E-ARTSUP.
DÉPARTEMENT DESIGN INTERACTIF & COMMUNICATION VISUELLE.

ALÉATOIRES
(BRUIT BROWNIEN & PERLIN)

Digital Lab S.04 // Alexandre Rivaux arivaux@gmail.com

Département Design interactif & communication visuelle:

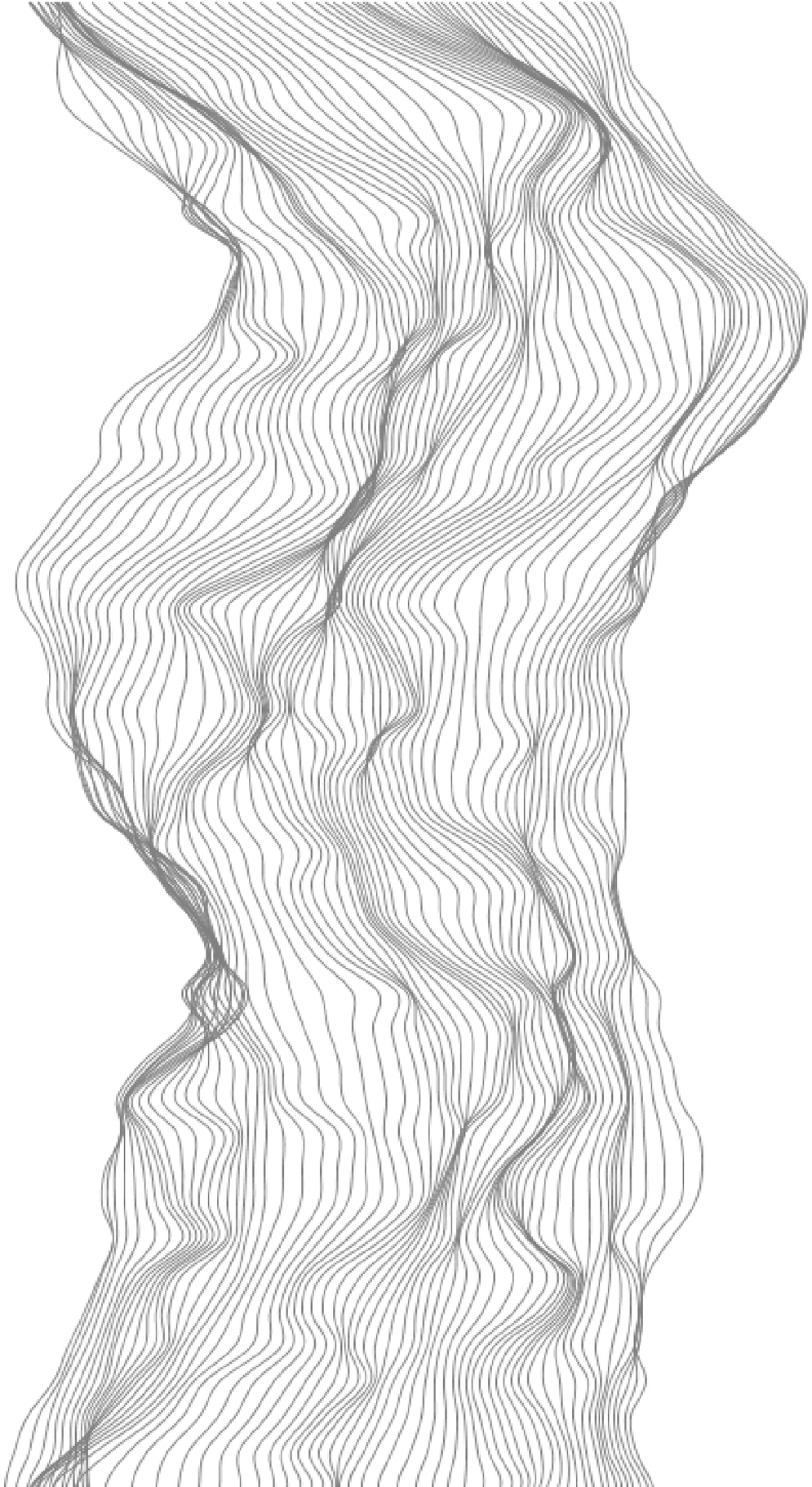
Enseignants:
Nicolas Baumgartner
Félicie d'Estienne D'Orves
Rémi Jamin
Wolf Ka
Jonathan Munn
Gustave Bernier
Alexandre Rivaux

1 - L’aléatoire.

La notion d’aléatoire ou de hasard à une grande place dans le domaine du design generatif et de l’experience interactive. Nous avons régulièrement besoins dans de nombreux cas de générer une ou des valeurs aléatoires. Par exemple, si nous désirons recréer le déplacement chaotique d’un insecte volant, il peut être très pratique de définir sa vitesse ou son déplacement de manière aléatoire. Nous ne serons pas dans une reproduction parfaite de la nature mais nous aurons créé un objet qui aura ses propres propriétés qui ont pour particularité d’être aléatoire.

Si nous pouvons utiliser divers type d’aléatoire dans Processing (brownien, gaussien ou perlin) nous ne verrons ici que les deux principaux types d’aléatoire à savoir le bruit brownien et le bruit perlin.

Lorsque nous souhaitons obtenir une valeur aléatoire en code, il nous faut toujours définir une valeur minimum et un valeur maximum. Imaginons ici que nous souhaitons obtenir une valeur aléatoire comprise entre 0 et 100.



2 - Le bruit Brownien.

Le bruit brownien est un aléatoire où les valeurs produites sont réparties de manière totalement aléatoire. Chaque valeur pourra être grandement espacée les une des autres ou totalement proche voir identique. Une répartition brownien peut être imaginé par ce schema :

Dans processing le bruit brownien est produit par la fonction :

```
random(valeurMin, valeurMax);
```

Il nous suffira de définir les valeurs entre parenthèses pour attribuer une valeur minimum ou maximum. Dans notre cas nous voulions obtenir une valeur comprise entre 0 et 100. Cela nous donne donc :

```
random(0, 100);
```

Appliquons cela dans processing en créant un skecth de 500*500 où nous dessinerons une ellipse de 10*10 à une position aléatoire définie par un random.
Cela nous donne donc :

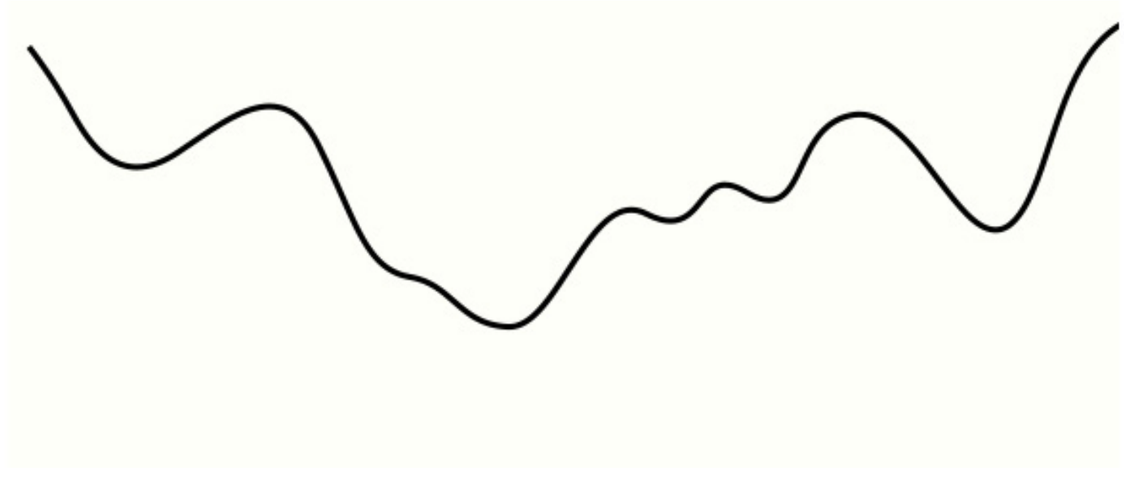
```
float y;
float x;

void setup()
{
  size(500, 500, P2D);
  smooth(8);
}
 void draw()
{
  background(255);
  x= random(0, 500);
  y = random(0, 500);
  ellipse(x, y, 20, 20);
}
```

3 - Le bruit perlin.

Le bruit perlin est bien différent du bruit brownien tant dans sa syntaxe que dans les valeurs qu'il renvoie. Contrairement au bruit brownien le bruit perlin produit une séquence de valeur où chaque valeur est proche de la valeur précédente car défini par ses coordonnées. On obtient donc un répartition harmonieuse de valeur. La structure de bruit généré est visuellement similaire à celle d'un signal audio. Similaire à la notion d'harmoniques en physique, bruit de Perlin est calculée sur plusieurs octaves qui sont additionnés pour le résultat final.

On peut illustrer le bruit perlin par ce schema



Ce bruit, inventé par Ken Perlin en 1980 est très utilisé dans la génération de texture procédural mais aussi dans les déplacement de particule puisqu'il nous permet d'avoir une harmonie entre les valeurs et donc des sensations d'incrémentation et décrémentation plus douces.

Dans sa syntaxe le bruit perlin est bien différent du bruit brownien. Voici comment il se déclare :

```
noise(valeur x);
noise(valeur x, valeur y);
noise(valeur x, valeur y, valeur z);
```

On remarque de prime abord que là où le bruit brownien accueille deux valeurs entre ses parenthèses (minimum et maximum) le bruit perlin peut en accueillir trois. En effet on dis du bruit perlin qu'il peut être uni, bi ou tridimensionnel. Nous nous attarderons sur le bruit perlin uni-dimensionnel dans cette partie.

La seconde remarque que nous ferons sur le bruit perlin est qu'il ne nous renverra toujours qu'une seule valeur comprise entre 0 et 1 et celle-ci sera toujours identique. En effet la répartition de bruit perlin est définie par les coordonnées données donc statique. Il faudra donc incrémenter notre coordonnée puis recalculer notre noise afin d'obtenir une suite de valeurs répartie de façon harmonieuse.
Pour cela nous allons utiliser un valeur que nous appellerons xInc et qui sera égale à 0. Nous calculerons un noise à partir de cette valeur puis à la fin de chaque boucle nous l'incrementerons de 0.01 pour calculer un nouveau noise à partir de la valeur précédente.

3 - Le bruit perlin.

```
float xInc = 0.0;
float x;
x = noise(xInc);
xInc+= 0.01;
```

Nous obtenons donc une suite de valeurs harmonieuses dépendante de la valeur précédente. Cependant nos valeurs restent comprise entre 0 et 1. Pour obtenir nos valeurs entre 0 et 100 il faudra donc effectuer une règle de trois (ou règle de proportionnalité, voir ici) qui nous renverra nos valeurs sur une échelle de 1 à 100.

Pour cela processing nous offre une fonction toute faite : map(). Cette fonction va nous permettre de mapper nos valeurs comprises entre 0 et 1 entre 0 et 100 de la manière suivante :

```
x = map(variable à mapper, valeur Min d'origine, valeur Max,
d'Origine, valeur Min Finale, valeur Max Finale);
```

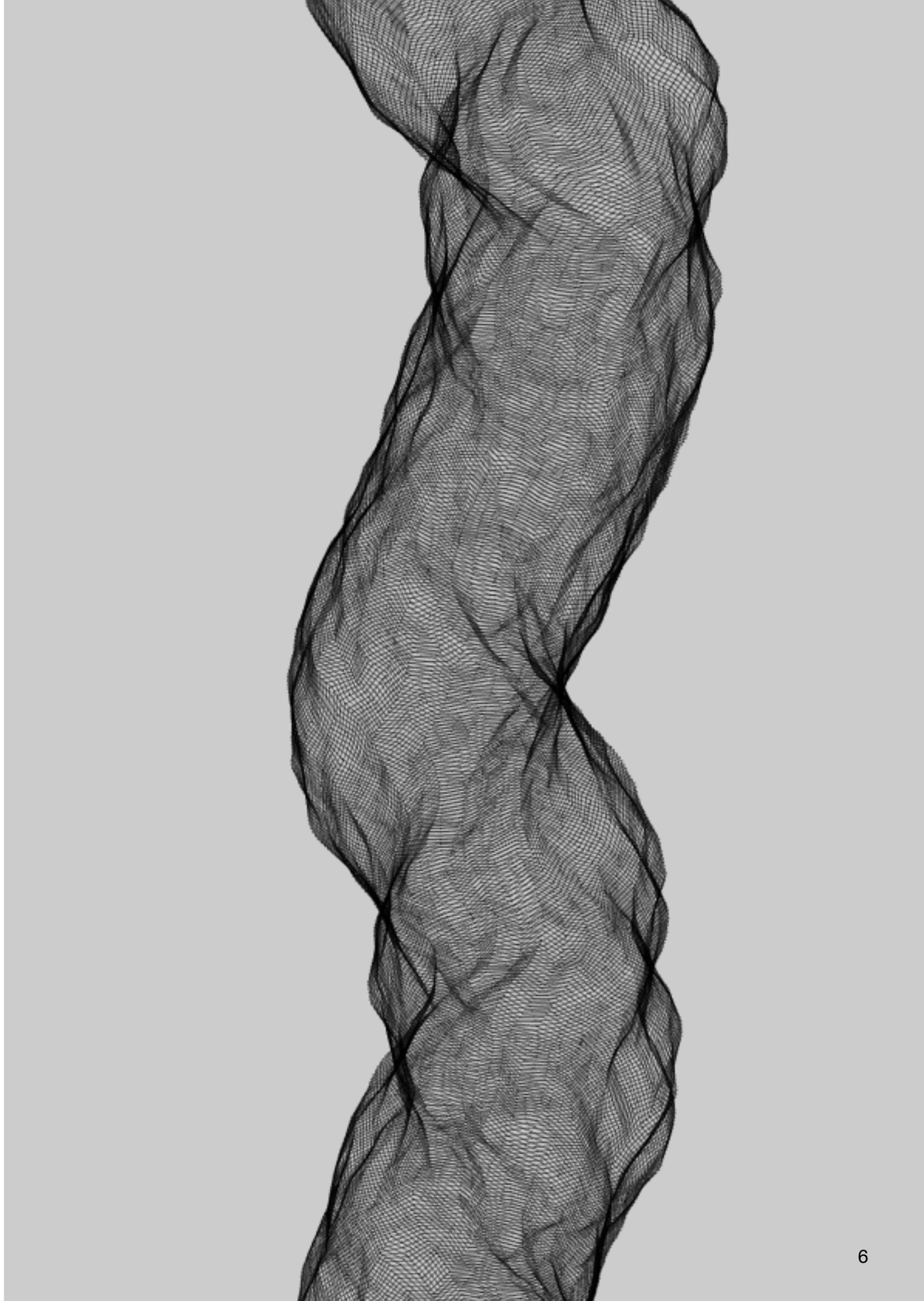
Ainsi pour obtenir nos valeurs en aléatoire perlin compris entre 0 et 100 nous pourrons écrire :

```
float xInc = 0.0;
float x;
x = noise(xInc);
x = map(x, 0, 1, 0, 100);
xInc+= 0.01;
```

Appliquons maintenant cette méthode à notre sketch de 500*500 où nous dessinerons une ellipse de 10*10 à une position aléatoire définie par un noise.

```
float x;
float y;
float xInc = 0.0;
float yInc = 0.0;
void setup()
{
  size(500, 500, P2D);
  smooth(4);
}
void draw()
{
  background(255);
  x= noise(xInc);
  y = noise(yInc);
  x = map(x, 0, 1, 0, 500);
  y = map(y, 0, 1, 0, 500);
  xInc += 0.01;
  yInc += 0.02;
  ellipse(x, y, 20, 20);
}
```

4 - Un exemple de visuel utilisant le bruit brownien et perlin.



5 - Exercice : l’aléatoire

À l’aide de vos connaissances acquises en cours, réaliser une série de 6 visuels basés sur l’utilisation de différent bruits (Perlin et Brownien).

La série devra être évolutive et chaque visuel pourra être reconnu comme étant une évolution du précédent.

La série se présenter sous forme de poster dont la nomenclature sera fourni et devra être respectée.

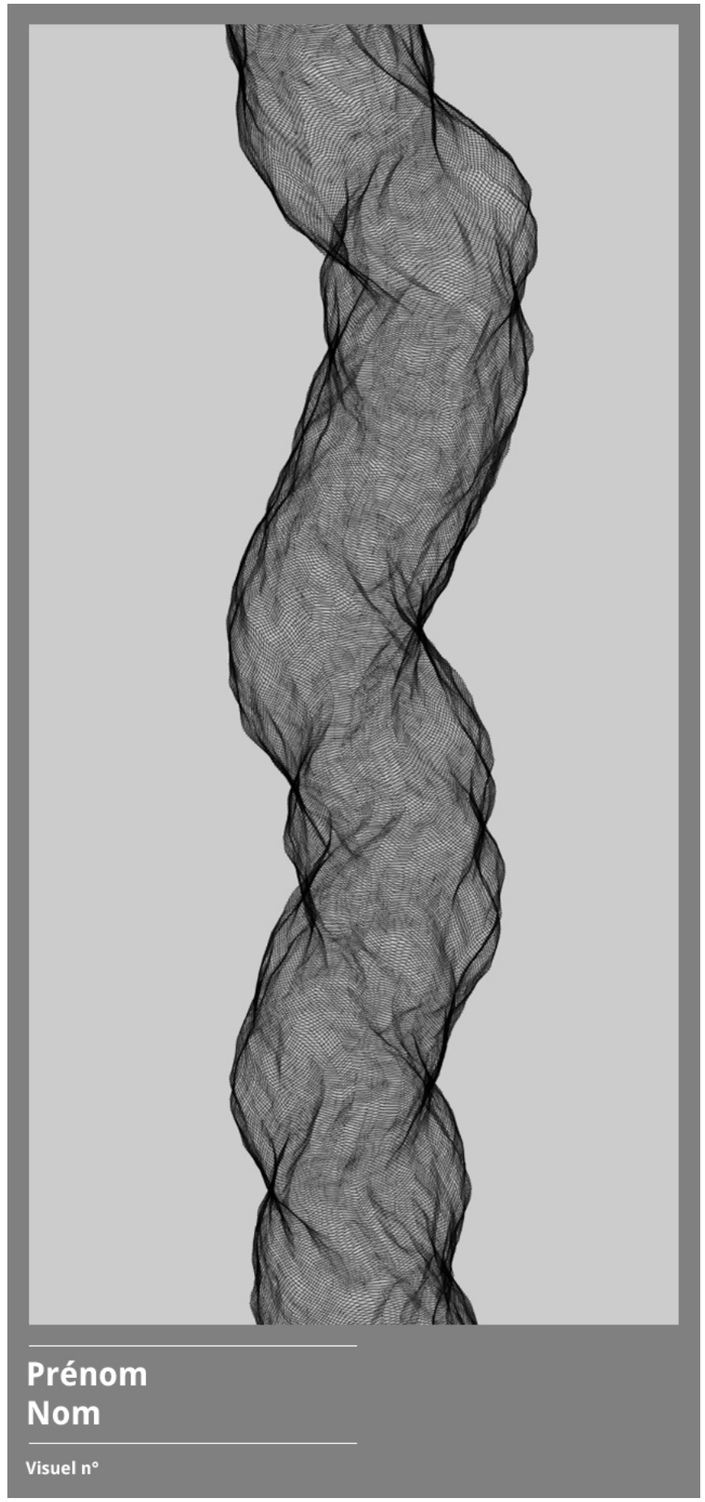
<u>Contraintes :</u>

- **Format du Sketch : 600*1200**
- **Format du poster : 640*1380**

<u>Rendu :</u>

- **Images au format Jpg**
- **Sketch processing**

L’ensemble de fichiers necessaire à la réalisation du sujet sont téléchargeable via GitHub à l’adresse suivante : *https://github.com/alexr4/eartsup-course*



Contact

Alexandre Rivaux
Visual Designer & Partner Bonjour, interactive Lab

www.bonjour-lab.com
arivaux@gmail.com