UNIVERSITÉ PARIS SUD

Numéro de copie :

Licence MPI. Info 112 (IGOR)

F. Vernier, C. Jacquemin et M. Thiliti

Examen 2016-2017 (2h, barème /20)

1 Dessin Récursif (7 pts)

Dans cet exercice, il vous est demandé de fournir un certain nombre d'éléments nécessaires à la réalisation d'une application écrite en langage Processing. Nous partirons du programme ci-dessous qui donne le résultat ci-contre.

```
1.  void setup(){
2.  size(1000, 1000);
3.  noLoop();
4.  }

5.  void drawR(int n, int x1, int y1, int x2, int y2, int x3, int y3){
6.    // Partie 1
7.    int xa = (2*x1+x2+x3)/4;
8.    int ya = (2*y1+y2+y3)/4;
9.    int xb = (x1+2*x2+x3)/4;
10.   int yb = (y1+2*y2+y3)/4;
11.   int xc = (x1+x2+2*x3)/4;
12.   int yc = (y1+y2+2*y3)/4;

13.   // Partie 2
14.   fill(0, 48);
15.   beginShape();
16.    vertex (x1, y1);
17.    vertex (x2, y2);
18.    vertex (x3, y3);
19.   endShape(CLOSE);

20.   // Partie 3
21.   if (n>0) {
22.     drawR(n-1, x1, y1, (x2*3+x1)/4, (y2*3+y1)/4, xa, ya);
23.     drawR(n-1, x2, y2, (x3*3+x2)/4, (y3*3+y2)/4, xb, yb);
24.     drawR(n-1, x3, y3, (x1*3+x3)/4, (y1*3+y3)/4, xc, yc);
25.     drawR(n-2, xa, ya, xb, yb, xc, yc);
26.   }
27. }

28. void draw(){
29. // Partie 4
30. drawR(3, width/2, 5, 5, height-100, width-5, height-100);
31. }
```

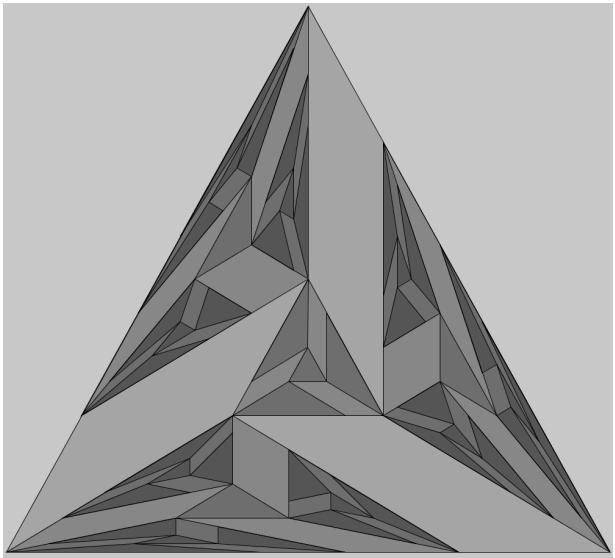


Figure 1 : Triangle

1.1 ***(2 pts)*** *Décrivez en 1 ligne le rôle de chaque partie du code.*

Partie 1 : calcul des points à l'intérieur du triangle
Partie 2 : dessin du triangle
Partie 3 : appels récursifs
Partie 4 : appel initial

1.2 ***(1 pts)*** *Quelle principe de mathématique utilise la partie 1.*

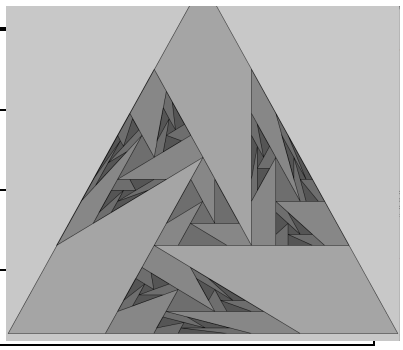
Une moyenne pondérée

1.3 ***(1pt)*** *Indiquez (sur la figure si possible) le point xa, ya lors du premier appel à drawR*

Voir figure au dessus du centre

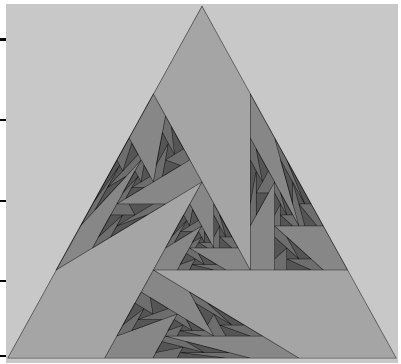
1.4 ***(2pt)*** *que modifier aux premiers paramètres des lignes 22 à 25 pour obtenir la nouvelle figure ci-dessous ?*

drawR(n-1, (x1*3+x2)/4, (y1*3+y2)/4,
drawR(n-1, (x2*3+x3)/4, (y2*3+y3)/4,
drawR(n-1, (x3*3+x1)/4, (y3*3+y1)/4,
aux début des lignes 22, 23 et 24



1.5 ***(1pt)*** *que modifier aux premiers paramètres des lignes 22 à 25 pour obtenir la nouvelle figure ci-dessous ?*

A la ligne 25 on met n-1 au lieu de n-2
drawR(n-1, xa, ya, xb, yb, xc, yc)



2 Traitement des pixels (7 pts)

Nous partirons du programme ci-dessous qui produit les cercles concentriques de la Figure 2.1.

```
1.  void draw(){
2.   loadPixels();
3.   for (int j=0; j<height; j++) {
4.    for (int i=0; i<width; i++) {
5.      float d1 = dist(i,j, width/2, height/2) ;
6.       pixels[i+j*height] = color(255*sin(d1*PI/100)*sin(d1*PI/100));
7.     }
8.   }
9.   updatePixels();
10. }
```

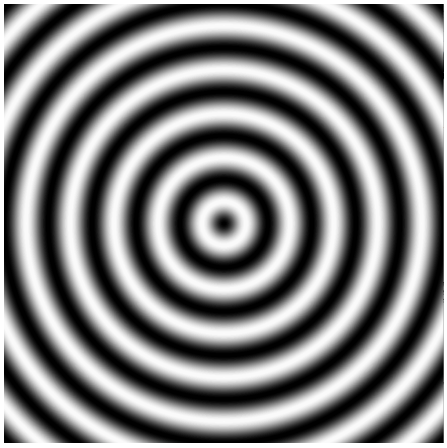


Figure 2.1 : cercles concentriques

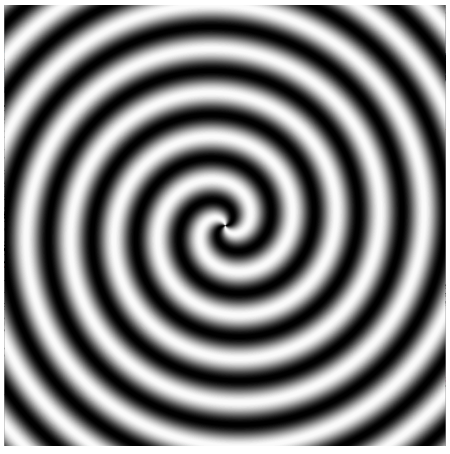


Figure 2.2 : spirale

Le programme ci dessous dessine la spirale de l'image 2.2

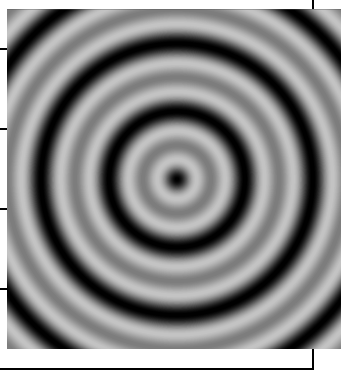
```
1.  void draw(){
2.   loadPixels();
3.   for (int j=0; j<height; j++) {
4.    for (int i=0; i<width; i++) {
5.      float d2 = dist(i,j, width/2, height/2)+
6.                 atan2(j- height/2, i-width/2)/PI*100;
7.       pixels[i+j*height] = color(255*sin(d2*PI/100)*sin(d2*PI/100));
8.     }
9.   }
10.  updatePixels();
11. }
```

2.1 ***(1 pts)*** *Que faut-il changer dans le premier code pour avoir 2 fois moins de cercles concentriques à l'écran ?*

Remplacer 100 par 200 a la ligne6

2.2 ***(2 pts)*** *Comment obtenir autant de cercles concentriques que la figure 2.1 mais un cercle sur deux sera plus clair que la figure 2.1 (utilisez votre réponse à la question précédente) ?*

color(127*sin(d1*PI/100)*sin(d1*PI/100) +
127*sin(d1*PI/200)*sin(d1*PI/200));

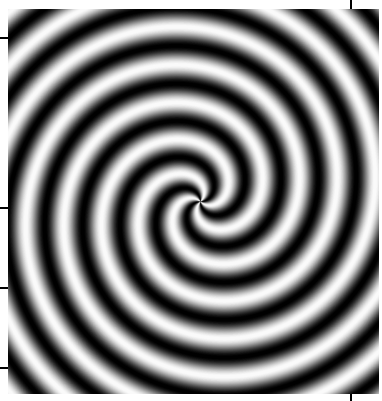


2.3 ***(1 pts)*** *Que calcul atan2(`j- height/2, i-width/2`)? A la ligne 6 du deuxième code ?*

l'angle entre i,j et le centre de l'écran

2.4 ***(1 pts)*** *Quel principe de math est a l'œuvre dans la suite de la formule (/PI*100)?*

La règle de trois (des proportions)



2.5 ***(2 pts)*** *comment modifier le deuxième code pour obtenir plus de spirales (atan2 doit « décaler » chaque cercle deux fois plus loin pour le connecter avec un autre cercle et créer ainsi 2 spirales imbriquées)*

On remplace de 100 de la ligne 6 par 200.

3 La toile d'araignée (5 pts)

Soit le code suivant en langage Processing :

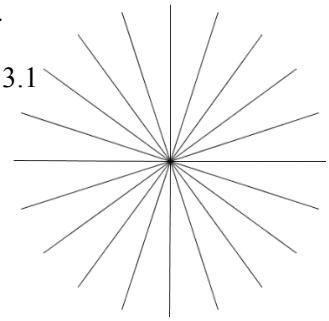
```
void setup() {
  size(600, 600);
  background(255);
  noLoop();
}
void draw() {
  int nombre_fils = 20;
  int r = 250;
  int nombre_cercles = 7;
  for(int i=0; i<nombre_fils; i++){
    …
  }
}
```

Le code suivant

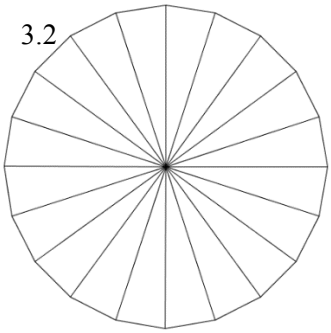
```
beginShape();
vertex (x1, y1);
bezierVertex (xa, ya, xb, yb, x2, y2);
endShape();
```

dessine une courbe de Bézier entres les points x1,y1 et x2, y2. En x1,y1 la tangente est orientée vers xa,ya. En x2, y2 la tangente est formée par un vecteur qui va de xb,yb vers x2, y2

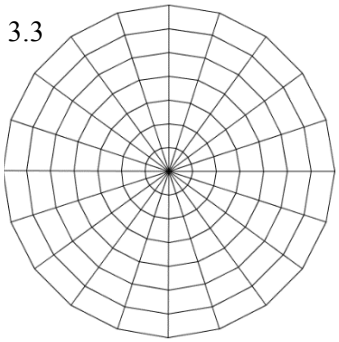
3.1



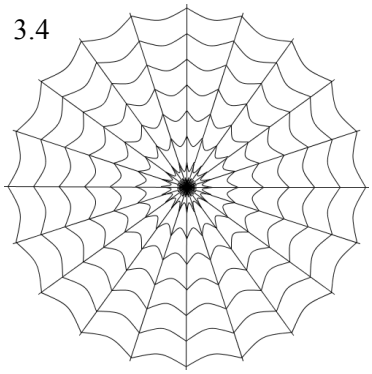
3.2



3.3



3.4



3.1 (1.5 pt) *Compléter la boucle `for` d'un `line(…)` pour obtenir la figure 3.1, c'est-à-dire une ligne partant du centre de la fenêtre au bout du rayon de longueur `r`.*

```
float a = i* (2*PI)/nombre_fils ;
line(width/2, height/2, width/2+r*cos(a), height/2+r*sin(a)) ;
```

3.2 (1.5 pts) *Rajouter un autre `line(…)` pour obtenir la figure 3.2, où chaque extrémité d'un fil est reliée à la suivante.*

```
float a2 = a+(2*PI)/nombre_fils;
line(width/2+r*cos(a), height/2+r*sin(a) ,
     width/2+r*cos(a2), height/2+r*sin(a2));

```

3.3 (2pts) *Remplacer le code de la question 3.2 pour obtenir la figure 3.3 avec nbc cercles.*

```
float a2 = a+(2*PI)/nombre_fils;
for (float r2 = 0; r2<=r; r2 += r/ nombre_cercles) {
  line(width/2+r2*cos(a), height/2+r2*sin(a) ,
       width/2+r2*cos(a2), height/2+r2*sin(a2));
}
```

3.4 (2pts) *Sur votre copie d'examen donnez un code qui produit un effet de toile d'araignée comme à la dernière figure en transformant les lignes en courbes de Bézier (Figure 3.4).*

```
beginShape( );
  vertex(width/2+r2*cos(a), height/2+r2*sin(a));
  bezierVertex(width/2+r3*cos(a3), height/2+r3*sin(a3),
               width/2+r3*cos(a3), height/2+r3*sin(a3),
               width/2+r2*cos(a2), height/2+r2*sin(a2));
endShape();
}
```