Chapitre 1

La nuit géométrique

1.1 Notes

Je veux documenter tout le travail fait pour *La nuit géométrique*. Les méthodes utilisées, les idées lancées, etc.

Le film doit n'avoir qu'une seule scène de 12 minutes. Pas de coupes, juste des transitions douces. Donc il me faut une fonction qui gère le temps qui passe, et qui, selon la durée écoulée (la valeur frameCount), assigne les points du graphe à des oscillateurs différents. Je vais appeler mes fonctions, ou mes scènes, ainsi : des oscillateurs.

Un oscillateur peut n'être qu'une simple fonction paramétrique, mais il doit controler lui-même la valeur m. Un oscillateur doit pouvoir prendre les données qui sont présentement dans le graphe g et en faire une interpolation linéaire vers sa propre *forme oscillante*. Un oscillateur doit pouvoir être multiple, c'est-à-dire varier entre 2 équations paramétriques par une interpolation linéaire qu'il contrôle lui-même. Un oscillateur doit pouvoir contrôler l'output vers les fonctions de coloration et effets spéciaux. Un oscillateur doit pouvoir créer des faux *mouvements de caméra*, c'est-à-dire déplacer les sommets du graphe dans n'importe quelle direction à n'importe quelle vitesse.

Clairement, un oscillateur devrait être un prototype d'objet.

```
Oscillator = function (equationX, equationY) {
    this.equationY = equationY;
    this.equationY = equationY;
};

Oscillator.prototype.setPostProcessing = function(postProcessing) {
    this.postProcessing = postProcessing;
};

PostProcessor = function()  {

};

var spiraleEtrange = new oscillator();
```

```
spiraleEtrange.setEquation(function(m) {
t = i/60;
x = sin(t*(-15+m)) * asin(t*4*(m/4)) * 800 * i/1000;
y = sin(t*(15+m)) * asin(t*4*(m/4)) * 550;
});
```

Pour faire une interpolation linéaire entre 2 oscillateurs, il faut que les 2 roulent en même temps.

1.1.1 Autres notes

Pour établir le timing du film, il faudrait que je puisse faire ça :

```
for (50 frames) {
    Run this oscillator;
}
for (150 frames) {
    Run this other oscillator;
}
```

Et si j'utilisais un array pour stocker une liste de durées ?

```
if (drawCount > xSheet.sum(6) && drawCount < xSheet.sum(6)+xSheet[6]){
}
if (drawCount > xSheet.sum(7) && drawCount < xSheet.sum(7)+xSheet[7]){
}
if (drawCount > xSheet.sum(8) && drawCount < xSheet.sum(8)+xSheet[8]){
}
if (drawCount > xSheet.sum(9) && drawCount < xSheet.sum(9)+xSheet[9]){
}
```

J'aurais un array dont chaque cellule contiendrait une durée en frameCount, et j'ajouterais une fonction au prototype array qui me donnerait, pour n'importe quelle cellule, la somme des cellules qui la précèdent.

Ou un objet ? Non.

```
var xSheet = [50,100,300,450,200,250,300,800,900,250,300];
var xSheet = {
    1 : 100,
    2 : 150,
    3 : 300,
    4 : 50,
    5 : 800,
    7 : 50,
```

```
    8 : 350,
    9 : 200,
    10 : 400,
    11 : 100,
    12 : 130
};

var xSheet = {
    1 : {
        d :  500,
        f :  function(){
                nouvelleDimension.update();
                simple.output(nouvelleDimension,1);
            }
        },
    2 : {
        d :  250,
        f :  function(){
                nouvelleDimension.update();
                simple.output(nouvelleDimension,1);
            }
        },
    key : function(n) {
        return this[Object.keys(this)[n]];
    }
};
```

1.2 Mathématiques pour la simulation d'une troisième dimension

Les points sont générés avec un *for loop* qui incrémente une valeur i. Les valeurs sont multipliées par un quotient de la valeur i, par exemple $\frac{i}{1000}$. Donc il est normal que les points deviennent de plus en plus éloignés à mesure que le graphe se construit, puisque la valeur i augmente.

Ensuite, j'inclus une valeur m dans mon équation, définie ainsi :

```
dC = sin(frameCount/20)/100;
```

Voici la première équation paramétrique avec laquelle j'ai utilisé la valeur m :

$$t = \frac{i}{10}$$
$$x = \cos(t) \times \sin\left(\frac{t}{2}\right) \times \left(\frac{\sin\left(t \times (2+m)\right)}{4}\right) \times 2400 \times \frac{i}{1000}$$
$$y = \sin^3\left(\frac{t}{2}\right) \times \cos\left(\frac{t}{20}\right) \times 350$$

```
t = i/10;
x = cos(t) * sin(t/2) * (sin(t*(2+m))/4) * 2400 * i/1000;
y = pow(sin(t/2),3) * cos(t/20) * 350;
```

1.3 Algorithmes de création de sommets

J'ai déjà développé des dizaines d'algorithmes différents pour créer mes graphes de façons diverses.

1.3.1 Méthode de déclanchement des algorithmes

Tout d'abord, dans ma fonction draw, je dessine un background pour effacer le frame précédent, puis je vide l'array vertices. Je crée ensuite la valeur dC (drawCount), qui est pour l'instant plutôt mal définie. Et il en existe 2 versions. La première utilisée est maintenant mise en commentaire. Ensuite, je lance la fonction createVertices() avec la variable dC comme paramètre. L'idée de base était d'introduire une variation à chaque lancement de createVertices, puisqu'ensuite, createVertices fait toujours la même chose, c'est une suite finie et non-ambigüe d'instructions (donc un algorithme).

Ensuite, une fois que le graphe vertices est rempli, je lance la fonction drawVerticesSimple(), qui pour l'instant ne fait que looper dans l'array et dessiner une ellipse blanche pour chaque sommet.

```
function draw() {
  background(0);
  vertices = [];
  // dC = sin(frameCount/20)/100;
  dC = sin(frameCount/700)/5;
  createVertices(dC);
  drawVerticesSimple();
}
```

1.3.2 La fonction drawVertices()

Cette fonction est très simple. Elle a un paramètre m, et a besoin d'une équation paramétrique qui lui donne des valeurs x et y afin de créer des vecteurs $\vec{v}$. Elle ajoute ensuite ces vecteurs dans

l'array *vertices*.

```
function createVertices(m) {
    for (var i = 0; i < 3000; i++) {
        //Insérez une équation paramétrique ici.
        var v = createVector(x, y);
        vertices.push(v);
    }
}
```

1.3.3 Équations paramétriques

Voici la liste de toutes les équations dont je me sert pour créer *La nuit géométrique*. Le nombre d'itérations est indiqué à titre de simple suggestion.

Formule magique

C'est ma toute première équation qui utilise la valeur m.

```
t = i/10;
x = cos(t) * sin(t/2) * (sin(t*(2+m))/4) * 2400 * i/1000;
y = pow(sin(t/2),3) * cos(t/20) * 350;
```

Formule magique modifiée

```
t = i/10;
x = sin(t*(2+m)) * 400 * i/1000;
y = sin(t*(-2+m)) * 350;
```

Spirale en toupie horizontale I

Cette formule a d'abord été créée avec une valeur i maximale de 1000, et un valeur m générée ainsi :

```
dC = sin(frameCount/20)/100;
```

$$
\begin{aligned}
t &= \frac{i}{10} \\
x &= \sin\big(t \times (2+m)\big) \times 400 \times \frac{i}{1000} \\
y &= \cos\big(t \times (2+m)\big) \times \sin\big(t \times (4 \times m)\big) \times 350
\end{aligned}
$$

```
t = i/10;
x = sin(t*(2+m)) * 400 * i/1000;
y = cos(t*(2+m)) * sin(t*(4*m)) * 350;
```

Spirale extra-magique 2

1000 itérations.

```
t = i/10;
x = sin(t*(2+m)) * sin(t*(2*m)) * 400 * i/1000;
y = cos(t*(2+m)) * sin(t*(2*m)) * 350;
```

Juste débile

```
t = i/10;
x = sin(t*(2+m)) * sin(t*(m/2)) * 800 * i/1000;
y = cos(t*(2+m)) * sin(t*(2*m)) * 350;
```

Formule modifiée - Intéressant !

```
t = i/10;
x = sin(t*(2+m)) * cos(t*(4*m)) * 800 * i/1000;
y = cos(t*(2+m)) * cos(t*(2*m)) * 350;
```

Spirale en toupies entrelacées

```

```

```
dC = sin(frameCount/700)/5;
```

$$t = \frac{i}{10}$$
$$x = \sin\left(t \times (2+m)\right) \times \cos\left(t \times \frac{m}{4}\right) \times 800 \times \frac{i}{1000}$$
$$y = \cos\left(t \times (2+m)\right) \times \cos\left(t \times \frac{m}{4}\right) \times 350$$

```
t = i/10;
x = sin(t*(2+m)) * cos(t*(m/4)) * 800 * i/1000;
y = cos(t*(2+m)) * cos(t*(m/4)) * 350;
```

Spirale envoûtante

```
t = i/10;
x = sin(t*(2+m)) * asin(t*(m/4)) * 800 * i/1000;
y = cos(t*(2+m)) * asin(t*(m/4)) * 350;
```

1.3.4 Les équations paramétriques avec oscillateur

Je me sert d'une fonction oscillante pour modifier certains de mes graphes. Mon oscillation est générée ainsi : je me crée une valeur oscillante entre 0 et 1 à partir de la fonction sinus du frameCount, que je "normalise" avec la fonction map. Ensuite, je fais une interpolation linéaire entre deux équations paramétriques, en utilisant mon oscillateur comme valeur d'interpolation. J'obtiens ainsi un graphe qui oscille entre deux équations paramétriques. Chaque sommet du graphe semble déchiré entre 2 objectifs, 2 forces différentes qui le poussent, et l'oscillation est douce puisque j'utilise la fonction sinus.

J'ai conçu mentalement cette notion d'oscillateur un peu à la manière d'un vocoder, c'est-à-dire qu'un vocoder a besoin de deux choses : un modulateur (la voix humaine) et un *carrier signal* ou *onde porteuse*. Dans mon esprit, la première équation paramétrique est le modulateur, et la deuxième équation est l'onde porteuse. Mais au fond, les deux équations sont tout à fait interchangeables et c'est la fonction d'interpolation linéaire qui détermine l'*expression* de chacune d'elles.

Comme on peut le voir, je me sers des valeurs tt, xx et yy pour les valeurs de l'équation no. 2, et ensuite je fait une interpolation linéaire entre x et xx, puis entre y et yy.

```
//MODULATOR
```

```
//OscNorm max  : 0.05;
// t = i/2;
// x = cos(t+(m*10)) * 350 * t/1000;
// y = sin(t+(m*40)) * 350;

//OscNorm max  : 1;
t = i/10;
x = sin(t*(2+m)) * asin(t*(m/20)) * 800 * i/1000;
y = cos(t*(2+m)) * asin(t*(m/20)) * 350;

//CARRIER SIGNAL
//Spirale ultra—magique.
tt = i/10;
xx = sin(t*(2+m)) * 400 * i/1000;
yy = cos(t*(2+m)) * sin(t*(4*m)) * 350;

// tt = i/10;
// xx = sin(t*(2+m)) * asin(t*(m/20)) * 800 * i/1000;
// yy = cos(t*(2+m)) * asin(t*(m/20)) * 350;

//OSCILLATOR FUNCTION
oscillator = sin(frameCount/20);
oscNorm = map(oscillator, —1, 1, 0, 1);
morphX = lerp(x, xx, oscNorm);
morphY = lerp(y, yy, oscNorm);
x = morphX;
y = morphY;
```

1.4 Coloration et effets spéciaux

Je vais devoir explorer différents concepts de coloration et d'effets spéciaux.