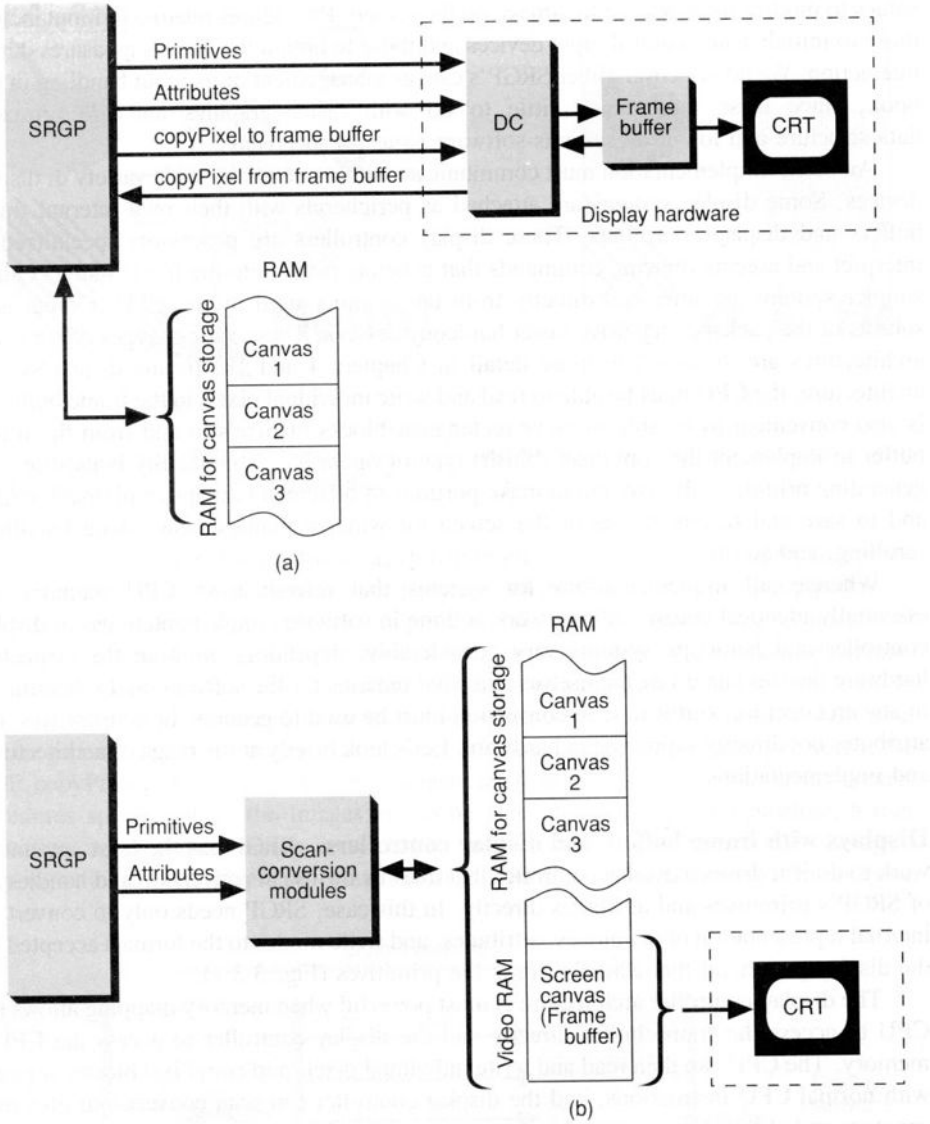


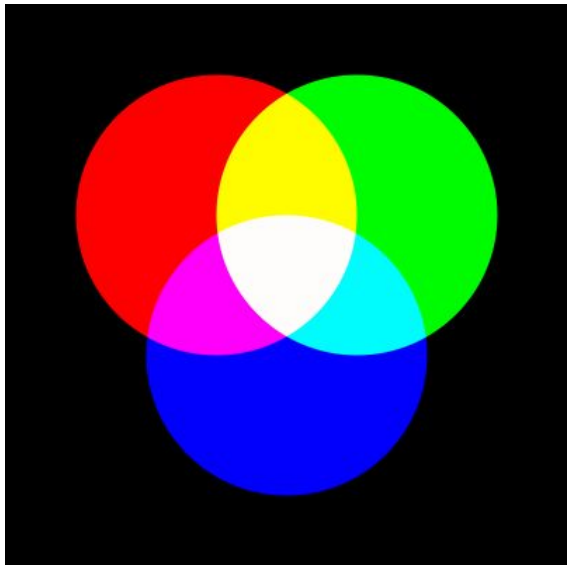
Fig. 3.3 SRGP driving two types of display systems. (a) Display peripheral with display controller and frame buffer. (b) No display controller, memory-shared frame buffer.

[Foley et al., p.70]

Opération sur les bitmaps

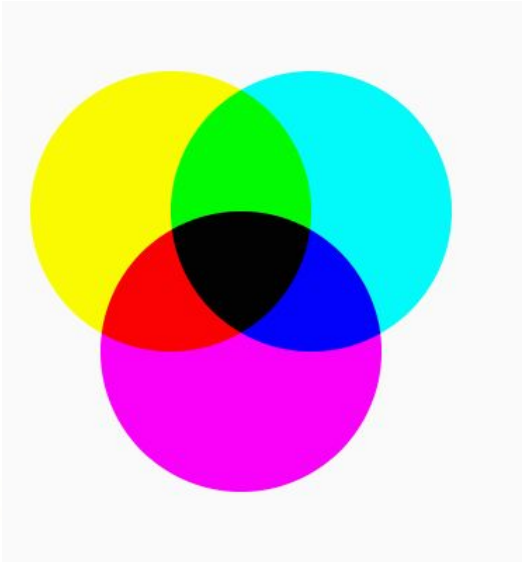
syntheseAdditive §

```
PGraphics cr,cv,cb;

cr = createGraphics(400,400, JAVA2D);
cr.beginDraw();
cr.smooth();cr.background(0);cr.fill(255,0,0);cr.noStroke();
cr.ellipse(150,150,200,200);
cr.endDraw();
cv = createGraphics(400,400, JAVA2D);
cv.beginDraw();
cv.smooth();cv.background(0);cv.fill(0,255,0);cv.noStroke();
cv.ellipse(250,150,200,200);
cv.endDraw();
cb = createGraphics(400,400, JAVA2D);
cb.beginDraw();
cb.smooth();cb.background(0);cb.fill(0,0,255);cb.noStroke();
cb.ellipse(200,250,200,200);
cb.endDraw();

cr.blend(cv, 0,0,400,400, 0,0,400,400, ADD);
cr.blend(cb, 0,0,400,400, 0,0,400,400, ADD);
cr.save("resuADD2.png");
```



syntheseSoustractive

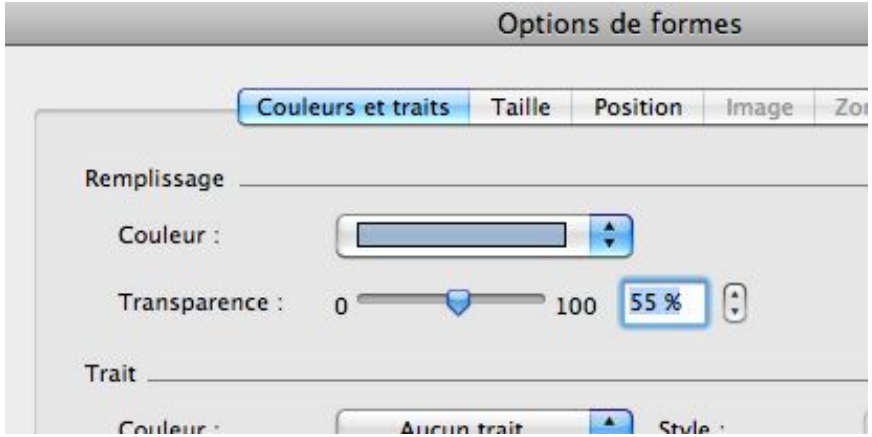
```
PGraphics cr,cv,cb;

cr = createGraphics(400,400, JAVA2D);
cr.beginDraw();
cr.smooth();cr.background(255);cr.fill(255,255,0);cr.noStroke();
cr.ellipse(150,150,200,200);
cr.endDraw();
cv = createGraphics(400,400, JAVA2D);
cv.beginDraw();
cv.smooth();cv.background(255);cv.fill(0,255,255);cv.noStroke();
cv.ellipse(250,150,200,200);
cv.endDraw();
cb = createGraphics(400,400, JAVA2D);
cb.beginDraw();
cb.smooth();cb.background(255);cb.fill(255,0,255);cb.noStroke();
cb.ellipse(200,250,200,200);
cb.endDraw();

cr.blend(cv, 0,0,400,400, 0,0,400,400, DIFFERENCE);
cr.blend(cb, 0,0,400,400, 0,0,400,400, DIFFERENCE);
cr.save("resuDIFF.png");
```



Nombreux autres opérateurs dans la fonction blend()

La transparence ("alpha channel")

On ajoute à la description <rvb> ou <tsl> du pixel un 4ème paramètre de dosage (en %) pour les opérations de composition



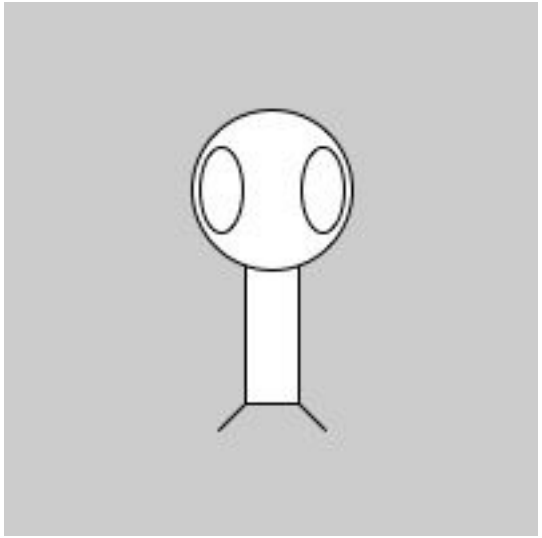
ex de PowerPoint

transparence

```
size(600,600);
background(255);
noStroke();
fill(0,0,250,50);
rect(200,200,400,200);
fill(250,0,0,50);
rect(400,200,200,400);
```

(2) Primitives de tracé

Exemple de Processing : http://processing.org/tutorials/drawing/



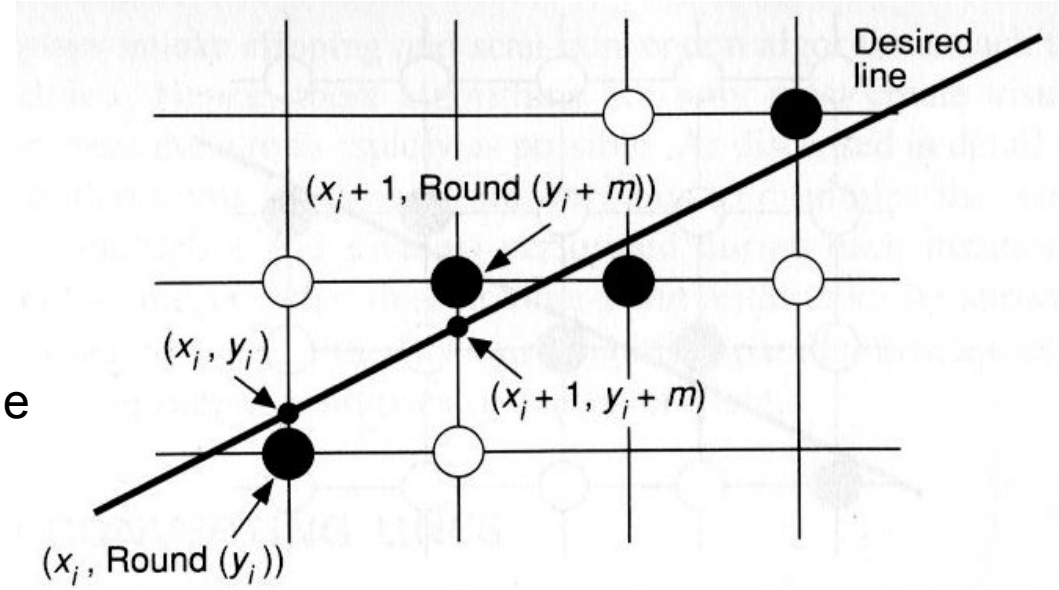
```
size(200,200);
rectMode(CENTER);
rect(100,100,20,100);
ellipse(100,70,60,60);
ellipse(81,70,16,32);
ellipse(119,70,16,32);
line(90,150,80,160);
line(110,150,120,160);
```

Ca marche comment ???

Le tracé de droite

un algo. incrémental ☺
qui utilise des nombres réels ☹

rque : pour les pentes m>1, le rôle de x et y est permutté et on progresse en y




```
procedure Line (                        {Assumes -1 ≤ m ≤ 1, x0 < x1}
    x0, y0,                             {Left endpoint}
    x1, y1,                             {Right endpoint}
    value : integer);                   {Value to place in line's pixels}
var
    x : integer;                        {x runs from x0 to x1 in unit increments
    dy, dx, y, m : real;
begin
    dy := y1 - y0;
    dx := x1 - x0;
    m := dy/dx;
    y := y0;
    for x := x0 to x1 do
      begin
        WritePixel (x, Round (y), value);   {Set pixel to value}
        y := y + m                          {Step y by slope m}
      end
end; {Line}
```

[Foley et al. p73-75]

Algorithme en nombres entiers de Bresenham (1965)

```
void myLine(int x1, int y1, int x2, int y2, color c) {
int   incx,incy,inc1,inc2,dx,dy,piv,d,x,y,xd,yd,xf,yf;
boolean   invert;

  if (x1<x2) incx=1; else if (x1==x2) incx=0; else incx=-1;
  if (y1<y2) incy=1; else if (y1==y2) incy=0; else incy=-1;
  dx=abs(x2-x1);dy=abs(y2-y1);
  // si la pente est >1, inversion des axes
  if (dx<dy) {
    piv=dx;dx=dy;dy=piv;
    piv=incx;incx=incy;incy=piv;
    xd=y1;yd=x1;xf=y2;yf=x2;
    invert=true;
  }
  else {
    xd=x1;yd=y1;xf=x2;yf=y2;
    invert=false;
  }

   // premier point
    x=xd;y=yd;inc1=2*dy;inc2=(dy-dx)*2;d=(dy*2)-dx;
    if (invert) myPoint(y,x,c); else myPoint(x,y,c);
    // boucle de dessin
    while (x != xf) {
     x=x+incx;
     if (d<0)
      d=d+inc1;
     else {
      y=y+incy;
      d=d+inc2;
     }
     if (invert) myPoint(y,x,c); else myPoint(x,y,c);
    }
}
```

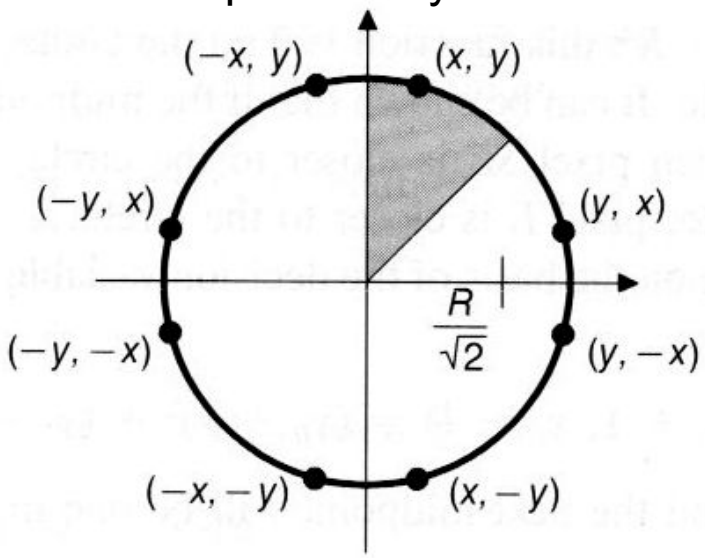
Le tracé de cercle

$$(x - x_C)^2 + (y - y_C)^2 = R^2$$

$$\text{ou forme explicite : } \begin{cases} x = x_C + R\cos(t) \\ y = y_C + R\sin(t) \end{cases}, t \in [0, 2\pi]$$

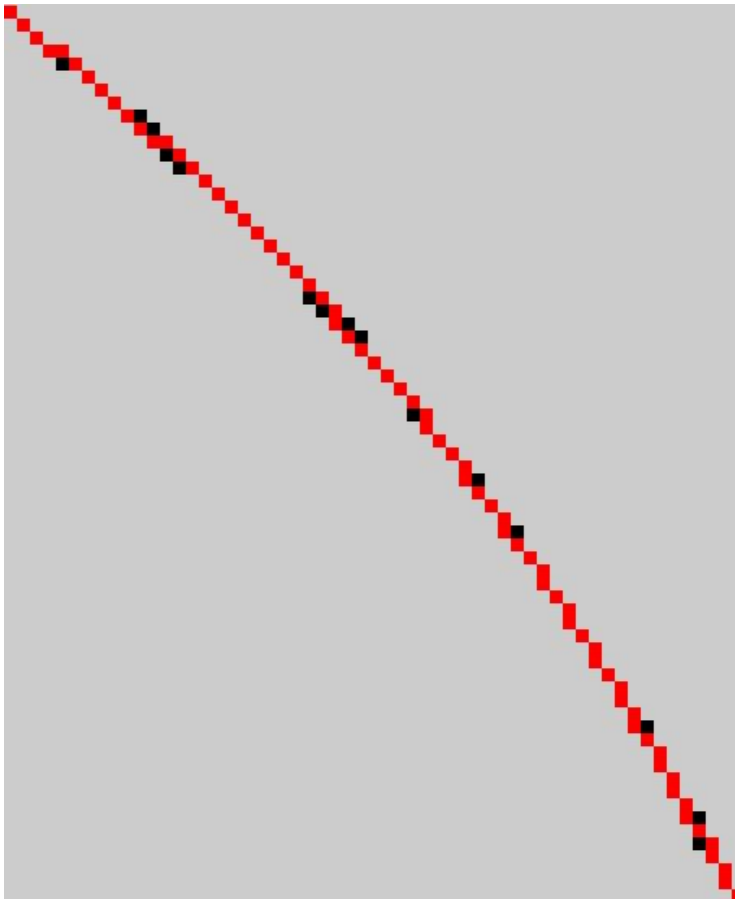
Pas très rapide => utiliser une approche de récurrence comme avec la droite
Crée des lacunes

On exploite la symétrie

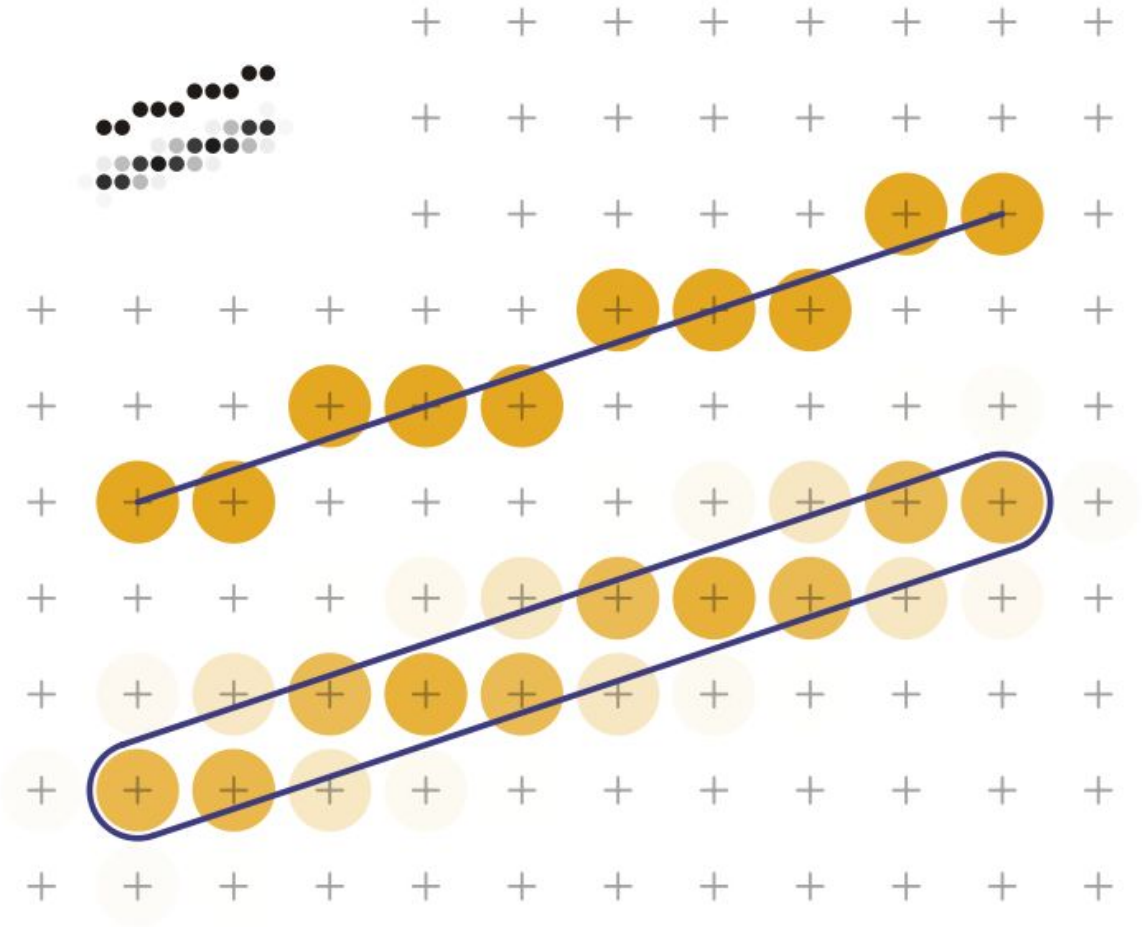


```
void drawCircle(int xC, int yC, int r) {
 int x = 0;
 int y = r;
 int d = r - 1;

 while(y >= x){
  point(xC + x, yC + y);
  point(xC + y, yC + x);
  point(xC - x, yC + y);
  point(xC - y, yC + x);
  point(xC + x, yC - y);
  point(xC + y, yC - x);
  point(xC - x, yC - y);
  point(xC - y, yC - x);

  if (d >= 2*x) {
   d -= 2*x + 1;
   x ++;
  }
  else if (d < 2 * (r-y)) {
   d += 2*y - 1;
   y --;
  }
  else {
   d += 2*(y - x - 1);
   y --;
   x ++;
  }
 } //while
}
```

comparaison avec le tracé de Processing :

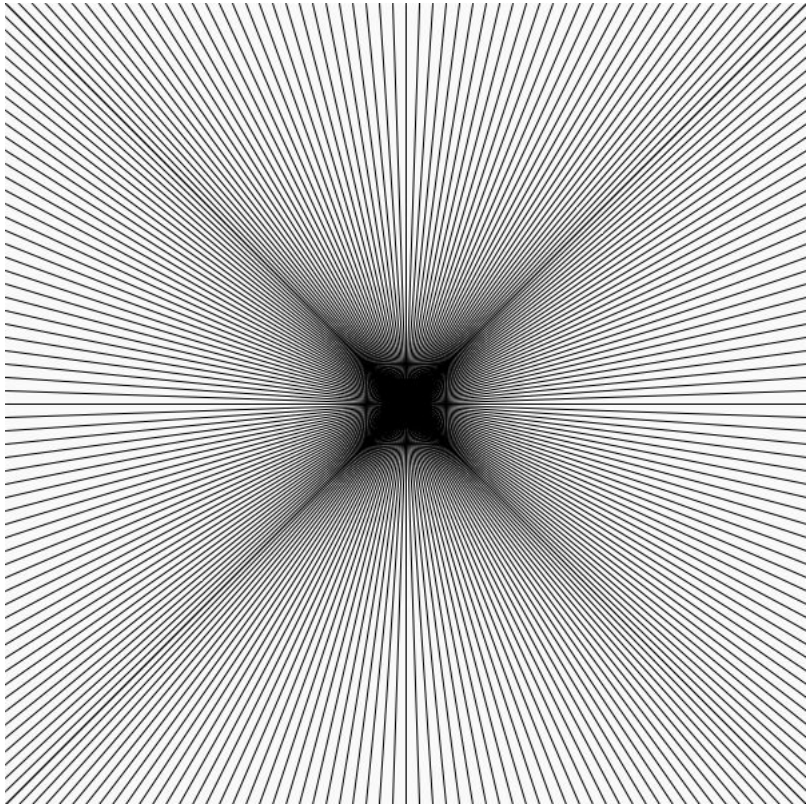


Problème du crenelage, anti-aliasing

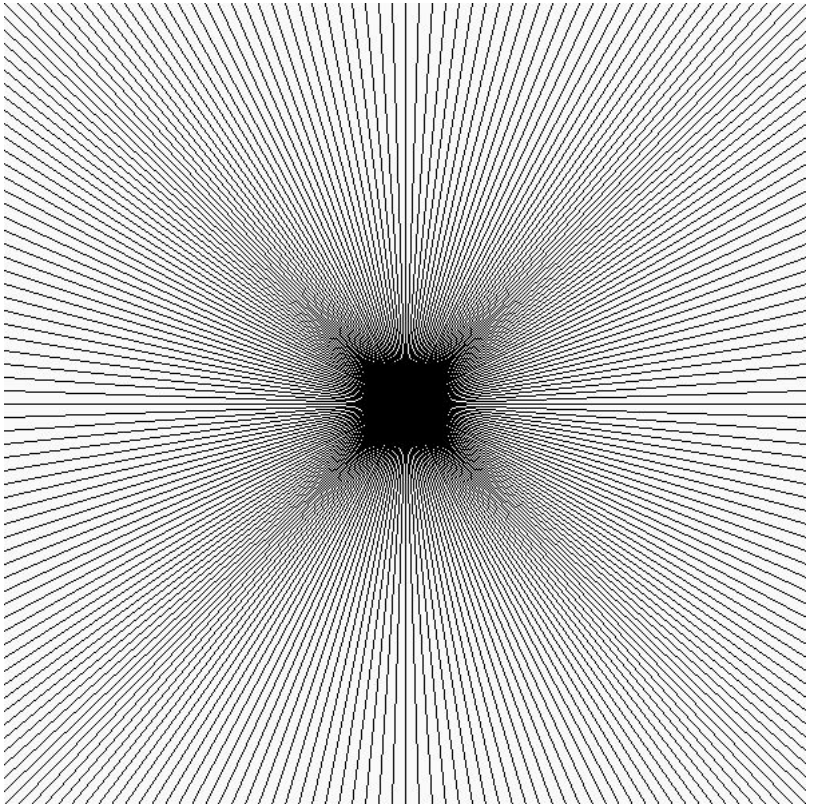


http://en.wikipedia.org/wiki/Line_drawing_algorithm#mediaviewer/File:Line_scan-conversion.svg

Traitement de l'aliasing en Processing : l'ordre smooth()



avec



sans

(3) Courbes paramétrées

- Le besoin : approximer et/ou interpoler des contours d'objets "réels"
- Critère esthétique : continuité du premier ordre (tangentes), du 2ème ordre (points d'inflexion)
- Flexibilité : courbes quadratique (ordre 2) insuffisantes

=> une solution : les courbes 2D polynomiales d'ordre 3, généralisées ensuite aux surfaces 3D

de Casteljau : Citroën 1959

Bézier : Renault 1962

Fergusson : Boeing 1964

Coons : MIT 1967

Catmull : Univ. Utah 1974

Définitions

• Courbe en 2D : localisation d'un point P qui se "déplace" dans le plan avec un degré de liberté :

$$p\begin{pmatrix} x \\ y \end{pmatrix} = f\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} f_x(u) \\ f_y(u) \end{pmatrix}$$

• Courbe en 3D : localisation d'un point P qui se "déplace" dans l'espace avec un degré de liberté :

$$p\begin{pmatrix} x \\ y \\ z \end{pmatrix} = f\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_x(u) \\ f_y(u) \\ f_z(u) \end{pmatrix}$$

• On préfère utiliser ici une forme *explicite* pour définir la surface, plutôt qu'une forme *implicite* comme : $f(x,y,z) = 0$

• On se restreint dans tout ce qui suit à des courbes définies sur des intervalles *bornés*

• On prendra toujours $0 \leq u \leq 1$

Courbes de Bézier

- Définition générale
  - Courbe lisse définie par N+1 points de contrôle P0,P1,…PN
  - Passe par les extrémitées P0 et PN
  - Passe au voisinage des autres points

$$p\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f_x(u) \\ f_y(u) \\ f_z(u) \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{N} B_i(u)x_i \\ \sum_{i=0}^{N} B_i(u)y_i \\ \sum_{i=0}^{N} B_i(u)z_i \end{pmatrix}$$

avec $B_i(u) = \dfrac{N!u^i(1-u)^{N-i}}{i!(N-i)!}$ polynômes de Bernstein

- Notation vectorielle plus commode : $\vec{p} = f(u) = \sum_{i=0}^{N} B_i(u)\vec{P}_i$

• En pratique, on prend N=3 et donc 4 points de contrôle pour définir la courbe.

La fonction f devient alors : $$f(u) = \sum_{i=0}^{3} B_i(u)\vec{P}_i$$
$$= \vec{P}_0(1-u)^3 + 3\vec{P}_1 u(1-u)^2 + 3\vec{P}_2 u^2(1-u) + \vec{P}_3 u^3$$

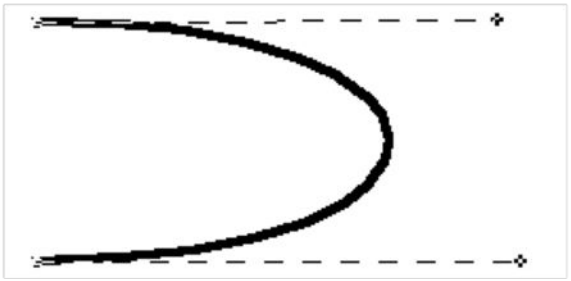
$$f(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \vec{P}_0 \\ \vec{P}_1 \\ \vec{P}_2 \\ \vec{P}_3 \end{pmatrix}$$

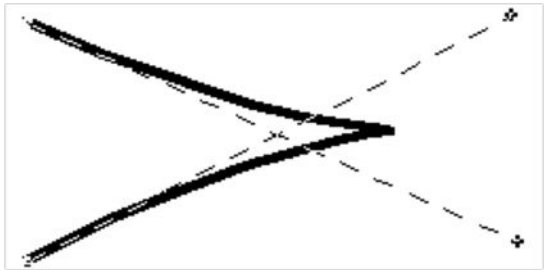
Notation matricielle plus commode : $$f(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} M_B \begin{pmatrix} \vec{P}_0 \\ \vec{P}_1 \\ \vec{P}_2 \\ \vec{P}_3 \end{pmatrix}$$

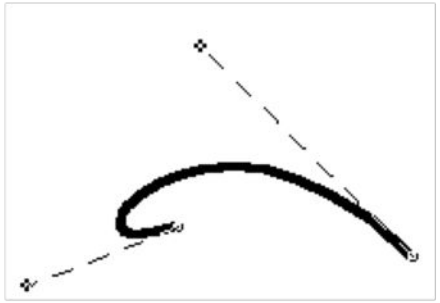
MB est la *matrice génératrice* des courbes de Bézier.

Exemple

Tracé sous MacDraw Pro :

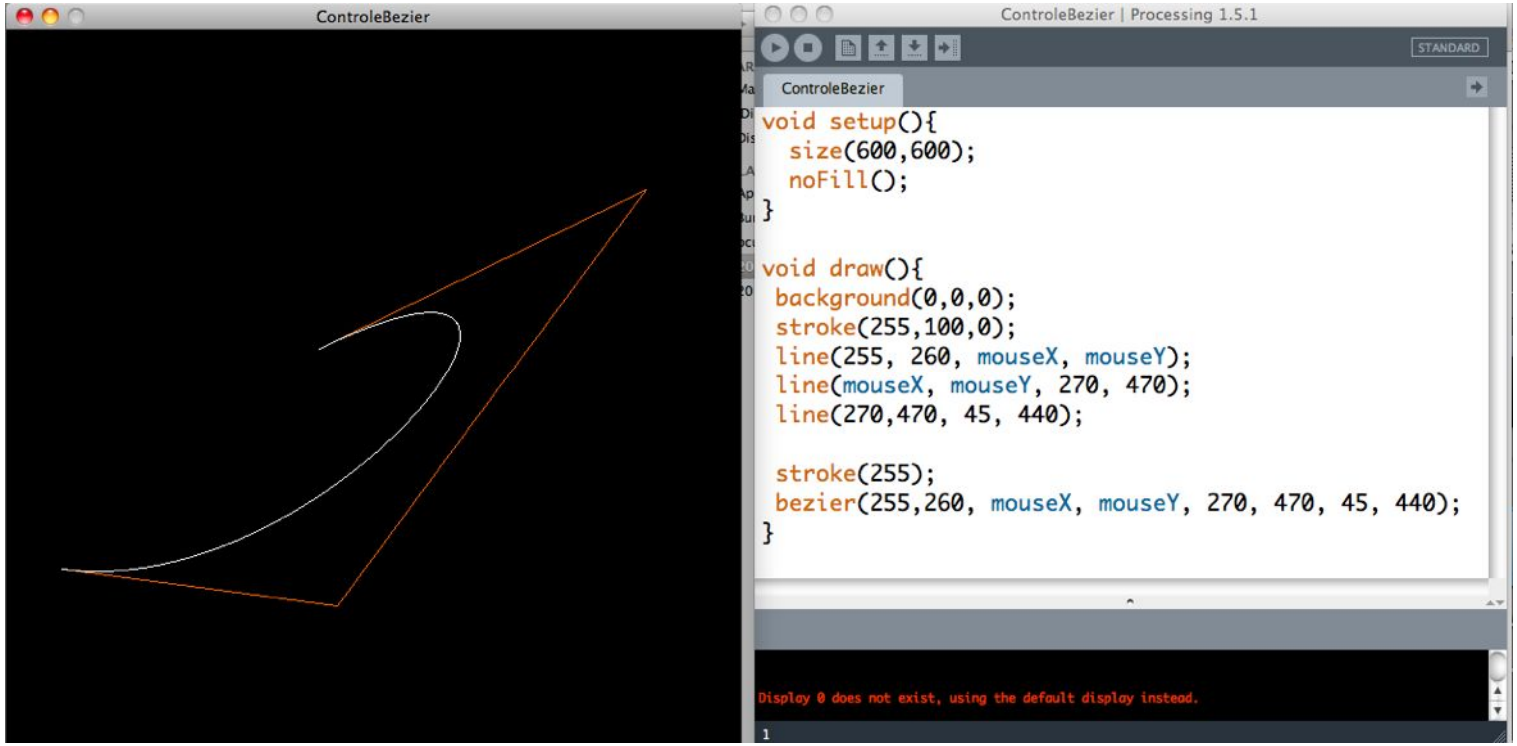






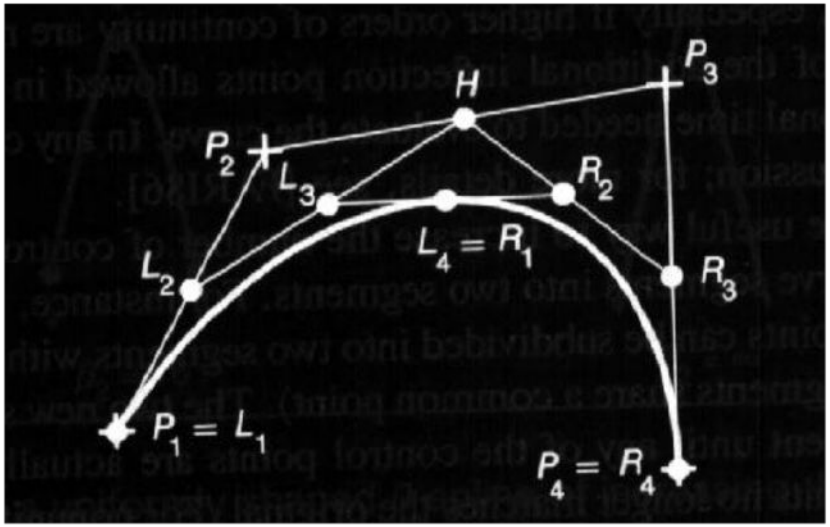
Les points de contrôle définissent l'enveloppe convexe de la courbe

Primitive bezier() de Processing



- Méthode de De Casteljau :

[FOLEY] p. 508



On calcule dans l'ordre :

(1) L1 = P1

(2) L2 = (P1+P2)/2

(3) H = (P2+P3)/2

(4) L3 = (L2+H)/2

(5) R3 = (P3+P4)/2

(6) R2 = (H+R3)/2

(7) R1 = (L3+R2)/2

(8) L4 = R1

Pas de multiplication, juste divisions par 2

Les Splines

- Inconvénients des Béziers

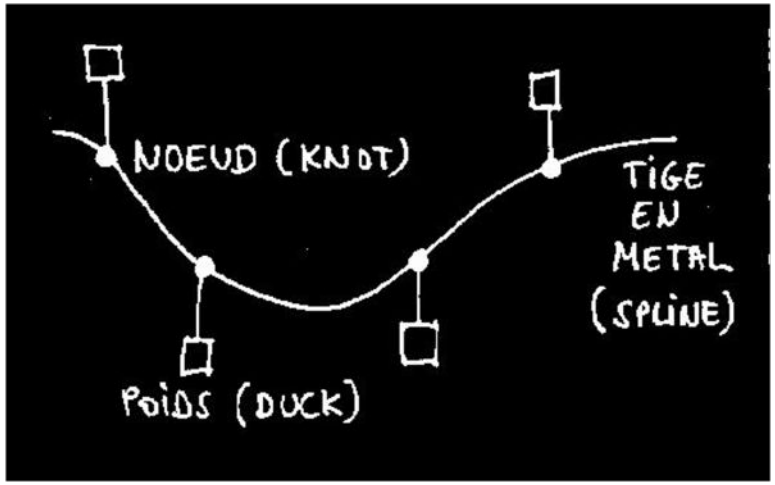
Si on a plus de 4 (16) points de contrôles, il faut générer 2 (4) courbes (surfaces)

Les tangentes sont définies aux extrémitées de la courbe de Bézier, mais pas la dérivée d'ordre 2 => point d'inflexions impossibles

Si un point de contrôle change légèrement de position, l'ensemble de la courbe (surface) est affectée

(mais faciles à dessiner...)

- Autre approche : les splines

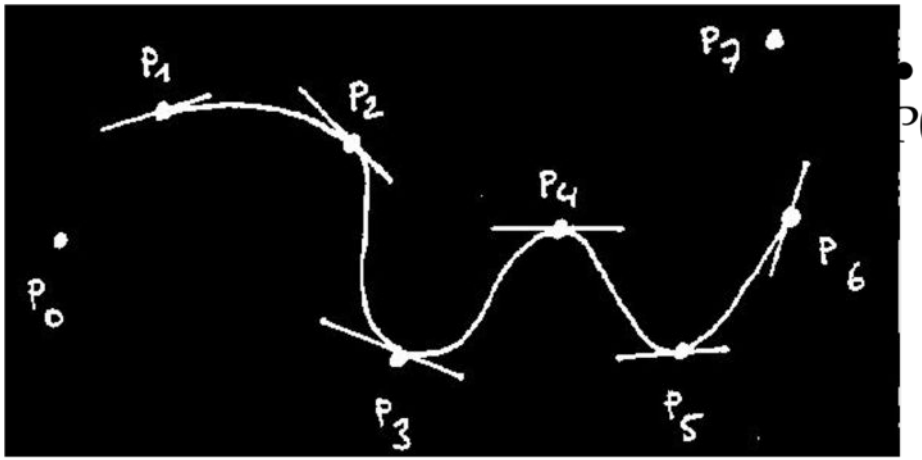


Utilisé pour le design des avions, des voitures et des coques de bateaux

Continuité d'ordre 2 possible aux jonctions

Splines de Catmull-Rom

- On connait N+2 points P(0) …P(N+1)
- On cherche une courbe *passant* par P(1) …P(N)
- La tangente au point P(i) est parallèle à la droite P(i-1), P(i+1)



- Chaque segment Si est défini par :

$$f_i(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \frac{1}{2} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} \vec{P}_{i-1} \\ \vec{P}_i \\ \vec{P}_{i+1} \\ \vec{P}_{i+2} \end{pmatrix}$$

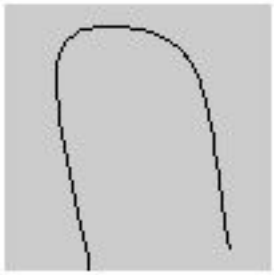
- Inconvénient : connaissance de P0 et P7 ??

Utilisation sous Processing :

Name

curveVertex()

Examples

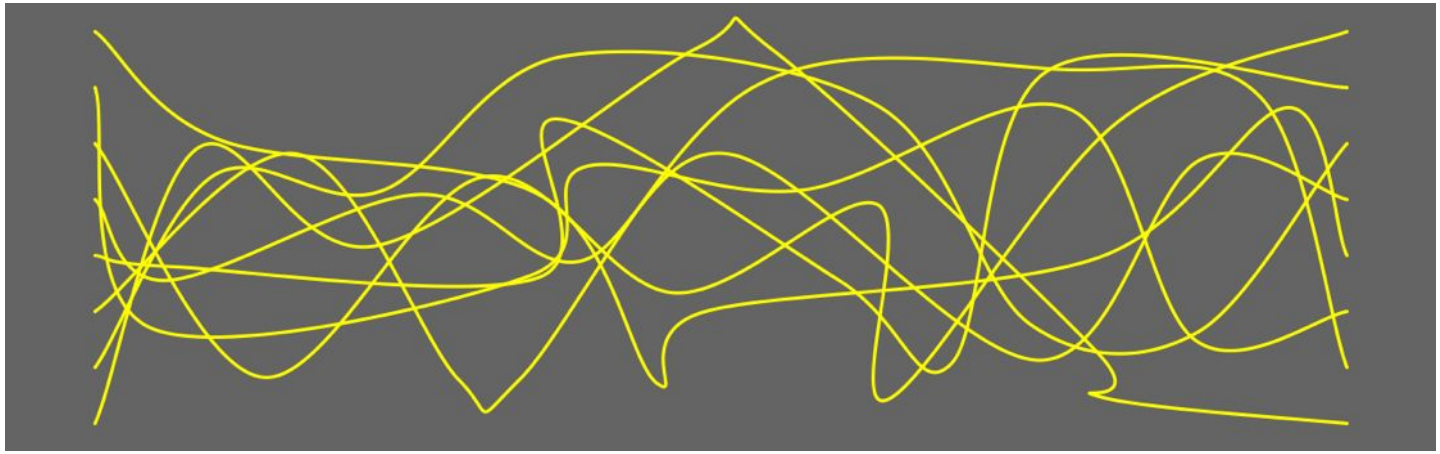


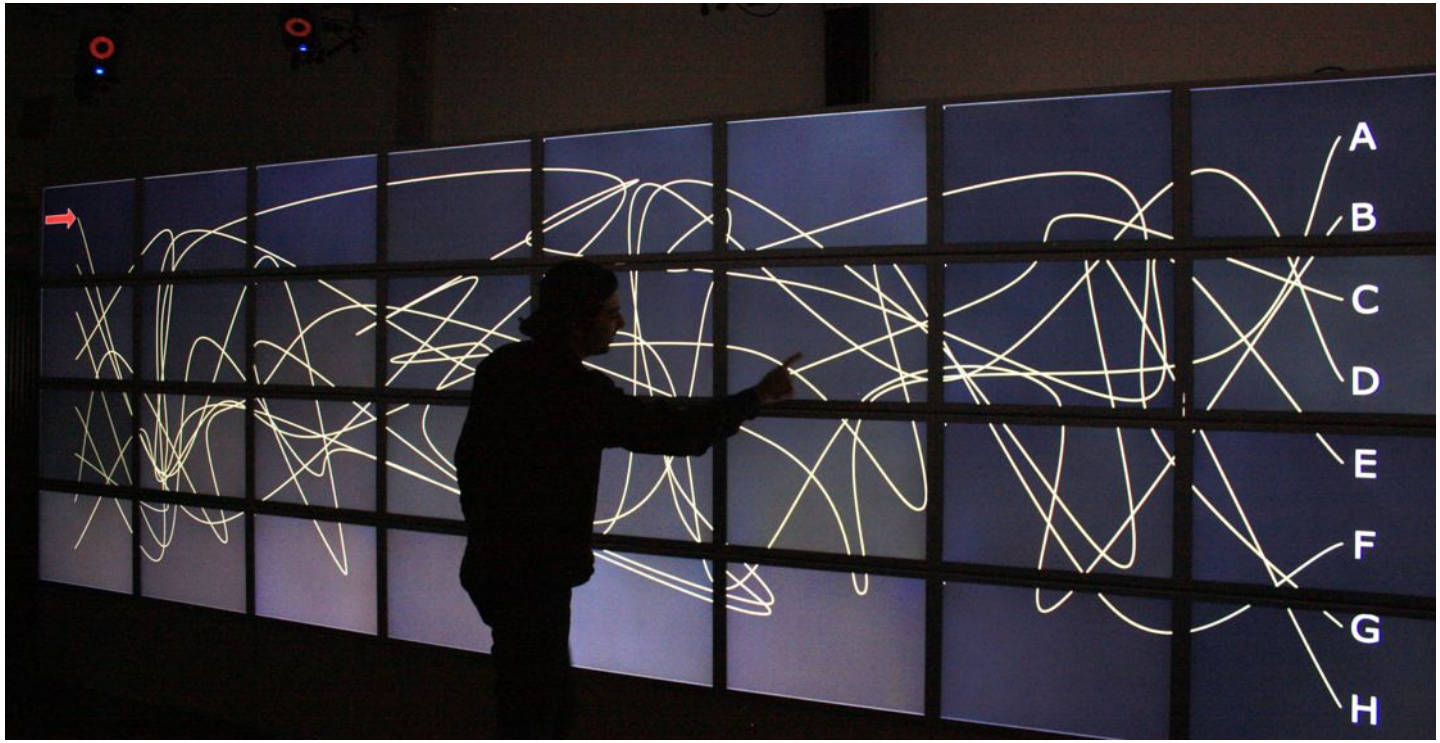
```
noFill();
beginShape();
curveVertex(84,  91);
curveVertex(84,  91);
curveVertex(68,  19);
curveVertex(21,  17);
curveVertex(32, 100);
curveVertex(32, 100);
endShape();
```

Description

Specifies vertex coordinates for curves. This function may only be used between **beginShape()** and **endShape()** and only when there is no MODE parameter specified to **beginShape()**. The first and last points in a series of **curveVertex()** lines will be used to guide the beginning and end of a the curve. A minimum of four points is required to draw a tiny curve between the second and third points. Adding a fifth point with **curveVertex()** will draw the curve between the second, third, and fourth points. The **curveVertex()** function is an implementation of Catmull-Rom splines. Using the 3D version of requires rendering with P3D or OPENGL (see the Environment reference for more information).

Demo





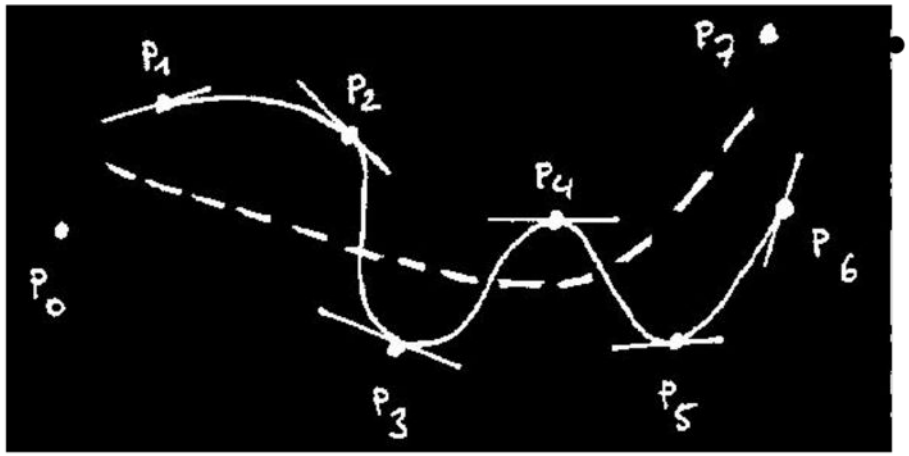
[Almeida et al., AVI'2012]

B-Splines uniformes

- On connait N points P(1) …P(N)
- On cherche une courbe *proche de* P(1) …P(N)
- Continuité d'ordre 1 et 2 d'un segment à l'autre

- Chaque segment Si est défini par :

$$f_i(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \frac{1}{6} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} \vec{P}_{i-1} \\ \vec{P}_i \\ \vec{P}_{i+1} \\ \vec{P}_{i+2} \end{pmatrix}$$



- Exemple :

Segment S3 = (P3,P4)

f3(0) = (P2 + 4P3 + P4)/6

f3(1) = (P3 + 4P4 + P5)/6

(4) Le remplissage

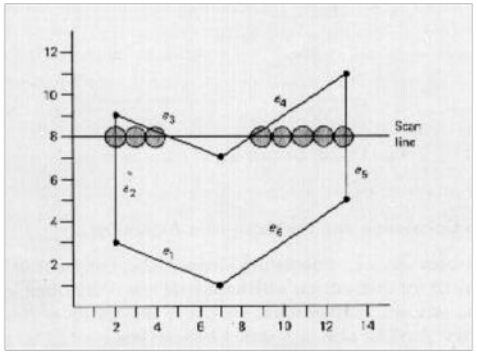
Un algorithme naïf : par inondation (flood fill)

floodfillINTER

```
void draw(){
 if (GOFILL)
  floodfill(38,38);
}

void floodfill(int x, int y) {
  color c = get(x,y);
  if (c == NOIR) {
    set(x,y,ROUGE);
    if (y>0) floodfill(x,y-1);
    if (y<height) floodfill(x,y+1);
    if (x>0) floodfill(x-1,y);
    if (x<width) floodfill(x+1,y);
  }
}

void keyReleased(){
  GOFILL = true;
}
```

Il faut se passer de la récursivité !

Balayage pour les polygones



[FOLEY83] p. 457

- On ne tient pas compte des arêtes horizontales

- Attention aux "pointes" (sommets extremums)

- Peut-être spécialisé si polygones = triangles

- Principe :

pour chaque ligne de balayage (coord. y) faire

Chercher les intersections de la ligne avec chaque arête du polygone

Trier les intersections par x croissant

pour chaque pixel compris entre 2 intersections

Comparaison avec Zbuffer

Coloriage

fin pour

fin pour

• Algorithme général

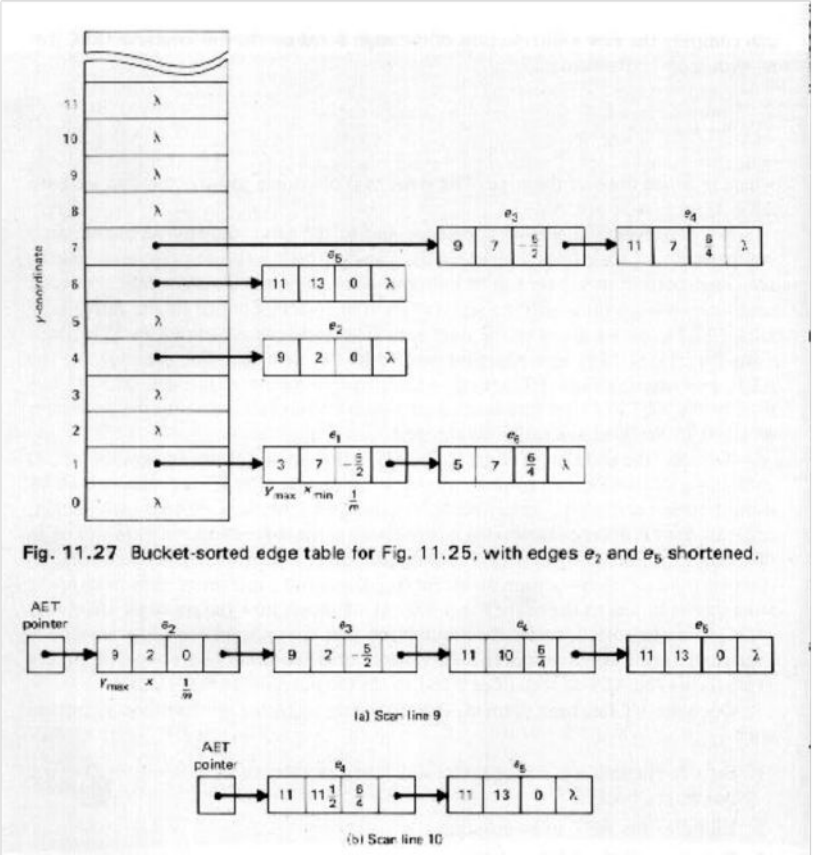


Fig. 11.27 Bucket-sorted edge table for Fig. 11.25, with edges e_2 and e_5 shortened.

[FOLEY ed. 1983 p. 460]

BET : bucket-sorted edge table
AET : active edge table

y := plus petite coord. en Y dans la BET

initialiser l'AET à vide

tant que BET et AET non vides faire

1) insérer dans l'AET par x croissant, les éléments de BET[y]

2) balayer les pixels entre les paires successives de l'AET (comparaison avec le Zbuffer + coloriage…)

3) retirer de l'AET les éléments dont y = ymax

4) pour chaque élément i de l'AET faire
AET[i].x = AET[i].x + AET[i].pente

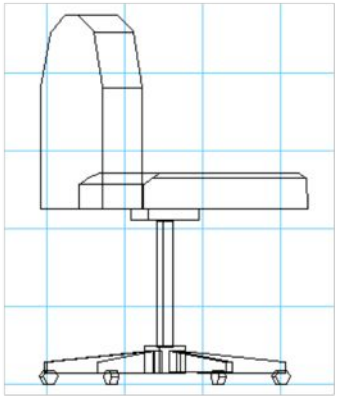
5) trier l'AET par x croissant

6) y = y + 1

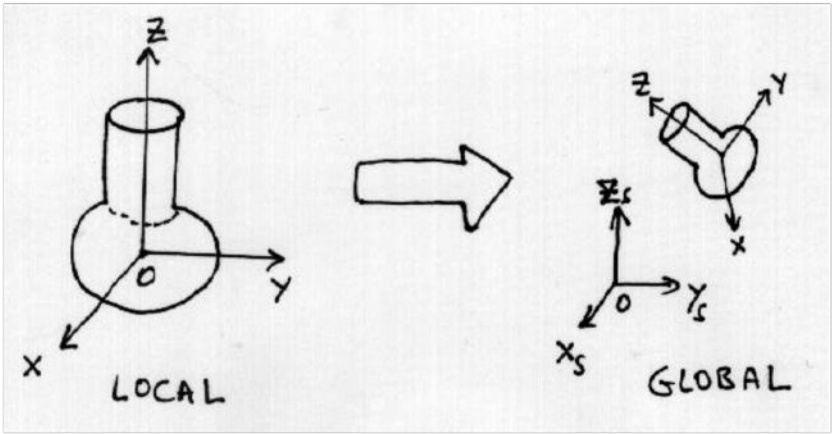
fin tant que

(5) Transformations géométriques

- Pour construire de nouveaux objets à partir de volumes élémentaires



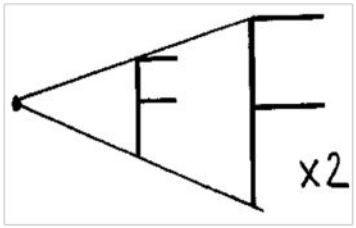
- Pour changer le système des coordonnées utilisées pour décrire un objet.



Différents types

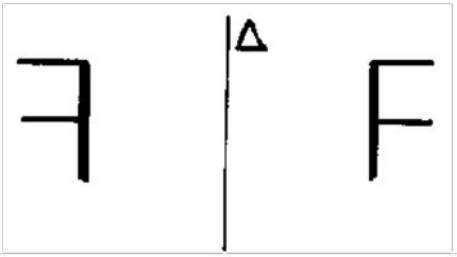
Translation

Homothétie

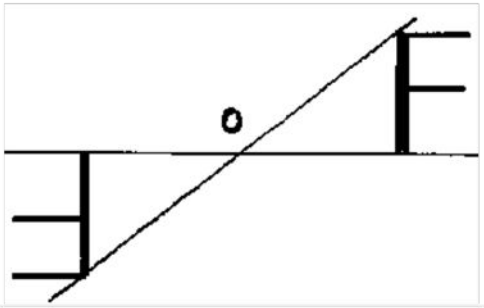


D'après [QUID] p. 278-9

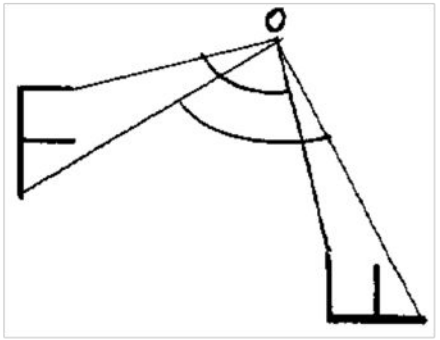
Symétrie orthogonale



Symétrie centrale



Rotation



v5 P. CUBAUD <cubaud@cnam.fr> Dept. informatique CNAM, Paris

Opérations matricielles (2D)

- Matrices homogènes

  - Toutes les opérations précédentes se ramènent à la multiplication d'une matrice par le vecteur des coordonnées de départ, SAUF pour la translation (addition de 2 vecteurs)

  - On va représenter un point P(x,y) par une famille de vecteurs de 3 éléments (x,y,W). W est le facteur d'échelle.

  Ainsi (x,y,W) et (x/W, y/W, 1) représentent le même point.

  - Si W=0, on dira que le point est à l'infini…

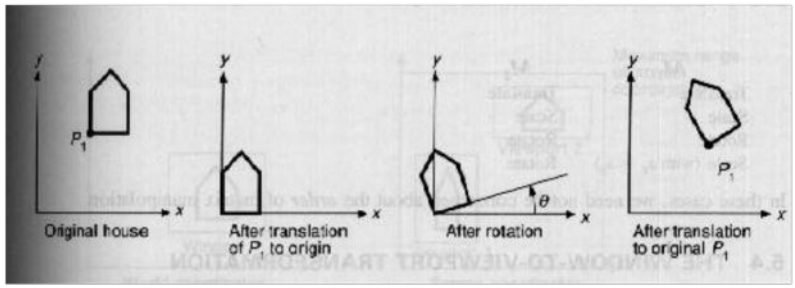
- Translation $\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$

- Homothétie $\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$

- Rotation par rapport à l'origine $\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$

Exemple de composition

On veut faire une rotation de la maison autour du point P1



[FOLEY] pp. 208-9

Suite d'opérations à réaliser :

Translation de l'objet à l'origine & Rotation & Translation de l'objet à sa position de départ

Matrices résultantes :

$$T(x_1,y_1)\circ R(\theta)\circ T(-x_1,-y_1)$$

$$=\begin{pmatrix}1&0&x_1\\0&1&y_1\\0&0&1\end{pmatrix}\times\begin{pmatrix}\cos\theta&-\sin\theta&0\\\sin\theta&\cos\theta&0\\0&0&1\end{pmatrix}\times\begin{pmatrix}1&0&-x_1\\0&1&-y_1\\0&0&1\end{pmatrix}$$

$$=\begin{pmatrix}\cos\theta&-\sin\theta&x_1(1-\cos\theta)+y_1\sin\theta\\\sin\theta&\cos\theta&y_1(1-\cos\theta)-x_1\sin\theta\\0&0&1\end{pmatrix}$$

Attention à l'ordre !!

Commutativité pas toujours vérifiée : T.R≠R.T

Exemple de base en processing

rotation §

```
void draw(){
  //background(200);
  translate(width/2, height/2);
  float a=map(mouseX,0,width,0,PI/2);
  rotate(a);
  rect(-26, -26, 52, 52);
}
```

Un exemple plus complet :

http://joyofprocessing.com/blog/2011/11/stars-and-asters/