

Fractales

[Vidéo ■ partie 17.1. Montagnes](#)

[Vidéo ■ partie 17.2. IFS](#)

[Vidéo ■ partie 17.3. L-systèmes \(2D\)](#)

[Vidéo ■ partie 17.4. L-systèmes \(3D\)](#)

Les fractales sont des formes géométriques auto-similaires : lorsque l'on zoome sur une partie, on retrouve une image ressemblant à la figure globale. Les structures fractales permettent de dessiner des paysages et de la végétation. La méthode est facile à implémenter, permet de générer aléatoirement une grande variété de structures, utilise très peu de données, mais par contre nécessite des calculs.

Dans ce chapitre on se contente d'une modélisation assez grossière : sans couleur ni texture.

1. Montagnes

Dessinons un paysage avec des montagnes et des vallées. La génération est rapide, aléatoire et paramétrable. Afin de mieux comprendre l'algorithme général, on commence avec une seule dimension par la génération du profil d'une montagne, ou bien d'un cours de bourse.

1.1. Une dimension

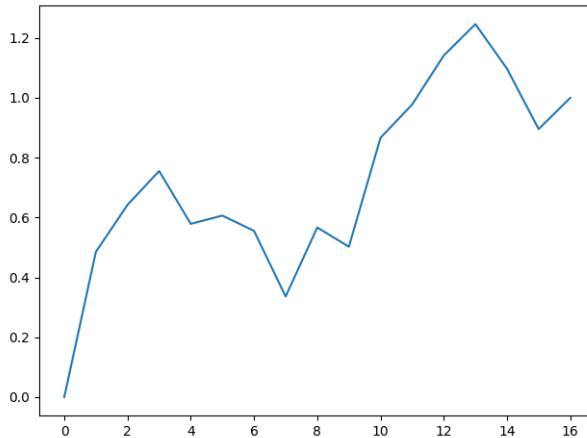


Données. Voici les données qui permettent de générer un profil de montagne :

- une altitude de départ H_a ,
- une altitude d'arrivée H_b ,
- une valeur d'amplitude maximale h_0 ,
- un coefficient de rugosité r ,
- un nombre de subdivisions $N = 2^n$.

On peut aussi fixer le germe (*seed*) du processus de génération pseudo-aléatoire afin de pouvoir reproduire les mêmes résultats lors d'une génération future.

Sortie. L'algorithme renvoie $N + 1$ valeurs (la première étant H_a et la dernière H_b).



Un exemple avec $n = 5$, $N = 16$, $H_a = 0$, $H_b = 1$, $h_0 = 1$ et $r = 0.7$.

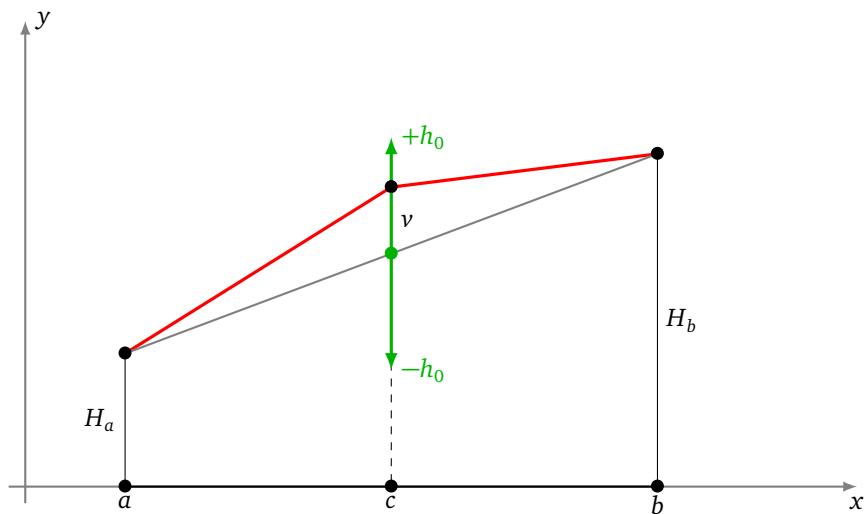
Algorithme.

On considère qu'on trace le profil au-dessus d'un segment $[a, b]$. L'algorithme est un calcul de moyenne des altitudes avec ajout d'un saut aléatoire. Il s'effectue par dichotomies successives de l'intervalle de départ.

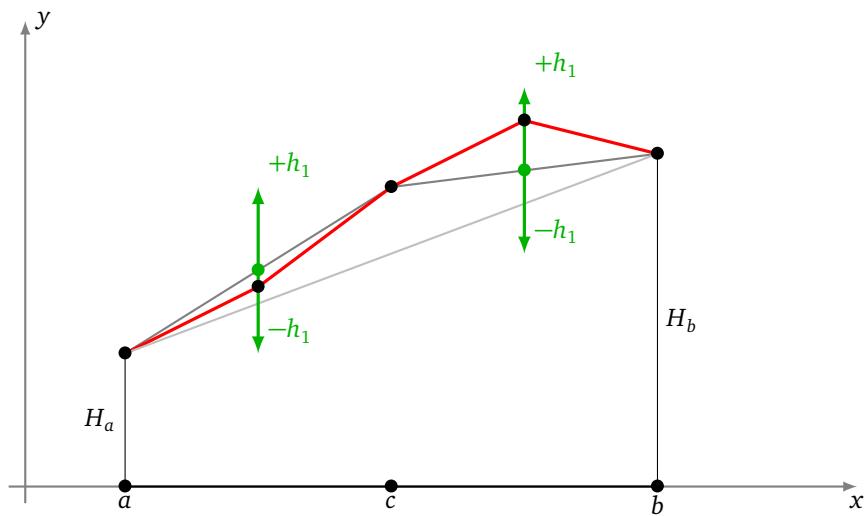
- *Étape 1.* En $c = \frac{a+b}{2}$, qui est le milieu de $[a, b]$, on calcule la moyenne des hauteurs de départ et d'arrivée à laquelle on ajoute une valeur aléatoire :

$$H_c = \frac{H_a + H_b}{2} + v.$$

La valeur v est tirée aléatoirement dans l'intervalle $[-h_0, +h_0]$. On obtient donc 3 ($= 2^1 + 1$) valeurs H_a, H_c, H_b .



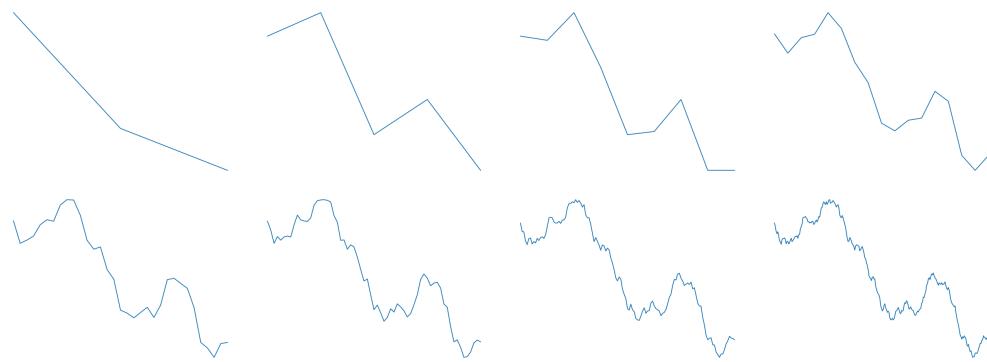
- *Étape 2.* On recommence entre a et c en calculant la moyenne entre H_a et H_c à laquelle on ajoute une valeur aléatoire de $[-h_1, +h_1]$ où $h_1 = \frac{h_0}{2^r}$. On fait de même entre c et b , avec la moyenne entre H_c et H_b et une nouvelle valeur aléatoire de $[-h_1, +h_1]$. On obtient 5 ($= 2^2 + 1$) altitudes.



- ...
- Étape i . On obtient $2^i + 1$ altitudes. La hauteur aléatoire est choisie dans $[-h_i, +h_i]$ où $h_i = \frac{h_0}{2^{ir}}$.
- Étape n . On obtient $2^n + 1$ altitudes.

Exemple.

Voici le déroulé de la construction étape par étape.



Les étapes de la construction avec n variant de 1 à 8.

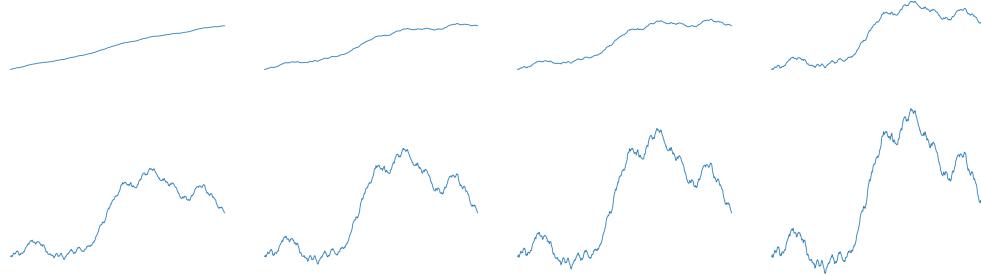
Exemple.

Voici les profils obtenus pour différents tirages aléatoires. Les valeurs H_a , H_b , h_0 et r sont les mêmes pour tous ces profils.



**Exemple.**

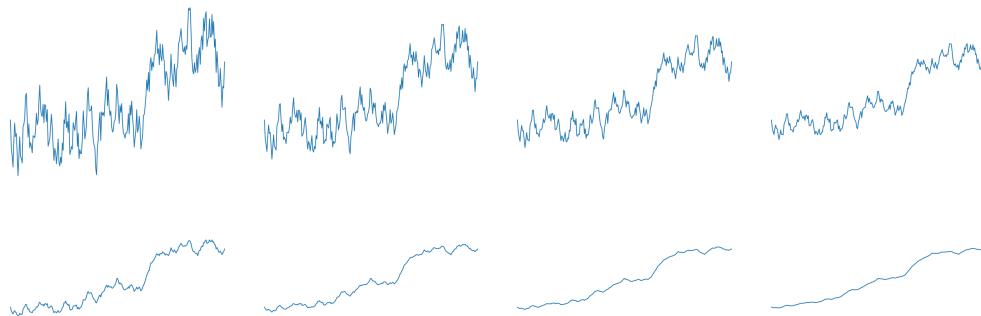
Voici les profils obtenus pour différentes valeurs de l'amplitude h_0 . Les autres valeurs (y compris le germe du tirage pseudo-aléatoire) sont les mêmes.



L'amplitude de variation aléatoire h_0 varie : 0.1, 0.3, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0.

Exemple.

Voici les profils obtenus pour différentes valeurs de la rugosité r . Les autres valeurs (y compris le germe du tirage pseudo-aléatoire) sont les mêmes. Plus le coefficient r est grand, plus la courbe est lisse.



Le coefficient de rugosité r varie : 0.2, 0.3, 0.4, 0.5, 0.7, 0.9, 1.1, 1.3.

Exercice.

Programmer cet algorithme sous la forme itérative comme expliqué ci-dessus. Programmer aussi une version récursive.

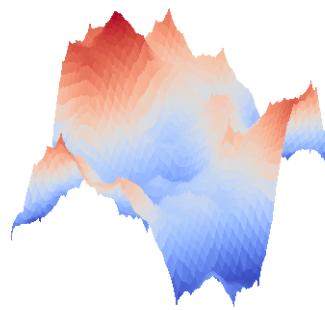
1.2. Deux dimensions

Tracer une surface dans l'espace qui représente un paysage est évidemment la situation qui nous intéresse le plus. Il s'agit d'adapter l'algorithme précédent avec une dimension supplémentaire afin de générer les altitudes d'un terrain.

Données. Voici les données qui permettent de générer un profil de montagne au-dessus d'une zone carrée :

- une altitude de départ H qui sera l'altitude aux quatre coins du carré,
- une valeur d'amplitude maximale h_0 ,
- un coefficient de rugosité r ,
- un nombre de subdivisions $N = 2^n$.

Sortie. L'algorithme renvoie $(N + 1)^2$ valeurs, correspondant aux altitudes au-dessus d'une grille carrée $(N + 1) \times (N + 1)$.



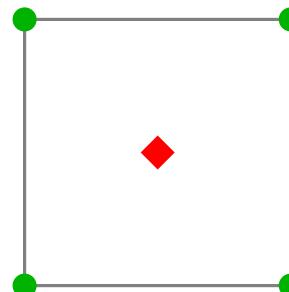
Algorithme.

La construction se fait en itérant une construction « diamant-carré ».

- *Étape diamant.* À partir de 4 valeurs connues aux sommets d'un carré, on détermine une valeur au centre selon la formule :

$$\text{moyenne des 4 valeurs} + \text{valeur aléatoire.}$$

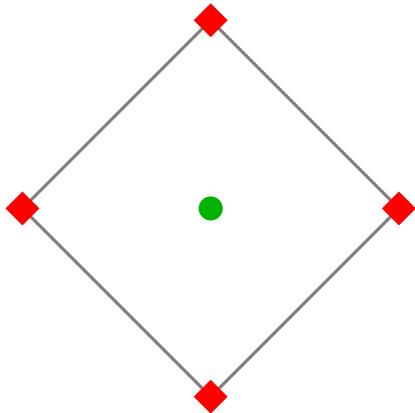
À l'étape numéro i la valeur aléatoire est dans l'intervalle $[-h_i, +h_i]$ où $h_i = \frac{h_0}{2^{ir}}$.



- *Étape carré.* À partir de 4 valeurs connues aux sommets d'un losange, on détermine une valeur au centre selon la formule :

$$\text{moyenne des 4 valeurs} + \text{valeur aléatoire.}$$

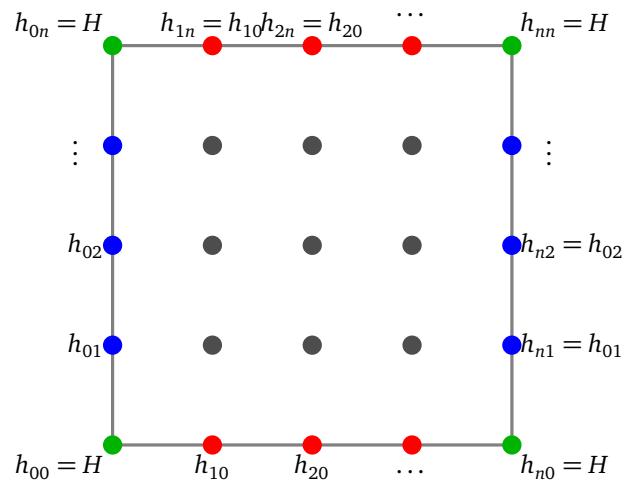
À l'étape numéro i la valeur aléatoire est de nouveau choisie dans l'intervalle $[-h_i, +h_i]$.



L'étape carré est en fait une étape diamant composée avec une rotation de 45° .

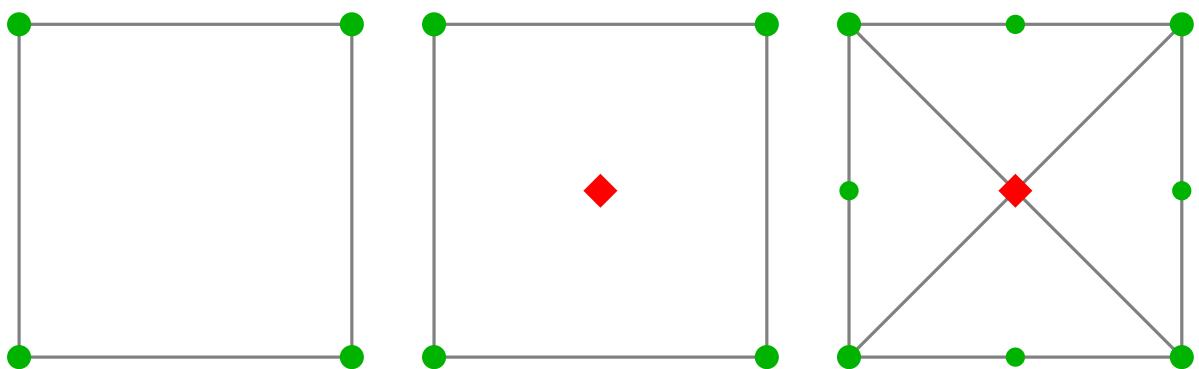
Il y a un problème avec l'étape carré car, aux bords de la zone carrée initiale, on ne dispose pas d'un losange complet et donc on n'a pas toujours 4 valeurs à moyenner. Plusieurs solutions sont possibles :

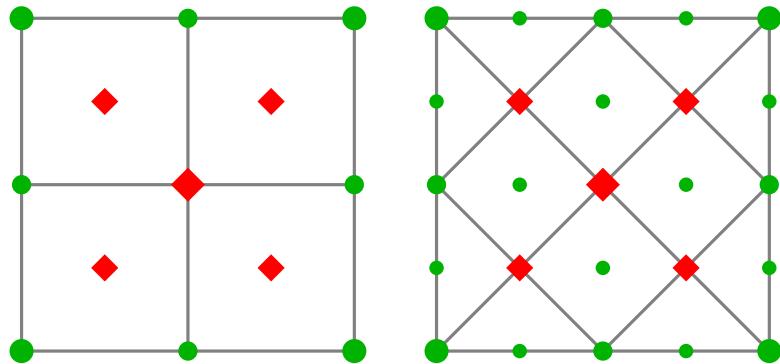
- ne tenir compte que des sommets présents en faisant la moyenne sur 3 sommets au lieu de 4 ;
- considérer que les valeurs sont périodiques : les valeurs du côté gauche du tableau initial sont les mêmes que les valeurs du côté droit, et les valeurs du bas sont les mêmes que les valeurs du haut.



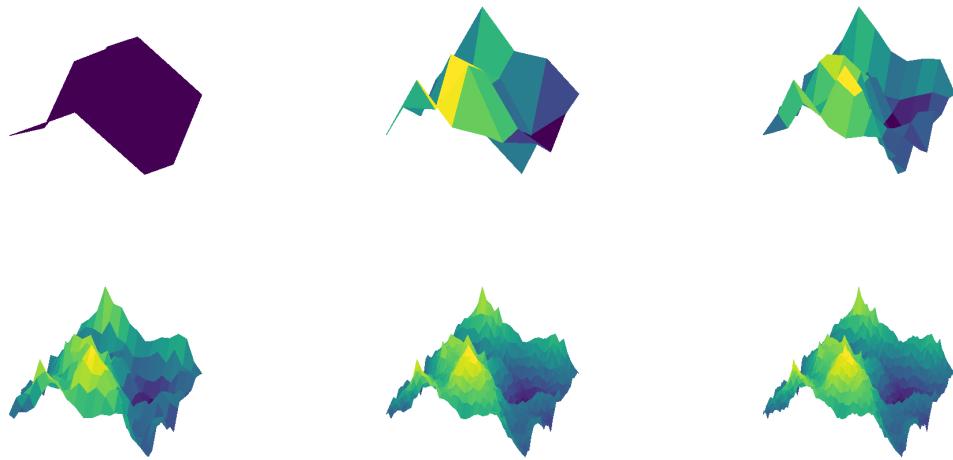
Nous adoptons la seconde solution qui a l'avantage de créer une surface avec le bord gauche identique au bord droit et le bord haut identique à celui du bas et donne ainsi un motif de base qui se recolle parfaitement lorsque qu'on en juxtapose plusieurs.

Voici les points où les altitudes sont calculées lors des premières étapes de la construction « diamant-carré ».



**Exemple.**

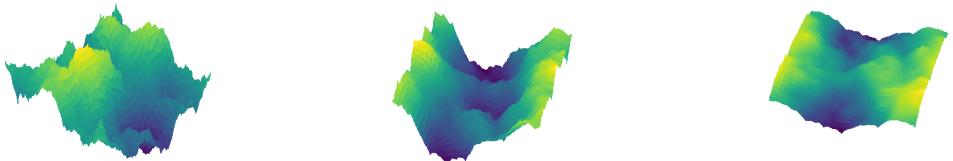
Voici un exemple de cette construction étape par étape.



Les étapes de la construction avec n variant de 1 à 6.

Exemple.

Voici les profils obtenus pour différents tirages aléatoires et différentes valeurs des paramètres.

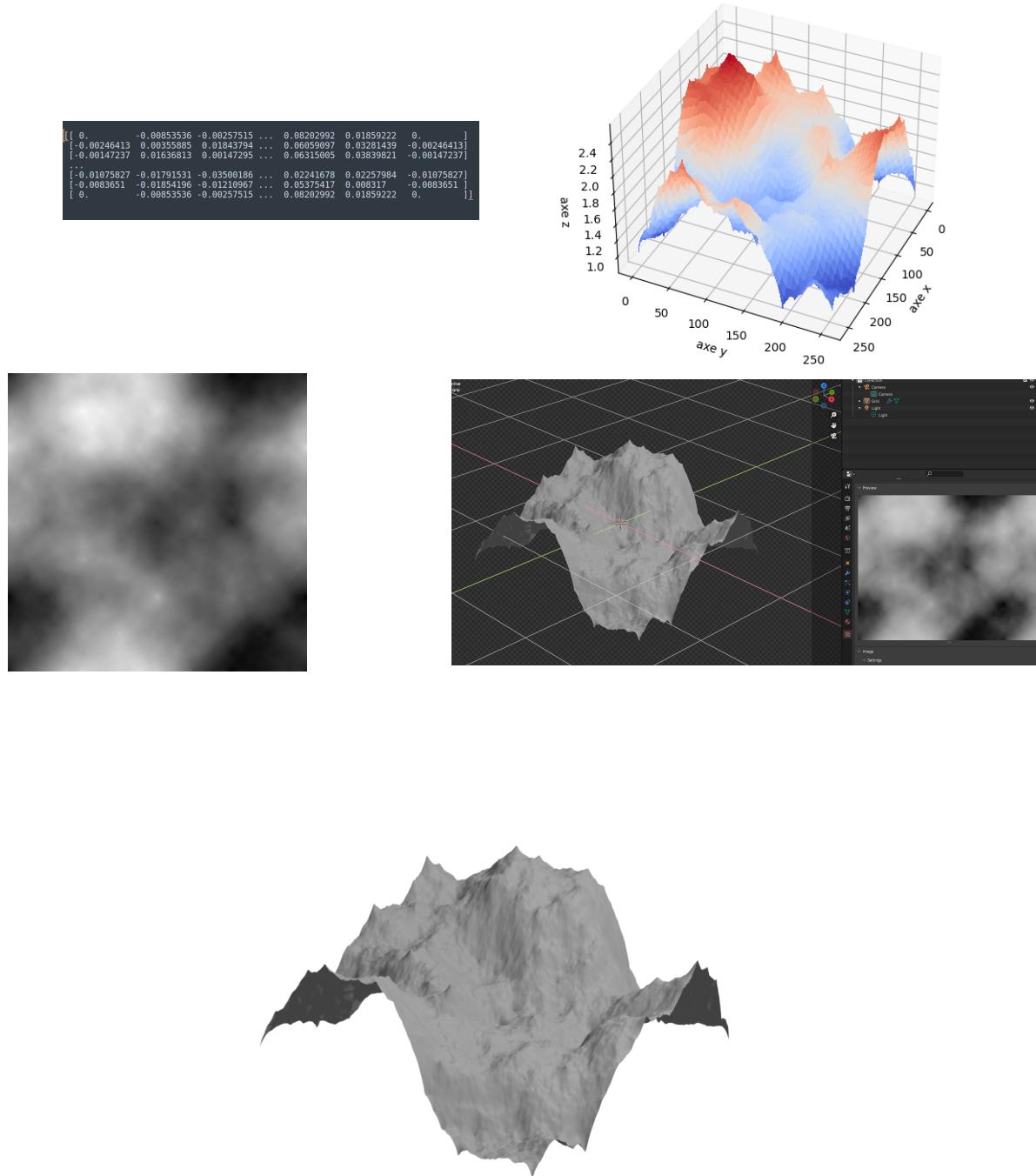
**Carte de hauteurs – *heightmap*.**

Comment utiliser les surfaces obtenues ? Tout d'abord les données produites se résument à un tableau de taille $(N + 1) \times (N + 1)$ où chaque entrée en (i, j) est un nombre réel représentant l'altitude du terrain au-dessus du point (i, j) . Les données se transforment aisément en une image en niveaux de gris, appelée **carte de hauteurs (*heightmap*)**, sa taille est $(N + 1) \times (N + 1)$ et chaque niveau de gris représente une hauteur mise à l'échelle et arrondie. Par exemple 0 (noir) pour l'altitude la plus basse et 255 (blanc) pour l'altitude la plus haute.

Cette image peut ensuite être importée dans un logiciel de traitement d'images (par exemple *Blender*) pour reconstituer le paysage et obtenir un rendu 3D avec éclairage, texture...

Exemple.

Construction d'une carte de hauteurs et rendu 3D.



Niveau de la mer.

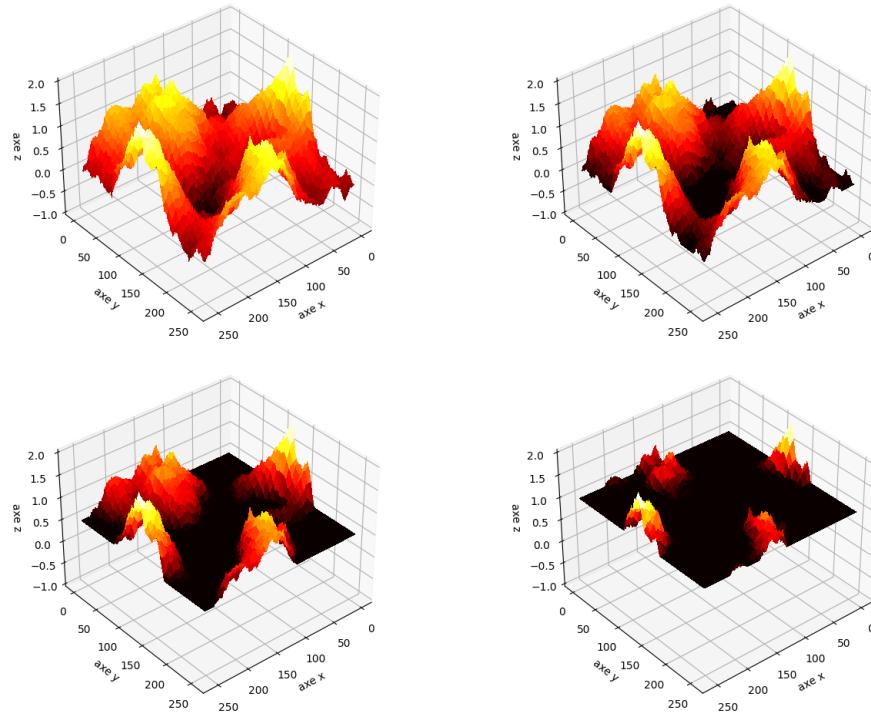
On peut créer un paysage de bord de mer : tout ce qui a une hauteur négative est ramené à l'altitude 0 par la formule $h' = \max(h, 0)$. Dans la pratique on fixe le niveau de l'eau à une altitude h_e et on applique la formule :

$$h' = \max(h, h_e)$$

qui conserve la hauteur h si on est au-dessus de la mer et ramène l'altitude à h_e sinon.

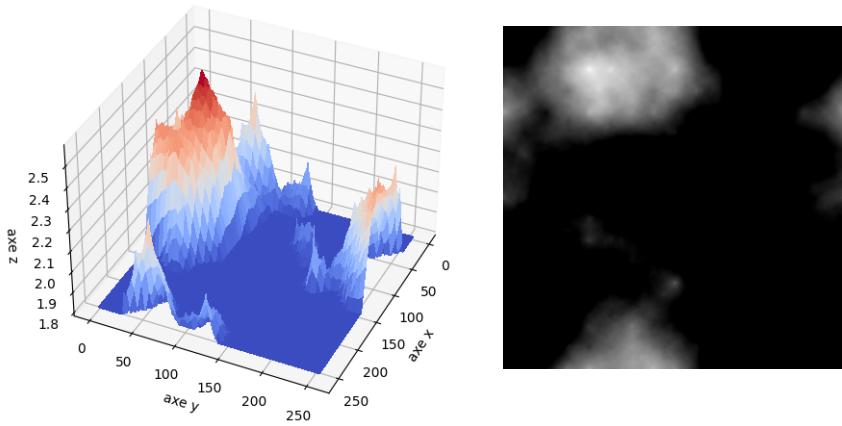
Exemple.

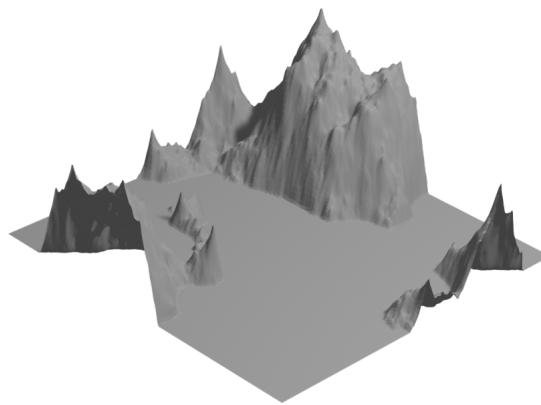
Voici le changement de paysage lorsque le niveau de l'eau monte.



Exemple.

Avec rendu 3D



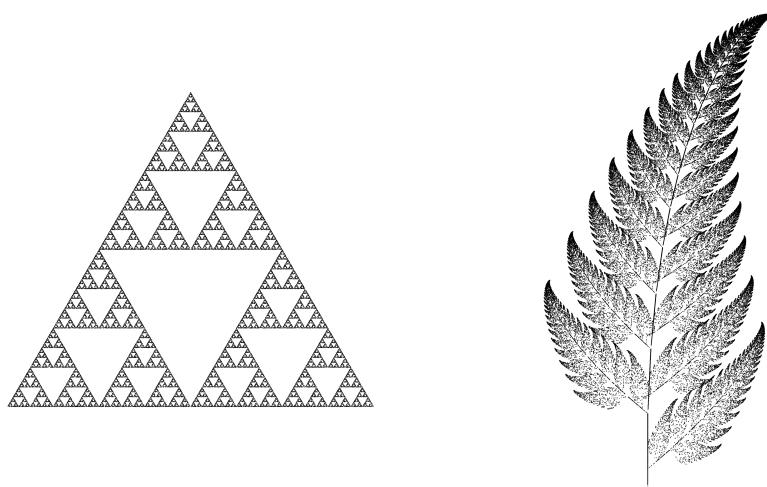


(a) La visualisation Python avec matplotlib. (b) L'image en niveaux de gris (heightmap). (c) Rendu 3D.

2. IFS

2.1. La fougère de Barnsley

Les IFS (pour *Iterated Functions System*) sont des fractales que l'on trace point par point. Elles permettent de dessiner très facilement des figures géométriques comme le triangle de Sierpinski, mais aussi des objets de la nature, en particulier des feuilles comme la fougère de Barnsley.



Ce qui est surprenant, c'est que ces dessins sont obtenus par un processus aléatoire et nécessitent très peu de données, par exemple la fougère est dessinée à partir de 24 nombres réels seulement.

2.2. Transformations du plan

C'est le moment d'aller relire la section « Transformations du plan » du chapitre « Matrices ». On rappelle juste ici la définition générale.

Une **transformation affine** du plan est l'application $F : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ définie par :

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix},$$

où a, b, c, d, e, f sont des réels quelconques.

En d'autres termes, l'image d'un point (x, y) du plan est le point $F(x, y) = (x', y')$ avec

$$\begin{cases} x' = ax + by + e \\ y' = cx + dy + f \end{cases}.$$

En fait, une transformation affine F est la composée d'une transformation linéaire

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto A \begin{pmatrix} x \\ y \end{pmatrix} \quad \text{où } A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

suivie d'une translation

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \end{pmatrix} + B \quad \text{où } B = \begin{pmatrix} e \\ f \end{pmatrix}.$$

2.3. Algorithme

La fractale associée à un système itéré de fonctions s'appelle un **attracteur**. Comment tracer l'attracteur d'un système itéré de fonctions ? Soient F_1, F_2, \dots, F_ℓ des transformations affines. La méthode la plus efficace afin de tracer la fractale est appelée « jeu du chaos » : partant d'un point quelconque $P_0 \in \mathbb{R}^2$, on construit une suite de points par récurrence. Si P_k est construit, alors on choisit aléatoirement l'une des transformations F_i et on définit $P_{k+1} = F_i(P_k)$. On laisse de côté les premiers points (disons les 100 premiers qui peuvent être « loin » de l'attracteur) et on trace les points suivants.

Algorithme.

Algorithme « jeu du chaos ».

Entrée : une famille $\{F_1, F_2, \dots, F_\ell\}$ de transformations affines.

Sortie : un tracé approché de la fractale.

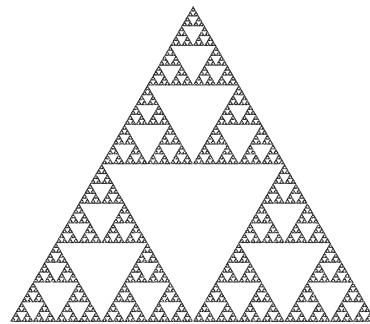
- Fixer $P_0 \in \mathbb{R}^2$ au hasard.
- Pour $k \in \{0, \dots, N_{\max}\}$:
 - on choisit au hasard $i \in \{1, \dots, \ell\}$,
 - on pose $P_{k+1} = F_i(P_k)$,
 - à partir de $k \geq N_{\min}$ on affiche ce point.

N_{\max} représente le nombre de points à calculer (par exemple $N_{\max} = 10\,000$). N_{\min} est une constante (par exemple $N_{\min} = 100$) qui permet de ne pas tracer les tout premiers points qui peuvent être loin de l'attracteur.

Probabilités. En plus, pour optimiser les calculs, on ne choisit pas toujours les transformations F_i de façon équiprobable mais avec une probabilité p_i où $p_1 + p_2 + \dots + p_\ell = 1$.

2.4. Exemples

Le triangle de Sierpinski.



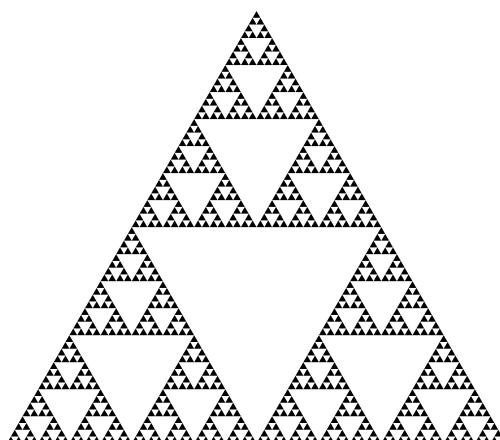
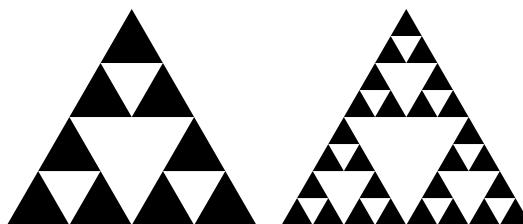
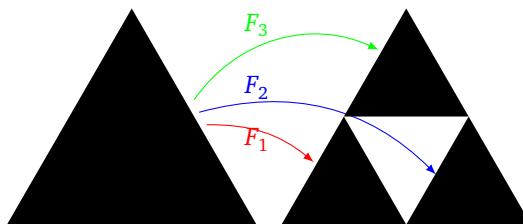
Voici les valeurs numériques (approchées) qui définissent cette fractale.

	a	b	c	d	e	f	p
F_1	0.5	0	0	0.5	0	0	0.33
F_2	0.5	0	0	0.5	0.5	0	0.33
F_3	0.5	0	0	0.5	0.25	0.43	0.33

Pour vraiment comprendre l'action des trois fonctions, partons d'un triangle équilatéral S_0 dont les sommets sont $(0, 0)$, $(1, 0)$, $(\frac{1}{2}, \frac{\sqrt{3}}{2})$. On considère les trois transformations affines suivantes :

$$F_1 \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{2} \begin{pmatrix} x \\ y \end{pmatrix} \quad F_2 \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{2} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix} \quad F_3 \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{2} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{4} \\ \frac{\sqrt{3}}{4} \end{pmatrix}$$

Chacune des transformations F_i est la composition d'une homothétie de rapport $\frac{1}{2}$ et d'une translation. À la première étape, chacune envoie donc le grand triangle équilatéral S_0 sur un triangle plus petit. Géométriquement, c'est comme si on retirait un triangle au centre de chaque plus gros triangle. On a représenté S_0 , S_1 , S_2 , S_3 et S_6 , ce qui donne une bonne idée de l'attracteur, appelé le *triangle de Sierpinski*.



La fougère de Barnsley

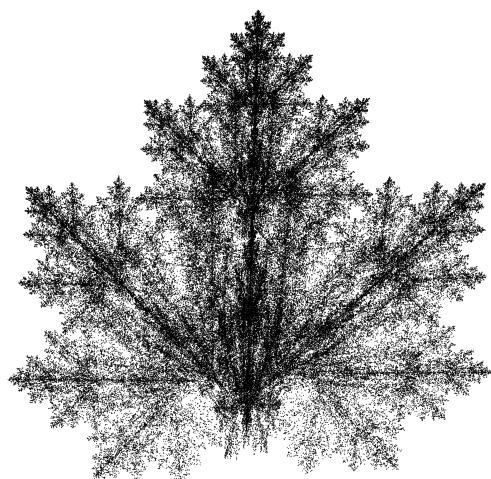


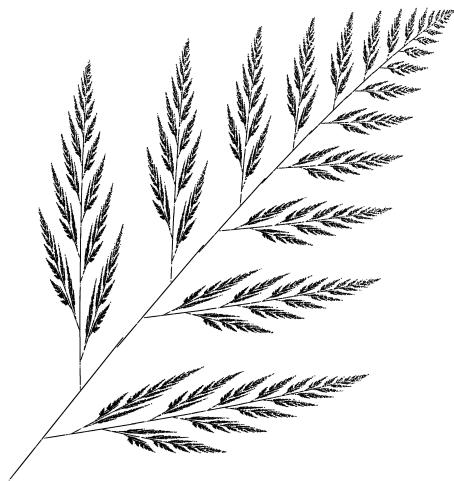
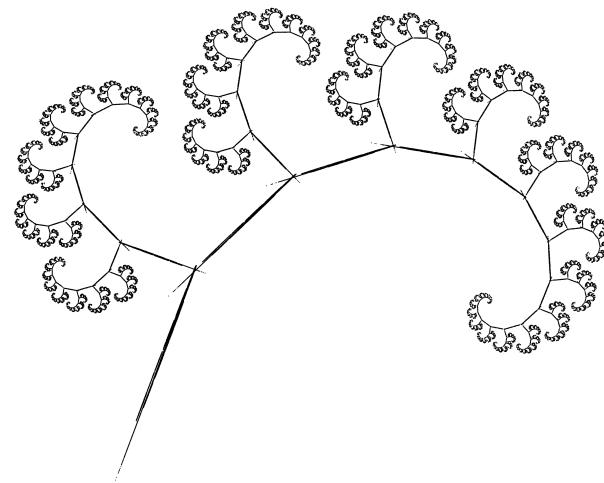
	a	b	c	d	e	f	p
F_1	0	0	0	0.16	0	0	0.01
F_2	0.85	0.04	-0.04	0.85	0	1.6	0.85
F_3	0.2	-0.26	0.23	0.22	0	1.6	0.07
F_4	-0.15	0.28	0.26	0.24	0	0.44	0.07

De part la nature itérative de la construction il est difficile de comprendre ce que fait chaque transformation. Par exemple, la première transformation F_1 est une projection composée avec une homothétie d'un petit rapport. Elle envoie toute la fougère sur la base de la tige (la portion verticale).

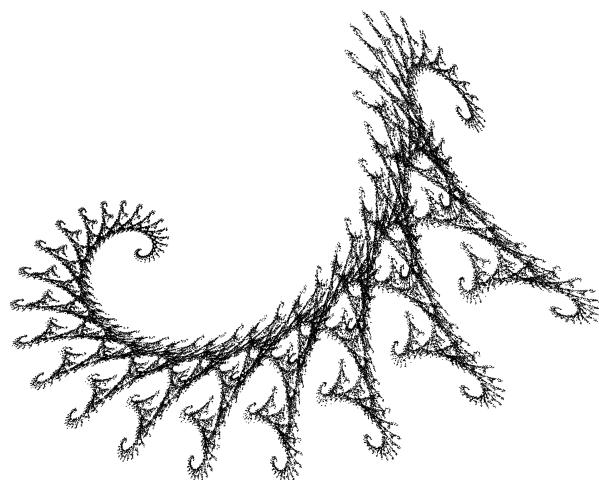
Autres exemples.

D'autres formes de feuilles.

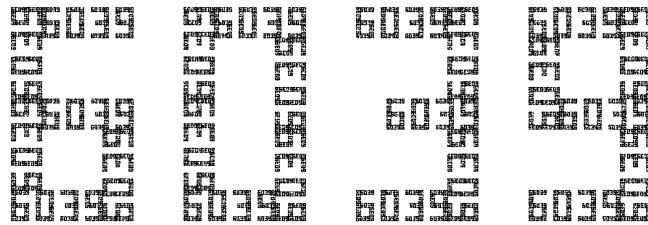




Un hippocampe (?).



Ci-dessous un exemple d'application du « théorème de collage » qui permet d'approcher n'importe quel ensemble voulu comme attracteur d'un système itéré de fonctions.



3. L-systèmes

Les L-systèmes ont été inventés par le botaniste A. Lindenmayer afin de modéliser les plantes. À partir d'un mot initial et d'opérations de remplacement, on arrive à des mots compliqués. Lorsque l'on « dessine » ces mots, on obtient de superbes figures fractales.

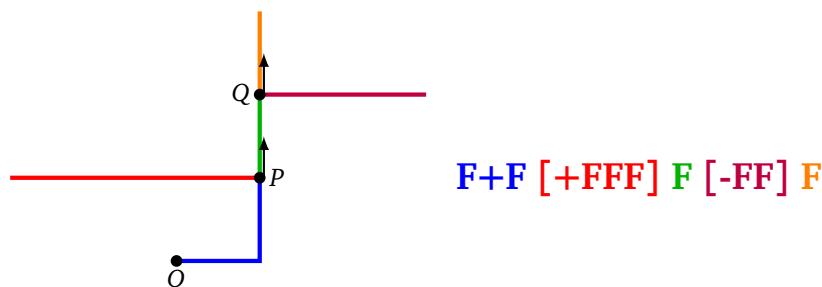
3.1. Tortue

La « tortue » est un langage simple pour dessiner : une tortue se déplace dans le plan (ou l'espace) et trace un trait sur son passage. Elle exécute les instructions selon une suite de lettres :

- **F** ou **G** : avance d'une quantité fixée (en traçant, F pour *forward*),
- **+** : tourne à gauche, sans avancer, d'un angle fixé (le plus souvent 90°),
- **-** : tourne à droite d'un angle fixé.
- **[** : mémorise la position de la tortue (coordonnées (x, y) et direction).
- **]** : repart de la position mémorisée par le crochet [correspondant.
- **X, Y** : ne font rien.

Exemple.

Comprendons l'exemple du tracé du **F+F [+FFF] F [-FF] F**.



- **F+F** : on part du point O , on avance, on tourne, on avance.
- **[+FFF]** : on retient la position actuelle (le point P) et aussi la direction ; on tourne, on avance trois fois (on trace le segment rouge) ; à la fin on replace la tortue à la position P (sans tracer et avec la même direction que celle auparavant).
- **F** : depuis P on avance (segment vert).
- **[-FF]** : on retient la position Q et la direction, on tourne et on trace le segment violet. On revient en Q

avec l'ancienne direction.

- **F** : depuis Q on trace le dernier segment.

Exercice.

Programmer une fonction qui, en entrée reçoit une chaîne de caractères, et trace les instructions correspondantes.

Indications. Avec *Python* il existe une librairie *turtle* qui fournit les instruction `forward()`, `left()`, `right()`.

Pile. Il est beaucoup plus délicat de gérer les crochets ! La méthode qui rend les choses très simples est d'utiliser une « pile » (concept que l'on n'expliquera pas ici).

- Au départ la pile est vide.
- On lit un par un les caractères du mot.
- Si le caractère est le crochet ouvrant « [» alors on ajoute à la pile la position et la direction courante de la tortue $((x, y), \theta)$ (que l'on obtient par `(position(), heading())` du module *turtle*).
- Si le caractère est le crochet fermant «] » alors on dépile (dépiler c'est lire l'élément du haut de la pile et le retirer). On met la position de la tortue et l'angle avec les valeurs lues (utiliser `goto()` et `setheading()`).

3.2. Chercher–Remplacer–Itérer

Un **L-système** est la donnée d'un mot initial et de règles de remplacement. Voici un exemple avec le mot de départ et une seule règle :

initialisation : **G+F-G** règle : **F → FGF**

Le **k-ème itéré** du L-système s'obtient en appliquant k fois la substitution au mot de départ. Avec notre exemple :

- Première itération. Le mot de départ est **G+F-G**, la règle est **F → FGF** : on remplace le **F** par **FGF**. On obtient le mot **G+FGF-G**.
- Deuxième itération. On part du mot obtenu **G+FGF-G**, on remplace les deux **F** par **FGF** : on obtient le mot **G+FGFGFGF-G**.
- Le troisième itéré est **G+FGFGFGFGFGFGF-G**, etc.

Lorsqu'il y a deux règles (ou plus) il faut les appliquer en même temps. Voici un exemple de L-système à deux règles :

initialisation : **F** règle 1 : **F → G+F** règle 2 : **G → GG**

Avec notre exemple :

- Première itération. Le mot de départ est **F**, on applique la première règle **F → G+F** (la seconde règle ne s'applique pas, car il n'y a pas encore de **G**) : on obtient le mot **G+F**.
- Deuxième itération. On part du mot obtenu **G+F**, on remplace le **F** par **G+F** et en même temps le **G** par **GG** : on obtient le mot **GG+G+F**.
- Le troisième itéré est **GGGG+GG+G+F**, etc.

Exemple.

Considérons le L-système du flocon de Koch :

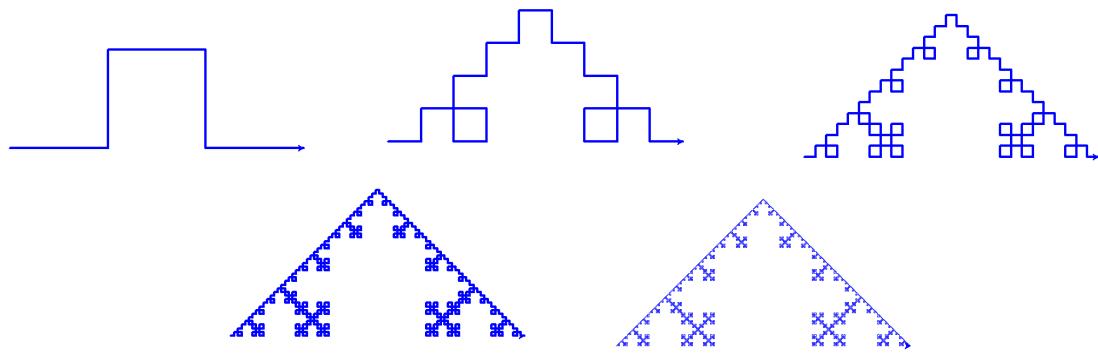
initialisation : **F** règle : **F → F+F-F-F+F**

- le premier itéré ($k = 1$) est **F+F-F-F+F**.
- le deuxième itéré ($k = 2$) est :

F+F-F-F+F+F-F-F+F-F-F+F-F-F+F+F+F+F-F-F+F

- avec le troisième itéré ($k = 3$), on obtient **F+F-F-F+F+...** un mot de 249 lettres.

Voici les images pour $k = 1$ jusqu'à $k = 5$ qui correspondent aux premières étapes d'un flocon de Koch.



Exemple.

Considérons le L-système du triangle de Sierpinski avec deux règles de remplacement :

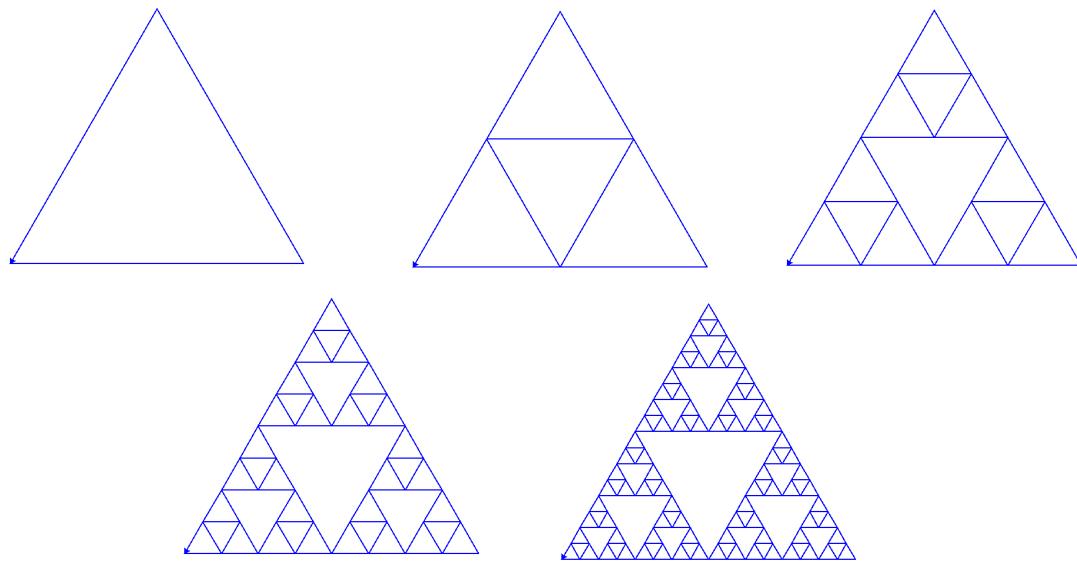
initialisation : **F-G-G** règle 1 : **F → F+G+F-G-F** règle 2 : **G → GG**

Par exemple, avec :

- $k = 1$, **F+G+F-G-GG-GG**,
- pour $k = 2$:

F-G+F+G-F-GG+F-G+F+G-F+GG-F-G+F+G-F-GGGG-GGGG

Lorsqu'on trace ces mots (avec un angle de -120° pour l'instruction **+** et 120° pour l'instruction **-**) on obtient la construction du triangle de Sierpinski.

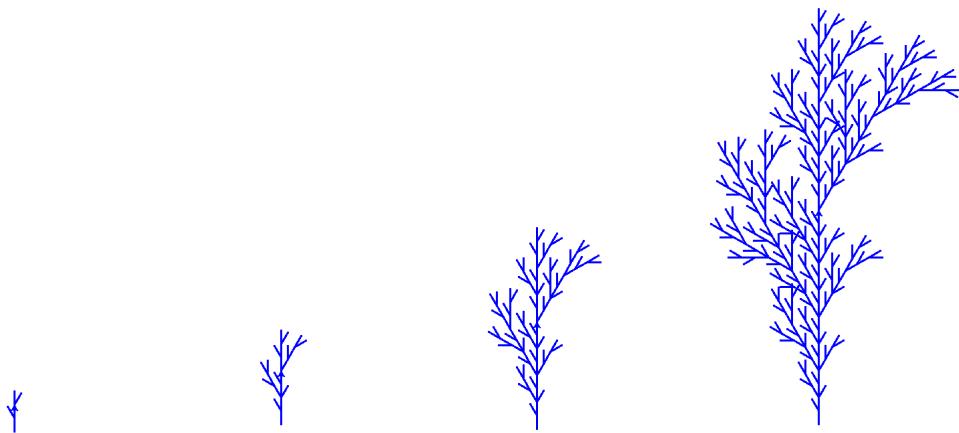


3.3. Plantes en 2D

Voici des exemples de L-systèmes dont le tracé ressemble à des plantes. Pour ces exemples, l'angle est généralement fixé à 30° .

Voici les quatre premiers tracés du L-système défini par :

initialisation : **F** règle : **F → F[+F]F[-F][F]**

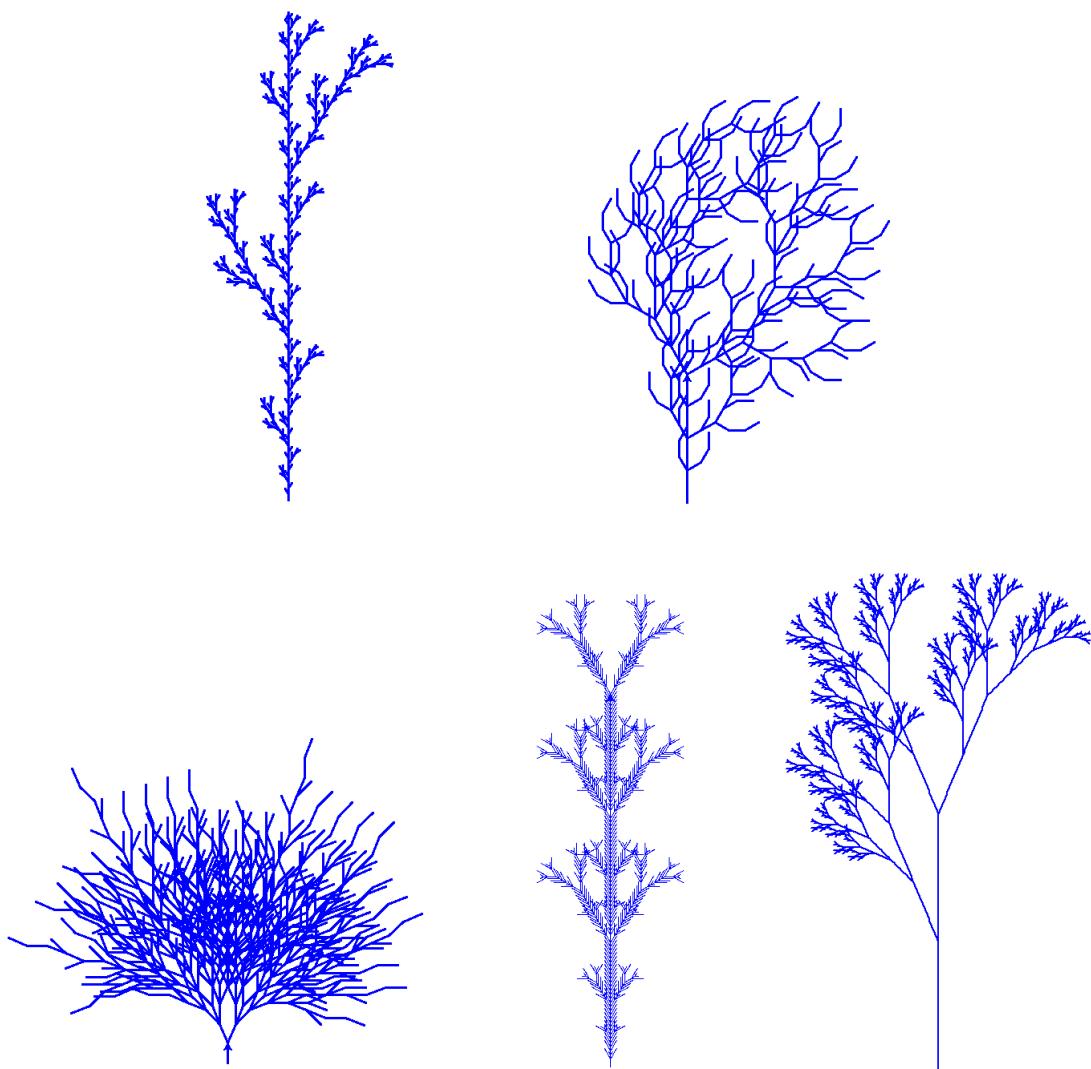


Ci-dessous, à gauche :

initialisation : F règle : $F \rightarrow F[+F]F[-F]F$

Ci-dessous, à droite :

initialisation : F règle : $F \rightarrow FF[-F+F+F]+[+F-F-F]$



De gauche à droite :

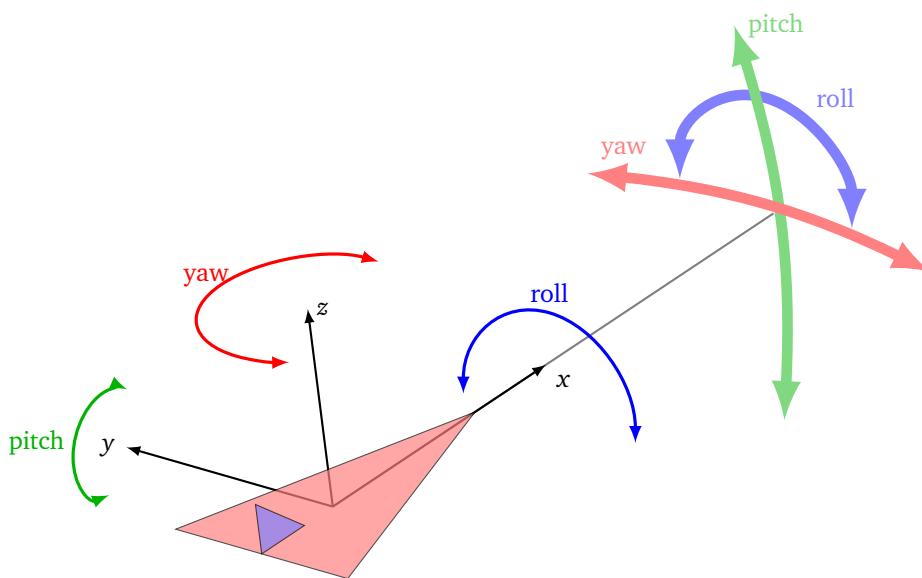
initialisation : F	règle 1 : $F \rightarrow F[-G][+G]$	règle 2 : $G \rightarrow F[-G]F[+F-G]$
initialisation : YYY	règle 1 : $X \rightarrow X[-FFF][+FFF]FX$	règle 2 : $Y \rightarrow YFX[+Y][-Y]$
initialisation : X	règle 1 : $X \rightarrow F[+X]F[-X]+X$	règle 2 : $F \rightarrow FF$

3.4. Plantes en 3D

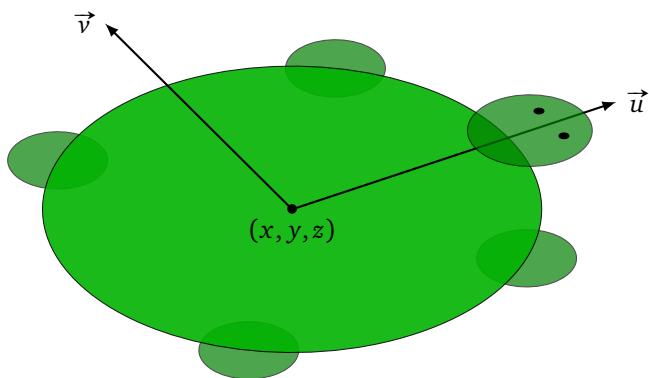
Tortue 3D

Souvenez-vous des trois mouvements de rotation possibles *yaw/pitch/roll* d'un objet de l'espace, par exemple un avion :

- lacet/*yaw* : rotation autour d'un axe haut/bas (un angle positif déplace le nez de l'appareil vers la gauche).
- tangage/*pitch* : rotation autour de l'axe des ailes (un angle positif monte le nez de l'appareil),
- roulis/*roll* : rotation autour de l'axe de la cabine (un angle positif fait monter l'aile gauche et baisser l'aile droite).



Jusqu'ici notre tortue se mouvait dans le plan. Une tortue de l'espace est caractérisée par la position (x, y, z) de son centre, un vecteur \vec{u} pointant vers l'avant et un vecteur \vec{v} indiquant la gauche de la tortue. Le vecteur $\vec{w} = \vec{u} \wedge \vec{v}$, dirigé vers le haut, complète ces vecteurs en un repère orthonormal direct.



On étend le langage de la tortue afin qu'elle puisse se déplacer dans l'espace :

- **F** ou **G** : avance d'une quantité fixée,
- **+** : tourne vers la gauche, sans avancer, d'un angle fixé (lacet à gauche).
- **-** : tourne vers la droite d'un angle fixé (lacet à droite).

- & : baisse le nez d'un angle fixé (tangage vers le bas).
- ^ : monte le nez d'un angle fixé (tangage vers le haut).
- < : effectue un roulis vers la gauche d'un angle fixé.
- > : effectue un roulis vers la droite d'un angle fixé.
- | : fait demi-tour (lacet de 180°).
- [: mémorise la position de la tortue (coordonnées (x, y, z) et vecteurs \vec{u} et \vec{v}).
-] : repart de la position mémorisée par le crochet [correspondant.
- X, Y : ne font rien.

L-système 3D

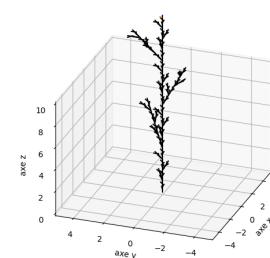
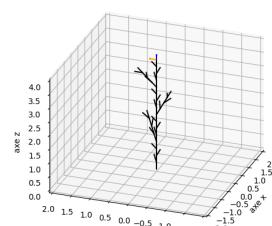
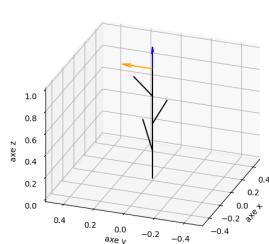


Considérons le L-système :

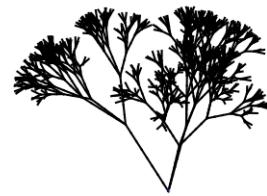
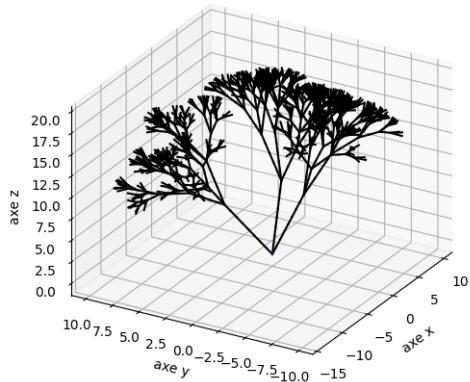
initialisation : F règle : $F \rightarrow F[\&+F]F[->F]F[+^F]F$

L'angle est $\frac{\pi}{8}$.

Voici les tracés des trois premières itérations.



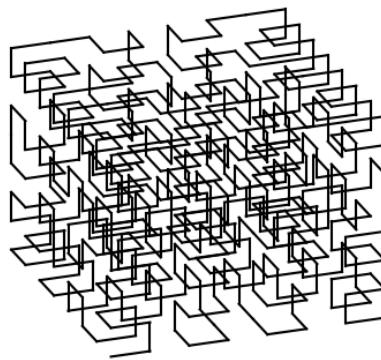
Voici un autre exemple avec deux vues différentes.



Des améliorations sont possibles pour aller plus loin dans le réalisme :

- ajouter une texture et des couleurs,
- diminuer le diamètre et la longueur des branches au fur et à mesure des itérations,
- ajouter des feuilles au bout de chaque branche terminale,
- faire une version stochastique c'est-à-dire décider aléatoirement si une branche doit être dessinée ou pas.

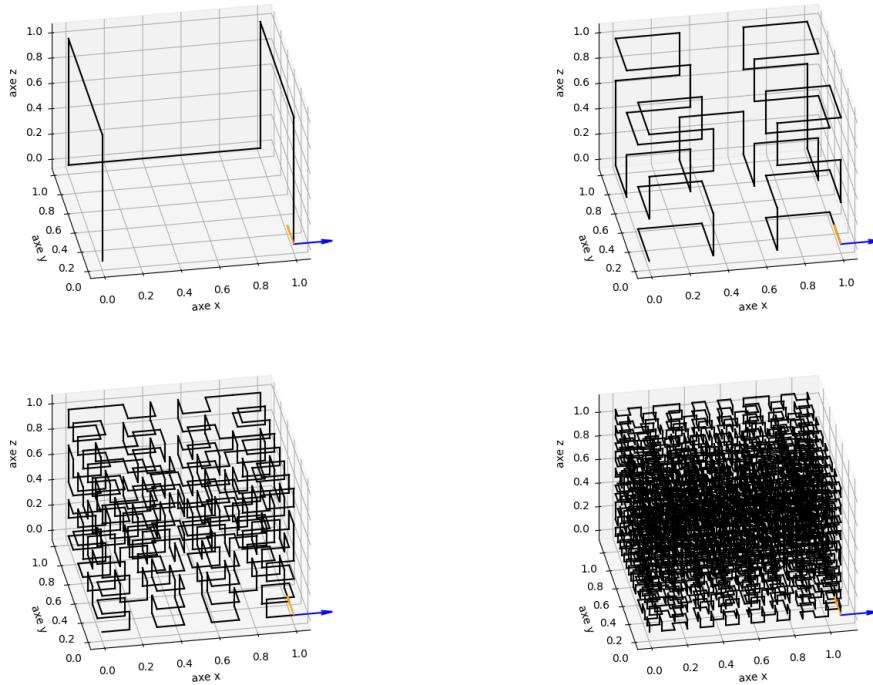
Courbe de Hilbert 3D



Terminons par une courbe continue de l'espace qui remplit complètement le cube unité. Cette courbe peut être approchée par les itérés du L-système :

initialisation : X règle : X → ^ <XF^ <XFX-F^ >>XFX&F+>>XFX-F>X->

L'angle est $\frac{\pi}{2}$.



Cela peut être utile si un personnage ou un robot doit inspecter de près chaque point du volume d'une pièce.

Certains paragraphes sont extraits d'un chapitre « Systèmes itérés de fonctions » d'un cours de géométrie, d'autres du chapitre « L-système » du livre « Python au lycée » dans lesquels vous trouverez plus de détails. Vous trouverez sur notre site une librairie `turtle3d` afin de tracer vos L-systèmes en dimension 3.