TD Bruit de Perlin

Tout d'abord, les fonctions de bruit (comme par exemple le bruit de Perlin) sont des fonctions continues définies sur $\mathbb{R}^n \mapsto \mathbb{R}$ avec $n \in \mathbb{N}^*$ qui semble chaotiques. Nous nous intéresserons dans ce TD au bruit de Perlin (utilisé dans la création d'effets spéciaux ou de génération de terrain tel que des terrains vagues... ou simplement des vagues).
Ce bruit a l'avantage de ne pas être breveté, ce qui le rend libre d'usage (du moins à des usages commerciaux...), ce qui n'est pas le cas de toutes les fonctions de bruit.

FIGURE 1 – Des bruits de Perlin (celui de droite est en réalité une somme de deux bruits)

I Principe de fonctionnement du bruit de Perlin

Le principe de cet algorithme est d'affecter un vecteur unitaire à chaque point de coordonnées entières de l'espace (les vecteurs sont unitaires pour que les valeurs de sorties soient comprises entre -1 et 1). Ainsi, pour obtenir l'image d'un point par cette fonction, il suffit d'interpoler le point avec quelques valeurs de référence proche de ce point. Le but de cet algorithme est d'être rapide et efficace (il ne va pas créer tous les vecteurs du plan ... il y en a une infinité. Il ne va pas non plus calculer la distance entre chaque vecteur pour ne garder que les plus proches, encore une fois, il y en a une infinité...).

Pour déterminer alors la valeur particulière de la fonction en un point, on calcule la valeur en quelques points particuliers, puis en faisant une moyenne pondérée nous pouvons déterminer la valeur de notre point particulier.

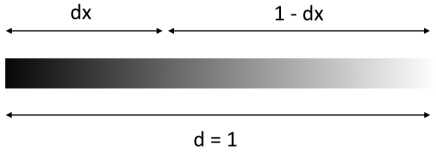

FIGURE 2 – Connaissant la couleur aux extrémités, nous pouvons déterminer toutes les valeurs des points du segment avec la relation $x = v1 * dx + (1 - dx) * v1$ où v0 est la valeur de l'extrémité gauche du segment et v1 celle de droite.

II Cas particulier du bruit de Perlin dans $\mathbb{R}^2$

Les vecteurs qui seront associés au coordonnées entières seront tous unitaires. Pour des questions d'efficacité, nous ne considérerons que 8 vecteurs (ceux suivant les axes x et y ainsi que les quatre diagonales).

1. Créer une fonction `element_aleatoire` qui prend en argument une liste, choisit un élément de celle-ci de manière aléatoire (on pourra utiliser la fonction `randint` du module `random`), l'enlève de la liste et renvoie cet élément.
2. Écrire une fonction `liste_aleatoire` prenant en argument un entier `n` et renvoyant une liste contenant les entiers $k \in [\![0\ ;\ n[\![$ dans un ordre aléatoire. (attention, chaque entier doit apparaître une et une seule fois).
   Par la suite on considérera des listes aléatoires d'une taille de 256.
3. Créer une fonction ne prenant pas d'argument et renvoyant la liste des 8 vecteurs unitaires décris ci-dessus (les vecteurs seront représentés par des couples (x, y)).
4. Créer alors une fonction `obtenir_vecteur` prenant en argument deux entiers `x` et `y` ainsi qu'une liste de vecteurs et une liste d'entiers et va renvoyer le $k^{\text{ème}}$ vecteur de la liste suivant la relation :
   $k = entiers[(y + entiers[x\%n])\%n]\%m$
   avec `n` la taille de la liste d'entiers, `m` le nombre de vecteurs et `entiers` la liste d'entiers. (Ce qui permet d'avoir deux vecteurs différents pour deux points différents, mais toujours le même vecteur pour le même point).
5. Écrire alors une fonction `scalaire` prenant en argument deux vecteurs et calculant leur produit scalaire. Ainsi qu'une fonction `vecteur` prenant en argument `x1`, `y1`, `x2` et `y2` des réels et renvoyant le vecteur d'origine (x1, y1) et "finissant" en (x2, y2).

II.1 Calcul de la valeur de sortie de l'algorithme

Il ne reste plus alors qu'à interpoler le point (x, y) donné en argument avec les quatre vecteurs les plus proches. Les questions suivantes seront toutes écrites dans une même fonction que l'on appellera très originalement `generer_bruit_de_perlin` prenant en argument une constante `resolution` et qui renverra une fonction `bruit_de_perlin` .

6. Créer alors une fonction `generer_bruit_de_perlin` qui va définir des variables locales :
   — `vecteurs` une liste de vecteurs (contenant les 8 vecteurs unitaires)
   — `entiers` une liste contenant les nombres inferieurs à 256 (mélangés)
   — `resolution` la constante `resolution` donné en argument

   Créons maintenant, dans cet environnement local, la fonction `bruit_de_perlin` qui va prendre en argument deux flottants `x` et `y` et va les diviser par la constante résolution (toutes les lignes de code suivantes seront écrites dans cette dernière fonction).

7. Tout d'abord, calculons les coefficients d'interpolations `dx` et `dy` définis par :
   $dx = x - \lfloor x \rfloor$ et $dy = y - \lfloor y \rfloor$. Ces coefficients vont permettre de pondérer les quatre vecteurs afin que le plus proche des quatre ait un impacte plus important sur le résultat. (Ici les coefficients sont calculés avec le polynôme $X(x - \lfloor x \rfloor)$, mais il est à noté que ses calculs peuvent être remplacés par n'importe quel polynôme de $[0, 1] \mapsto [0, 1]$ par

exemple le polynôme $(3X^2 - 2X^3)(x - \lfloor x \rfloor)$ (ce qui donne des transitions différentes entre les différentes cases du plan).

8. On considère maintenant le vecteur `u` associé au point (x0, y0) avec $xo = \lfloor x \rfloor$ et $y0 = \lfloor y \rfloor$ (le vecteur donné par la fonction `obtenir_vecteur` en (x0, y0)). Calculer le vecteur `v` entre les points (x, y) et (x0, y0) puis calculer le produit scalaire `s_x0y0` entre les vecteurs `u` et `v`.
9. De même calculer les produits scalaires `s_x0y1`, `s_x1y0` et `s_x1y1` avec $x1 = 1 + x0$ et $y1 = 1 + y0$ (en considérant les vecteurs entre (x, y) et (x0, y1) puis (x1, y0) et (x1, y1)).
10. Il ne reste plus qu'à appliquer l'interpolation finale. Pour cela, on calcul `l1` et `l2` définis par :
$l1 = dx * s\_x1y0 + (1 - dx) * s\_x0y0$ et $l2 = dx * s\_x1y1 + (1 - dx) * s\_x0y1$
Ensuite, on calcule la valeur `l` de la fonction de bruit en (x, y) par :
$l = dy * l2 + (1 - dy) * l1$
renvoyer alors cette dernière valeur et voilà, la fonction est terminée ! (n'oublier pas de renvoyer la fonction `bruit_de_perlin` à la fin de la fonction `generer_bruit_de_perlin`).

Pour d'autres explications de cet algorithme en dimension 2, vous pouvez aller voir le site :

http ://sdz.tdct.org/sdz/bruit-de-perlin.html

III Amélioration

Maintenant que ce programme fonctionne correctement, nous pouvons le perfectionner. On constate que ce bruit est périodique, en effet cette fonction renvoie la même valeur pour (x, y) et (x, y+256).
De plus, cette fonction renvoie toujours 0 si les coordonnées du point entrées en argument sont un multiple de `resolution`.
Pour toutes ses raisons, il est intéressant de rajouter quelques caractéristiques à notre bruit.

11. Pour régler tous ses problèmes, il suffit de constater que, la somme de deux fonctions periodiques n'est pas périodique (sous mesure de certaines conditions sur leur périodes respectives). Écrire alors une fonction `generer_bruit_de_perlin_2` qui renvoie une fonction de bruit faisant la moyenne de deux bruits de Perlin de période différente. (La période est modifiée grâce au paramètre `resolution`).
12. En allant encore plus loin, dans certaines situations, il est intéressant d'avoir la somme d'un bruit doux "dominant" avec un bruit brutal "atténué" (ou plus que 2 fonctions...) (Un exemple est présenté au début de l'énoncé) permettant ainsi une plus grande variété de fonction de bruit. Écrire donc une fonction `generer_bruit_de_perlin_3` prenant en argument une liste `layers` qui sera un tableau composé des couples (résolution, coefficient), correspondants aux différentes résolutions que l'on va appliquer à nos différents bruits de Perlin ainsi que le coefficient qui leurs sera affecté. La valeur de retour de cette fonction sera naturellement la moyenne pondérée des bruits de Perlin.

IV Bruit de Perlin dans $\mathbb{R}^n$

Dans certains cas, il est intéressant d'avoir une fonction de bruits à plus de 2 dimensions (par exemple, pour créer un bruit à 2 dimensions animé, le temps serait alors la 3ème dimension). Le principe n'est pas plus compliqué, il faut juste considérer des vecteurs à n dimension (que l'on représentera naturellement sous la forme d'une liste de n flottant).

13. Pour (re)commencer, écrire une fonction `increment` qui prend en argument un tableau `nombre` représentant un nombre en base b (nous travaillerons avec les bases 2 et 3) ainsi que la base `b` et qui incrémente ce nombre dans cette base.
14. Créer une fonction, prenant en argument la dimension dans laquelle vous travaillez et renvoyant la liste des $3^n - 1$ vecteurs unitaires.
    Les vecteurs ainsi créés sont tous les vecteurs de la forme $(\underbrace{0, \pm x_k, 0, ..., \pm x_k}_{n})$avec k le nombre de coordonnées non nul, n la dimension et $x_k = \frac{\sqrt{k}}{k}$ (naturellement, il y a alors $n - k$ coordonnées nul).
    Pour dénombrer ces vecteurs, il suffit de voir qu'il y a une bijection évidente avec l'écriture des nombres de 1 à $3^n - 1$ en base 3.
15. Réécrivez alors les fonctions du cas particulier d'un bruit à 2 dimensions, mais en n dimensions :
    — on veillera à avoir une liste d'entiers d'une capacité d'au moins le carré du nombre de vecteurs unitaires
    — on ne fera plus 2 interpolations, mais n (une dans chaque dimension)
    — pour obtenir un vecteur aléatoire, on utilisera la relation de récurrence : $k_i = x_i + entiers[k_{i-1}])\%n$ avec $k_0 = 0$, n la taille de la liste d'entiers et x_i la $i^{\text{ème}}$ coordonnée dans l'espace du point dont on veux obtenir un vecteur aléatoire. Le vecteur ainsi sélectionné sera donc $vecteurs[k_n\%m]$ avec m le nombre de vecteurs et `vecteurs` la liste des vecteurs.

On notera `p0 = [int(x) for x in p]` et `p1 = [x + 1 for x in p0]` avec p le point dont on veut connaître la valeur par la fonction de bruit.
Pour calculer les différents produits scalaires, encore une fois il faut remarquer une bijection entre l'écriture en base 2 et les coordonnées des vecteurs avec lesquelles on va faire des produits scalaires (la $k^{\text{ème}}$ coordonnée du vecteur avec lequel on calcul les scalaires est soit la $k^{\text{ème}}$ coordonnée de p0 soit celle de p1 (bijection plus qu'évidente entre les symboles 1 et 0 avec les vecteurs p0 et p1))