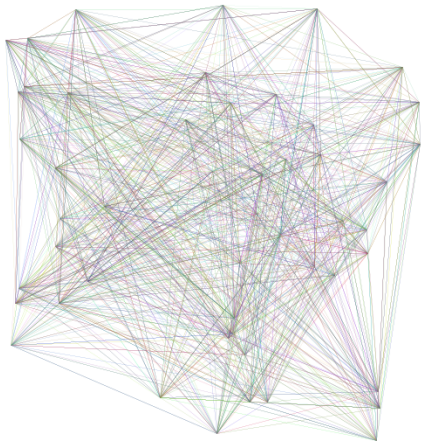
processing

Ateliers Processing de l'OA

Sketch 01



L'idée de ce premier atelier était d'implémenter un algorithme d'art contemporain proposé par Sol LeWitt.

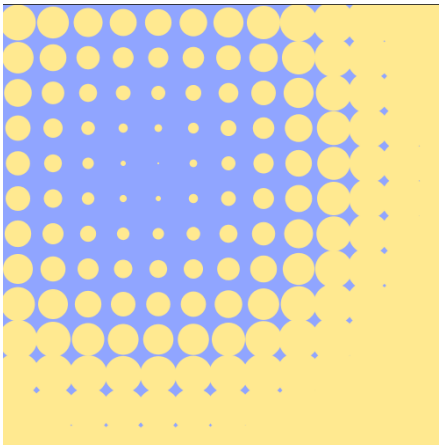
L'idée a été trouvée sur le site de Pol Guezennec

Bon, c'est vrai qu'on commence sur des chapeaux de roues, avec l'utilisation des boucles `for` et des listes, mais j'essaierai de garder un niveau de complexité constant, afin de ne pas pénaliser ceux·elles qui raccrocheraient le wagon en cours d'année. Si ce premier sketch vous semble compliqué (et il l'est lorsqu'on débute), les suivants devraient vous paraître de plus en plus simples, à force de répétition.

```
FloatList liste_x = new FloatList();
FloatList liste_y = new FloatList();

void setup() {
  // Dans la fonction setup on mets les instructions qui n'ont
  // besoin d'être exécutés qu'une seule fois, au démarrage

  size(500, 500);
  background(255);

  for (int i = 0; i < 50; i = i+1) {
    liste_x.append(random(width));
    liste_y.append(random(height));
  }
}

void draw() {
  // La fonction draw s'exécute à chaque rafraichissement de l'écran (60 fois/secondes par défaut)

  stroke(random(255), random(255), random(255));  // Couleur des contours
  strokeWeight(0.1);                              // Épaisseur des countours
  int i0 = int(random(50));
  int i1 = int(random(50));
  line(liste_x.get(i0), liste_y.get(i0), liste_x.get(i1), liste_y.get(i1));
}
```

Sketch 02

Ici nous abordons les boucles “for” pour répéter un bloc d'instructions. Nous imbriquons deux boucles “for” pour créer la grille sur deux dimensions.



```
int diametre = 40;  // Diamètre des cercles

void setup() {
  size(500, 500);
  noStroke(); // Désactive le countour des formes
  fill(#FFE990); // Couleur de remplissage des cercles
}

void draw() {
  background(#90A5FF); // On repeint le fond

  for (int j = 0; j < height; j += diametre) {
    // A chaque tour de la boucle externe on descend d'une ligne
    for (int i = 0; i < width; i += diametre) {
      // A chaque tour de la boucle interne on décalle d'une colonne
      int posx = i + diametre/2;
      int posy = j + diametre/2;
      // On calcule la distance entre le centre de chaque cercle et le curseur de la souris
      float d = dist(posx, posy, mouseX, mouseY);
      circle(posx, posy, d * 0.18);
    }
  }
}
```

sketch_02.mp4

Sketch 03

Pour sortir de la monotonie des lignes droites, essayons-nous aux courbes !

Première forme

Ce sketch est interactif. Cliquez dans la fenêtre pour rajouter des points d'ancrages à la courbe.

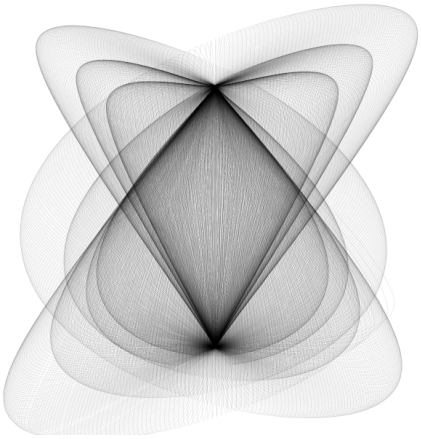
```
ArrayList<PVector> points = new ArrayList();

void setup() {
  size(500, 500);
  noFill();
}

void draw() {
  background(255);
  beginShape();
  curveVertex(0, 0);  // On rajoute un premier point de contrôle aux mêmes coordonnées que le premier point d'ancrage de la courbe
  curveVertex(0, 0);

  for (PVector p : points) {
    p.x = p.x + random(-1,1)*2;  // On modifie légèrement les coordonnées de chaque points pour l'effet de vibration
    p.y = p.y + random(-1,1)*2;
    curveVertex(p.x, p.y);
  }

  curveVertex(width, height);
  curveVertex(width, height);  // Un dernier point de contrôle pour terminer la courbe
  endShape();

  for (PVector p : points) {
    circle(p.x, p.y, 10);
  }
}

void mousePressed() {
  points.add(new PVector(mouseX, mouseY));  // Chaque clique ajoute un nouveau points aux coordonnées du curseur de la souris
```

```
}
```

Seconde forme

Dans le style de l'harmonographe.

```
ArrayList<PVector> list = new ArrayList();

void setup() {
  size(500, 500);
  background(255);
  strokeWeight(0.2);
}

void draw() {
  float x = 200 * cos(millis() * 0.005);
  float y = 200 * sin(millis() * 0.003);
  fill(0, 0);
  //background(255);
  beginShape();
  curveVertex(width*0.5, 50);
  curveVertex(width*0.5, 50);
  curveVertex(width*0.5 + x, height*0.5 + y);
  curveVertex(width*0.5, height-50);
  curveVertex(width*0.5, height-50);
  endShape();
}

void mouseClicked() {
  list.add(new PVector(mouseX, mouseY));
}
```

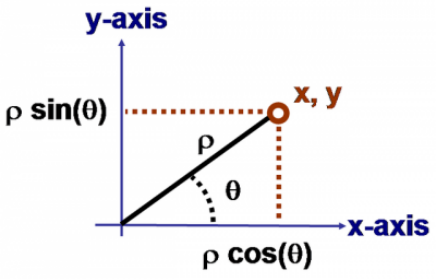


Sketch 04 : Spirale Polaire

Je ne m'attendais pas à voir venir beaucoup de monde le 4 Janvier, pour le premier atelier Processing de cette nouvelle année. Puisqu'à l'heure prévue il n'y avait qu'Alex et moi, j'ai voulu proposer quelque chose d'un peu plus complexe que d'habitude. L'idée était de créer des spirales denses, à la façon des sillons de disques vinyle.

La méthode la plus “simple” (à condition de connaître un peu de trigonométrie) est de faire usage des coordonnées polaires. Tout à fait approprié dans les conditions arctiques que nous avons actuellement à la Baleine. Ne vous laissez pas intimider par ces mathématiques froides et souvenez-vous que l'essentiel est de dessiner des jolis trucs à l'écran.

Le mini cours de trigo sur les coordonnées polaires



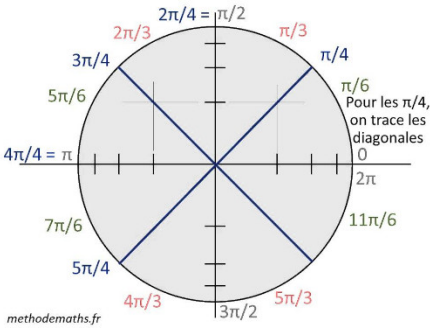
Pour décrire la position d'un point dans un espace en deux dimensions, on a l'habite d'utiliser les **coordonnées cartésiennes** `(x,y)` où x représente la distance depuis l'origine sur l'axe horizontal et y la distance sur l'axe vertical. Sur la figure précédente, l'origine est en bas à gauche du repère. Dans Processing l'origine du repère cartésien se trouve en haut à gauche. Normalement là je ne vous apprend rien.

Une autre façon pour décrire la position d'un point est par les **coordonnées polaires** (φ,θ), ou φ est la distance ~~euclidienne~~ *à vol d'oiseau* entre l'origine et notre point, et θ est l'angle entre la droite *origine-point* et l'axe horizontal.

Si on imagine un cercle centré sur l'origine du repère et traversant le point p, alors la distance φ est égal au rayon de ce cercle. D'ailleurs les lettres grecs sont un peu pénibles à taper au clavier alors on utilisera plutôt les lettres `(r,a)` pour nos coordonnées polaires.

On peut aussi imaginer le cadran d'un horloge avec les chiffres des heures situés sur le périmètre du cercle. Dans ce cas toutes les heures on la même coordonnée `r` (elles sont toutes à la même **distance du centre**, correspondante au rayon du cadran) mais elles ont toutes une coordonnée `a` (angle) différente. L'angle 0 (zéro) se situerait à 3 heures. “12h” aurait l'angle +90° et “9h” aurait l'angle -90°. Si on fait un tour complet (360°) on revient sur le même point, donc les coordonnées `(r, 10°)` et `(r, 370°)` décrivent exactement la même position.

Bon alors il y a une subtilité : en trigonométrie on ne compte pas les angles en degrés comme tout le monde, mais en **radians**, qui permettent de donner un angle en fraction de PI. Un tour de cercle complet (360°) fait 2×PI radians. Vous l'aurez deviné, un demi tour de cercle (180°) fait donc PI radians. Sur notre cadran d'horloge, “12h” est à l'angle PI/2 radians et “9h” est à l'angle -PI/2 radians (ou bien (3/4)×PI, si on tourne toujours dans le même sens). C'est le fameux **cercle trigonométrique**, où l'angle croît dans le sens anti-horaire.



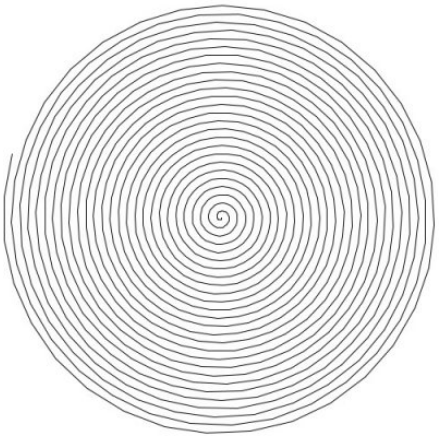
J'espère que vous n'avez pas la tête qui tourne trop, car il y encore une autre subtilité. Vous vous souvenez peut-être que dans Processing (et beaucoup d'autres environnement de programmation), l'axe y est inversé (y grandit de haut en bas) ? Et bien c'est la même chose pour le sens de rotation. Sous processing, l'angle grandit dans le sens horaire, contrairement aux conventions mathématiques !

Enfin, connaissant les coordonnées polaires un point, on peut calculer ses coordonnées cartésiennes (essentiel pour se situer sur une grille de pixels, comme celle de notre fenêtre graphique) en appliquant les formules suivantes :

`x = r × cos(a)` et `y = r × sin(a)`, où x et y sont nos coordonnées cartésiennes et `r` et `a` sont nos coordonnées polaires (avec `a` en radian bien entendu). Les fonctions `cos()` et `sin()` se trouvent comme telles dans Processing, et on a même les fonctions `radians()` pour convertir les angles en degrés vers radians, et `degrees()` pour convertir les angles en radians vers degrés.

Ouf ! On va enfin pouvoir programmer !

Première forme



```
void setup() {
  size(500, 500);
}

void draw() {
  background(255);

  translate(width/2, height/2); // Pour déplacer l'origine au milieu de la fenêtre

  // On initialise les variables dont on aura besoin
  PVector p1 = new PVector();  // Un premier point
  PVector p2 = new PVector();  // Un deuxième point
  float angle = 0.0;           // l'angle actuel (en radians)
  float radius = 0.0;          // le rayon actuel

  while (radius < width*0.5) {
    p1.set(radius * cos(angle), radius * sin(angle));  // On définit un premier point aux coordonnées actuelles
    angle += 0.2f;                                     // On augmente légèrement l'angle (en radians)
    radius += 0.3f;                                    // et le rayon
    p2.set(radius * cos(angle), radius * sin(angle));  // On définit le deuxième point aux nouvelles coordonnées
    line(p1.x, p1.y, p2.x, p2.y);                      // On trace une ligne entre nos deux points

    // Et on recommence ! (tant que le rayon est inférieur à un certain seuil)
  }
}


void keyPressed() {
  // Pratique pour exporter des captures d'écran
  // (elles seront placés dans le sous-dossier "data" du sketch)
  // On peut ouvrir le dossier du sketch avec le raccourci Ctrl+K
  if (key == 'p') {
    saveFrame("####.png");
  }
}
```

Seconde forme

```
import peasy.*;  // On importe la librairie peasyCam

PeasyCam cam;    // Cette variable va contenir les infos de notre caméra

void setup() {
  size(800, 800, P3D);            // On choisit le moteur de rendu "P3D" pour la 3D
  cam = new PeasyCam(this, 400);  // On crée la caméra
}

void draw() {
  background(255);

  PVector p1 = new PVector();
  PVector p2 = new PVector();
  float angle = 0.0;
  float radius = 0.0;

  //strokeWeight(2.0);
  while (radius < width*0.5) {
    p1.set(radius * cos(angle), radius * sin(angle));
    angle += 0.1f;
    radius += 0.02f;
    p2.set(radius * cos(angle), radius * sin(angle));
    float seg_angle = sin(2 * atan2(p2.y-p1.y, p2.x-p1.x));  // Un peu de magie trigonométrique
    stroke(seg_angle*200);                                   // Qui colore les segments en fonction de l'angle que forment les deux points
    line(p1.x, p1.y, p2.x, p2.y);
  }
}
```