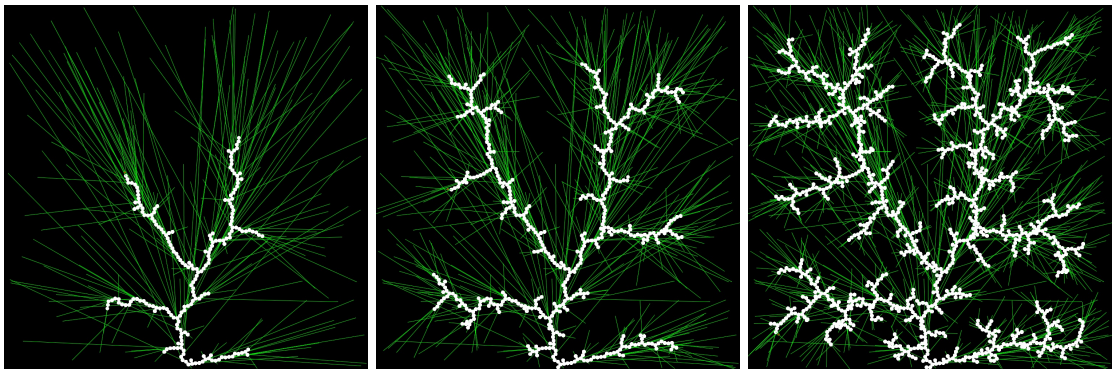
Nantes Université

proposé par Eric Languénou
UFR Sciences – Dpt d'Informatique

Création numérique
Des arbres attractifs



1 Travail en distanciel

Ce travail est à faire en autonomie, seul(e), à partir du cours magistral, des travaux dirigés et des travaux pratiques. Il est à rendre sur madoc dans la zone de dépôt correspondante, partie création numérique. Ce travail non obligatoire pourra ajouter jusqu'à 2 points sur la note de contrôle continu, qui ne dépassera pas 20/20.

2 Sujet : des arbres attractifs

Il s'agit d'étudier et de coder un algorithme en java sous processing qui implémente la création d'arbres aléatoires par ajout successifs de disques.

1. un premier disque est placé au milieu, en bas de l'écran
2. à chaque nouvelle image, on ajoute un nouveau disque j. Pour cela :
   — une position 2D A est tirée au hasard dans l'écran.
   — à partir de cette position A, le disque i dont le centre est le plus proche de A est identifié parmi les disques déjà ajoutés. Soit P_i son centre.
   — la nouvelle position P_j d'ajout de ce disque j est calculée en trouvant le point P, sur le vecteur $\overrightarrow{AP_i}$ qui est tel que le nouveau disque sera tangent au disque i

Sur les figures ci-dessus les vecteurs $\overrightarrow{AP_i}$ correspondent aux lignes vertes.

3 Gestion de l'arbre

Dans la suite, et pour commencer, vous étudierez et coderez :
— le stockage des disques
— l'ajout d'un nouveau disque
— l'affichage des disques.

3.1 Stockage de l'arbre et donc des disques

1. Vous utiliserez une liste, stockant les centres des disques, le diamètre des disques est défini en début de *sketch* (le nom donné à un code processing). La position du disque i sera donnée par le i^e élément de la liste de positions. Ci-dessous, quelques exemples d'utilisation de listes (pour cette question et les suivantes) :

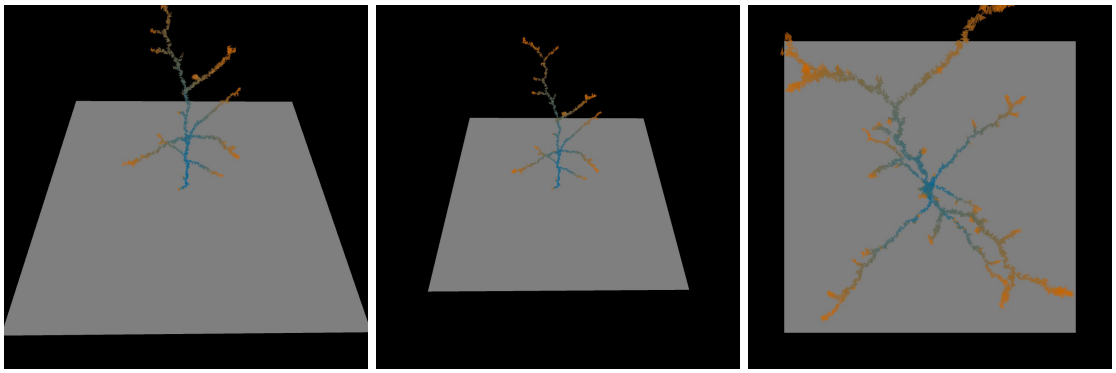
```
//.............declaration
List<PVector> posList; // la liste des positions des agents
//.............  initialisation (new arrayList() efface aussi la liste au besoin)
posList = new ArrayList<PVector>() ;
//.............  ajout d un element dans une liste
posList.add(P); // si P est un PVector
//.............  recuperation de l element a la place i
P = posList.get(i)
//..............  modification de la position de l element i
posList.get(i).add(addP);// si addP est un PVector
```

2. écrire une fonction `initArbre` qui allouera la liste et ajoutera le premier disque. Bien entendu, si vous êtes à l'aise avec les classes java, n'hésitez pas à en créer.
3. écrire une fonction qui prend la postion tirée aléatoirement et rend l'emplacement calculé du nouveau disque.

3.2 Affichage de l'arbre

— écrire une fonction `drawArbre()` qui affiche tous les disques de l'arbre.
— il est également possible de ne pas effacer l'écran à chaque appel à la fonction `draw()`, et donc de n'afficher que le nouveau disque qui se dessine en plus de l'image existante.

4 L'arbre en 3D



4.1 Un arbre 3D simple

Analyser une deuxième version de cette construction d'arbre en passant en 3D. Quelques informations pour vous aider :

— le type `PVector` est naturellement en 3D. Le champ `z` est présent et il suffit d'appeler le constructeur `new PVector(a,b,c)` en passant trois valeurs. Les opérations `add(), sub()` et autres travaillent également en 3D.
— la bibliothèque processing `peasy` rend aisée l'exploration tridimensionnelle. Elle peut être téléchargée via le menu "outil/ajouter un outil". Sur les ordinateurs des salles du CIE, il faut l'ajouter manuellement. Un exemple de début de code utilisant *peasy* est donné ci-dessous, n'oubliez pas d'ajouter *P3D* dans l'appel à *size()*.

```
import peasy.PeasyCam;
//----------------var globales----------------
PeasyCam cam;
//----------------  init  ----------------
void setup(){
    size(500,500,P3D);
    cam = new PeasyCam(this, 400);
}
```

— à la place de l'ajout d'un disque, vous utiliserez des triangles quelconques. Voici ci-dessous un exemple de dessin d'un triangle, avec `p` le `PVector` contenant le centre, et `cellRadius`, la taille du triangle, le code suivant placera un triangle en `p`, en le faisant tourner sur lui-même autour des axes x,y et z.

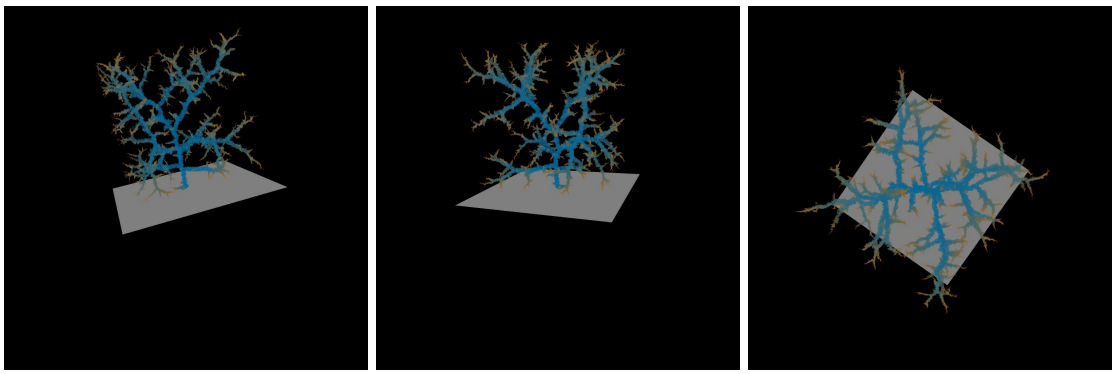
```
pushMatrix();
translate(p.x, p.y, p.z);
rotateX(rx);
rotateY(ry);
rotateZ(rz);
beginShape();
vertex(-cellRadius, -cellRadius, 0);
vertex(-cellRadius, cellRadius, 0.);
vertex(cellRadius, cellRadius, 0.);
endShape();
popMatrix();
```

— des fonctions de rotations sont placées entre `translate(...)` et `beginShape()`, pour orienter le triangle.
— en cas d'orientation aléatoire, ne pas oublier de mettre `randomseed(x)` avec x différent de zéro, au début de la fonction *draw()*, afin d'engendrer à chaque image la même suite de valeurs aléatoires.

4.2 Un arbre 3D moins simple

Les images ci-dessus, montre une coloration différente des triangles. Comme à chaque nouvelle image, un triangle est ajouté à la fin de la liste de positions, l'emplacement dans la liste indique un *âge* de triangle, les derniers étant les plus jeunes. Une interpolation de couleurs basée sur l'emplacement dans la liste permet d'associer une couleur à l'*âge* du triangle.

Les images ci-dessous montrent qu'il est également possible d'associer une taille de triangle à son *âge*. Par exemple un plaçant un appel à la fonction `scale(x)` avant l'instruction `beginShape()`



5 Rendu du travail

Le travail devra être rendu, pour le 21 avril 2023, 23h59, sur la zone dépôt *distanciel* de madoc, partie création numérique. Il contiendra vos deux répertoires de *sketch* sous forme compressée. Vous respecterez les consignes ci-dessous :

— nommage des variables consistant avec leur contenu ;
— nommage des méthodes indiquant ce qui s'y fait ;
— commentaires expliquant les zones de codages et les points critiques ;

Un court texte explicatif (2 ou 3 pages en format pdf) pourra également être rendu.