Serpents et Dragons

La première partie : montre des images qui font deviner qu'il se passe des choses
La seconde : fournit des preuves
La troisième : est consacrée à la façon dont on obtient les images (en utilisant caml ou Maple)
La quatrième : énumére ce qui resterait à faire ...
La dernière : contient des références, des sources papier ou web

I Les objets concernés

A le serpent S

1 le Serpent : version papier

a préparations matérielles

Le matériau idéal n'existe plus : c'étaient les bandes-d'entraînement-à-trous du papier des imprimantes de jadis : vous en preniez 4 (bord de) feuilles

Si vous voulez suivre les indications qui suivent, vous pouvez encore avec un rouleau de papier d'imprimante thermique (fax ou caisse enregistreuse un peu vieille), mais il faut le repasser

Ces choses n'existant plus beaucoup, vous prenez une feuille A_3, vous la pliez en 4 dans le sens de la longueur, vous découpez les bandes obtenues, vous les scotchez pour avoir une bande de environ 1m20

b pliage

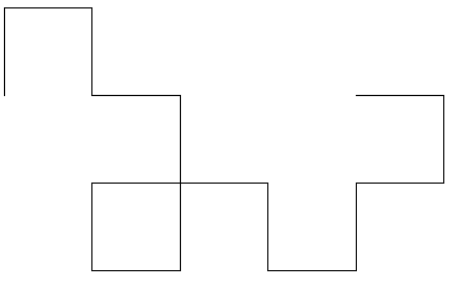
Vous pliez en deux, puis en deux, puis en deux ... attention lors du n-ième pliage, amenez bien le (n-1)-ième pli sur le (n-2)-ième et non sur les bouts libres. Si vous n'avez rien comppris c'est que vous n'avez pas de papier en mains, retournez au § précédent

c dépliage

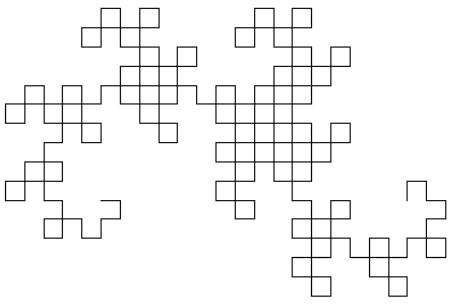
Quand vous avez fini de plier votre bande, vous la dépliez en faisant de telle sorte que chacun des vos (dé-)plis soit un angle droit (ou gauche) dans le sens que vous indique le papier

Tout cela est alors mis en équilibre sur la tranche et (selon que vous avez plié 4 ou 8 fois) vous voyez :

Serpent 4



Serpent 8



2 le Serpent : version programmation

Voir le dernier §

3 le Serpent : version transducteur

C'est la même idée, mais avec des mots à la place du papier. Un "transducteur" c'est juste une fonction dont la source comme le but sont des mots (finis ou infinis). Avec un mot (= liste de lettres finie ou infinie) on peut le tronquer, ne garder que les lettres de rang multiple de 3, le passer en MAJUSCULES, sil est fini on peut aussi : l'écrire 4 fois, le recopier à l'envers, enchaîner 7 des opérations précédentes

Ci-dessous les mots sont écrits avec un alphabet à 2 lettres : D et G (pour les plis Droit et Gauche du papier précédent) et on définit la suite de mots (le V est l'initiale de Virages) :

V(1) = D
V(2) = DDG
V(3) = DDGDDGG
V(4) = DDGDDGGDDDGGDGG
V(5) = DDGDDGGDDDGGDGGDDDGDDGGGDDGGDGG

Pour passer de V(n) à V(n+1) on peut :

1. Recopier V(n) ajouter un D puis "recopier V(n) à l'envers en échangeant les D et les G"
2. Distendre V(n) en copiant sa k-ième lettre en 2k-ième position, puis en insérant aux rangs impairs DGDGDGDGDG ...

La preuve de l'identité des résultats des deux méthodes peut se faire par récurrence

Chaque mot étant préfixe du précédent, on pourrait définir V(∞) mais ça ne servira pas ici

Pour tracer le serpent S(1) : une chenille part d'un point, se dépace de 1 cm (ou un carreau ou un demi ...), comme V(1)=D elle effetue un 1/4 tour Droit, enfin elle trace un dernier segment

Pour tracer le serpent S(2) : une chenille part d'un point, se dépace de 1 cm (ou un carreau ou un demi ...), comme V(2)=DDG elle effetue un 1/4 tour Droit, un segment, 1/4 tour Droit, un segment,1/4 tour Gauche, enfin elle trace un dernier segment

Si elle trace S(5) elle fera ... un grand nombre de 1/4 tour D ou G

A condition de bien choisir la longueur des segments tracés (il faut multiplier la longueur par $\frac{\sqrt{2}}{2}$ à chaque fois) et l'orientation du premier déplacement (en passant de n à n+1 on tourne de $\frac{\pi}{4}$ à chaque fois), S(n+1) repasse par les coins de S(n) en ayant effectué un coude Droit ou Gauche entre chaque paire de coins consécutifs de S(n) : c'est la définition 2 du passage de V(n) à V(n+1). Mais S(n+1) est également constitué de deux S(n) (plus petit et tournés) enchaînés par un virage : c'est la définition 1

4 le Serpent : version point fixe d'un ensemble de contractions

Ci-dessous r = $\frac{1+i}{2}$, donc $r^2 = \frac{i}{2}$, r est le complexe dont l'interprétation géométrique est la similitude plane d'angle $\pi/4$ et de rapport $\sqrt{2}/2$, celle qui transforme la diagonale de [0..1]x[0..1] en le segment 0..I (I est l'image de i)

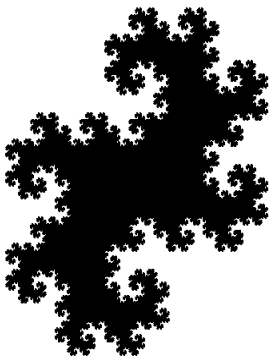
Je définis alors deux similitudes : s_0 est le produit par r et s_1 est $z \rightarrow r + \bar{r}z$ puis je définis l'application φ de l'ensemble des parties de $\mathbb{C}$ dans lui même par : $\varphi(A) = s_0(A) \bigcup s_1(A)$

Vous partez maintenant d'une quelconque partie bornée du plan (un segment, un point, 14 points, un disque) et vous appliquez φ à répétition : les $\varphi^n(A)$ convergent vers le serpent S

Pour l'expérimenter dans l'esprit des § précédents, vous pouvez prendre A=[0..1], appliquez s_0 vous avez rA, tournez-le : ça i r A mieux puis ajoutez r, unissez : vous avez un dos d'âne. Vous re-recommencez et vous obtenez le début de S_4 déjà dessiné au § précédent

On va reprendre et détailler cela au § II avec l'idée de faire des preuves propres

B le Dragon D



1 Généralités de numération

Chaque entier n $\in \mathbb{N}$ est représenté décimalement comme une somme de puissances de 10 multipliées par des chiffres $1789 = 1\times10^3 + 7\times10^2 + 8\times10^1 + 9\times10^0$. Pour représenter les nombres entre 0 et 1 on fait pareil avec des puissances négatives : $0,1789 = 1\times10^{-1}+7\times10^{-2}+8\times10^{-3}+9\times10^{-4}$. Pour représenter "tous le réels" on fait pareil avec un passage à la limite : $\frac{1}{3} = \sum_{k\in\mathbb{N}^*} 3\times10^{-k}$

Si vous remplacez la base b=10 par b=2 et ne prenez comme chiffres que 0 ou 1 vous avez la numération binaire. Tous les nombres de [0..1] sont des $\sum_{n\in A}(\frac{1}{2})^n$, et "beaucoup" de nombres ont deux représentations : $2^{-1} = \sum_{k\geq 2} 2^{-k}$

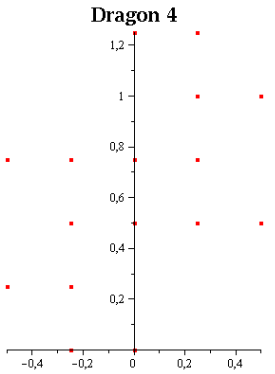
Si vous remplacez b=10 par b=1/3 et ne prenez comme chiffres que 0 ou 2 vous aurez les $\sum_{n\in A} 2\frac{1}{3}^n$: c'est l'ensemble de Cantor : au facteur 2 près, c'est juste l'ensemble des sommes de puissances de $\frac{1}{3}$

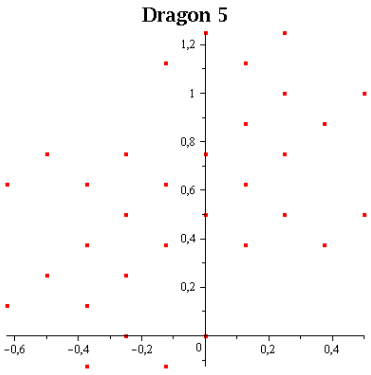
Le Dragon qui a débuté ce § est obtenu par la même méthode : j'ai choisi b=$\frac{1+i}{2}$ l'ensemble des résultats des $\sum_{n\in A} b^n$ (pour $_{A\subseteq\mathbb{N}^*}$ est D = "le Dragon de Knuth"

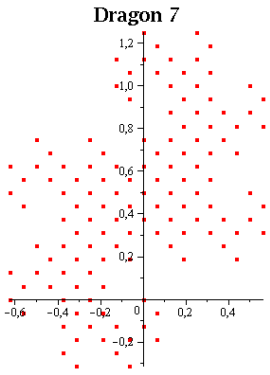
2 D_7 pave $\mathbb{G}$, les D pavent $\mathbb{C}$

Le b=$\frac{1+i}{2}$ utilisé ci-dessus, joue un peu le rôle de $\frac{1}{10}$ pour la numération habituelle; Pour avoir tous les réels (et pas seulement les petits) on utilise aussi 10. Ici on va utiliser c=$\frac{1}{b} = 1-i$. Comme $c^8 = 16$ on ne va considérer les ensembles :

$$\boldsymbol{\mathcal{D}}_m = \{\sum_{k\in A} c^k\} \text{ où } A \subseteq \{0,1...m\} \text{ pour } 1 \leq m \leq 7 \text{ et enfin } D = \{\sum_{k\in A} b^k\}_{A\subseteq\mathbb{N}}$$







En fait les D_4, D_5 et D_7 ont été fabriqués avec b et non c ... (voir les graduations) et cela "ne change rien" car $b^8 = \frac{1}{16}$, $c^8 = 16$: on passe des uns aux autres par une homothétie de rapport 16

Autre ennui technique : parfois il est plus commode d'utiliser $A \subseteq \mathbb{N}$ et parfois $A \subseteq \mathbb{N}^*$: on passe de l'un à l'autre en multipliant par b : c'est une similitude, la forme n'est pas changée

Les images de $\boldsymbol{\mathcal{D}}_4$, $\boldsymbol{\mathcal{D}}_5$, $\boldsymbol{\mathcal{D}}_7$ font deviner ce qui se passe ... et les résultats sont conformes à ce que l'on a deviné :

- $\mathbb{G}$ est pavé par des $\boldsymbol{\mathcal{D}}_7$
- $\mathbb{C}$ peut (presque) être pavé par des D

Ce pavage a lieu modulo $8\mathbb{Z}+ 16$ i $\mathbb{Z}$, le "presque" vient de la non injectivité aux frontières (comme chaque fois que l'on a une sorte de numération non finie)

C les autres dragons d(b)

1 définition

Pour eux on va sauter l'intermédiaire des mini-dragons et passer directement à la limite :

Dans toutes les définitions précédentes, on remplace : "b = $\frac{1+i}{2}$" par "b est un (quelconque) complexe de module strictement inférieur à 1" et :

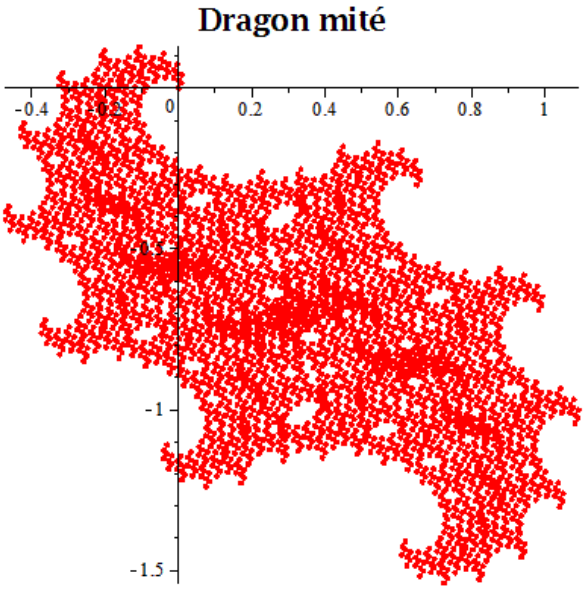
$$d(b) = \{\sum_{k\in A} b^k\}_{A\subseteq\mathbb{N}}$$

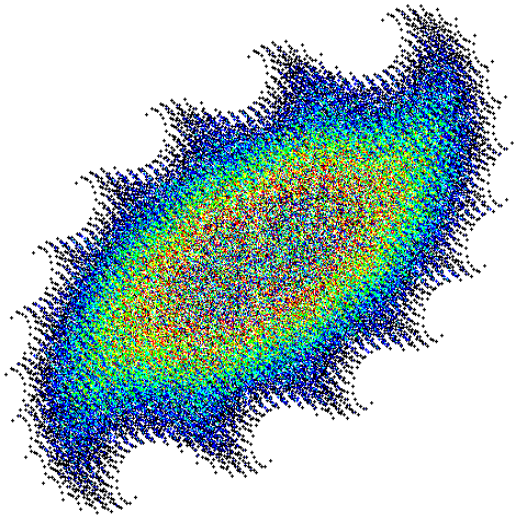
2 galerie de dragons

1. **Dragon Mité** est obtenu pour le choix de b=0.65-0.3 i, en limitant les sommes aux exposants inférieurs à 12 : cela permet de voir que ce Dragon mité semble empli de plein de petits Dragons Mités
2. **Cachemire par vagues** est obtenu pour le choix de b=0.8+0.2 i. Celui là est à visualiser dynamiquement (donc caml imposé, ou un vieux Pascal) avec les options couleurs et gros points : cela ressemble à des vagues se recouvrant les unes les autres
3. **Tartan raté** est obtenu pour le choix de b=0.931 i, j'ai une fois obtenu un effet de tartan très élégant, mais ce que l'on obtient avec le bitmap est dur à reproduire d'un écran à l'autre et pire à imprimer
4. **Set en rafia** est obtenu pour le choix de b=0.97 $\exp(\frac{i\pi}{1.95})$
5. **Le rucher s'effiloche** est obtenu pour le choix de b= -0.49 + 0.97 $\frac{\sqrt{3}}{2}$ i), il faut limiter les puissances à 15
6. **Virus vicieux** est obtenu pour le choix de b=0,9 $\exp(\frac{12i\pi}{13})$
7. **Flamenco** est obtenu pour le choix de b=0.7-0.39 i,
8. **le Dragon retourné** ... est obtenu pour le choix de b=$\frac{-1+i}{2}$ (au lieu de $\frac{1+i}{2}$, d'où la symétrie). Là l'intérêt est de voir qu'il n'y a pas de recouvrements, donc dessin avec des petits points, max = 17. Il y a peut-être des choses à voir avec les gros points couleur

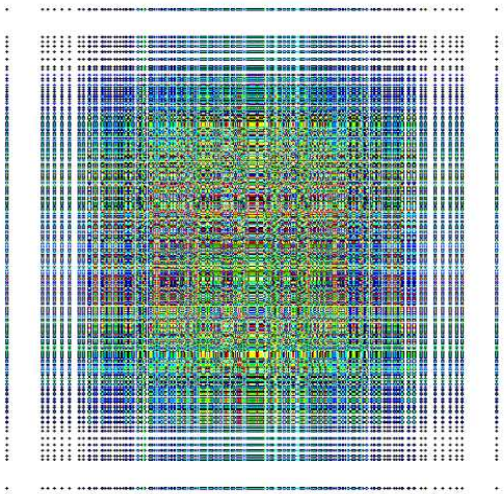
9. **Chou-fleur** est obtenu pour le choix de b=0.4+0.6 i, c'est "assez proche" du b du dragon! mais ici les sous-bouquets ne s'emboîtent pas exactement

10. **Danse macabre** est obtenu pour le choix de b=-0.625 $\exp(\frac{2i\pi}{7})$, le caractère totalement discontinu se devine bien (même s'il est nié par la taille non nulle des pixels)

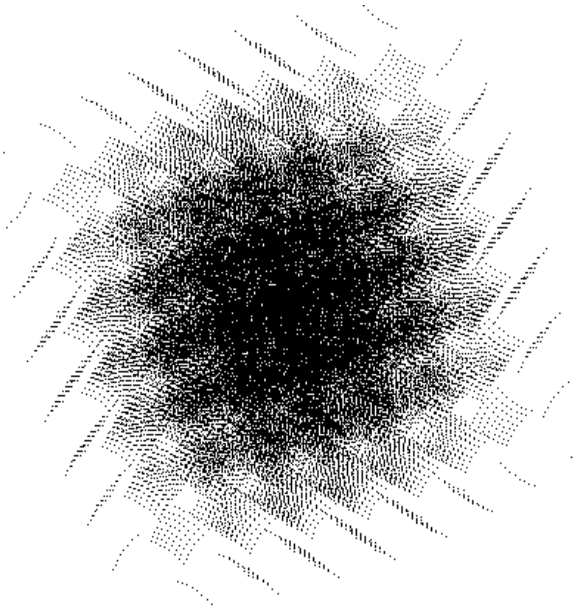




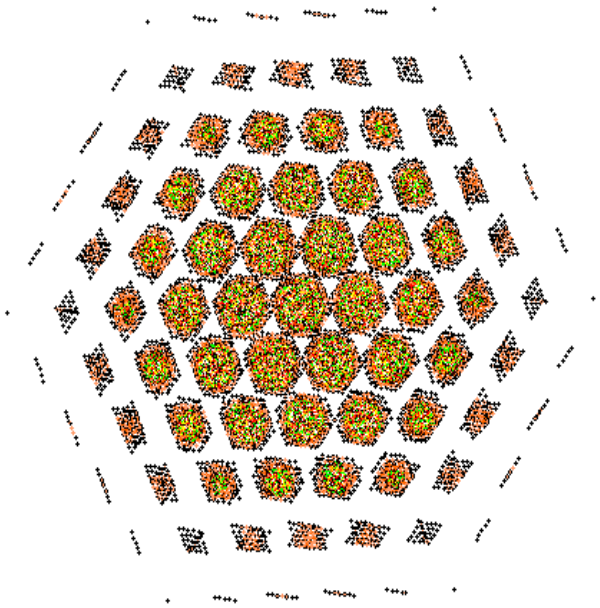
Cachemire par vagues



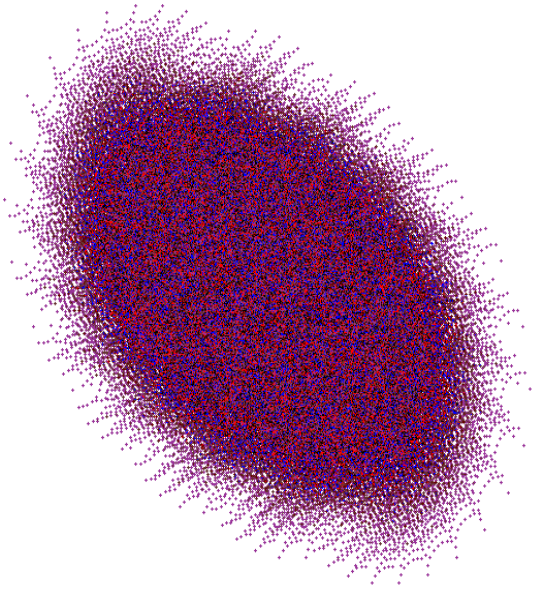
Tartan raté



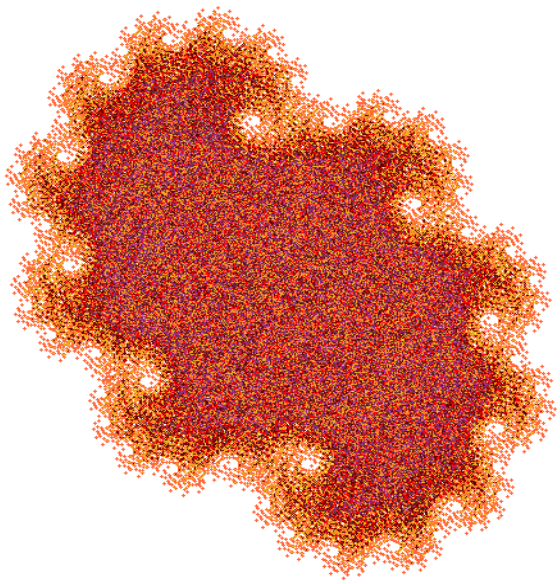
Set en rafia



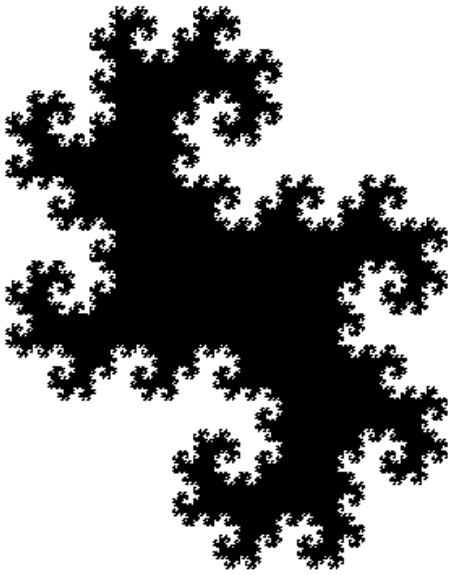
Rucher effiloché



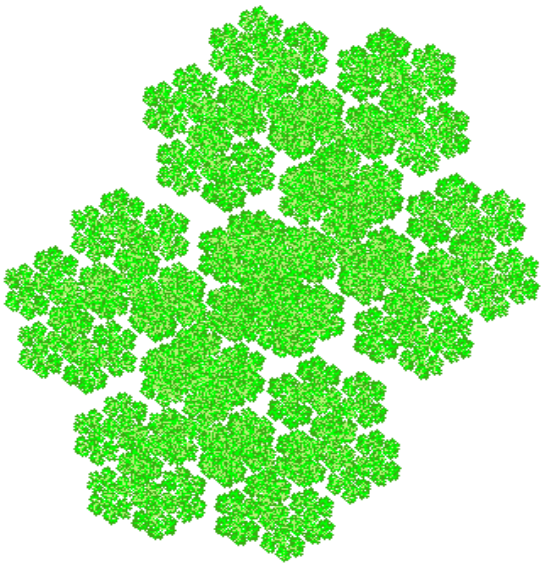
Virus vicieux



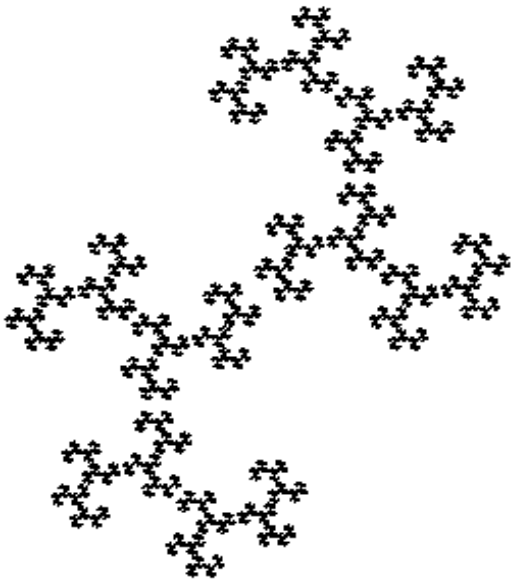
Flamenco



Dragon retourné



Chou fleur



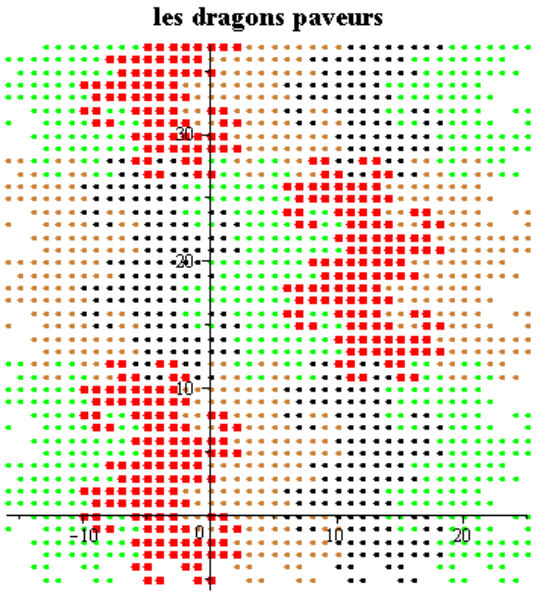
Squelette

II Les preuves

A concerant les dragons

1 les $\mathcal{D}_7$ pavent $\mathbb{G}$

$\mathbb{G}$ c'est $\mathbb{Z}+$ i $\mathbb{Z}=$ les enters de Gauss



Ce n'est pas tous les jours qu'une preuve peut s'effectuer par évidence visuelle : par inspection du bord droit de $\boldsymbol{\mathcal{D}}_7$ nous constatons qu'il s'emboite dans ... son bord gauche ! Il n'y a donc qu'à répeter cela horizontalement à droite et à gauche pour obtenir $\boldsymbol{\mathcal{BD}}$ = une bande de Dragons qui couvre déjà une bonne bande

Ce n'est pas tous les jours qu'une preuve peut s'effectuer par évidence visuelle : par inspection du bord Nord de $\boldsymbol{\mathcal{BD}}$ nous constatons qu'il s'emboite dans ... son bord Sud ! Il n'y a donc qu'à répeter cela verticalement vers le haut et le bas pour obtenir TOUS les complexes à composantes entières

2 les D pavent $\mathbb{C}$

C'est comme pour la numération en base 10 : pour approcher le réel t à 10^{-23} près, on le multiple par 10^{23}, on encadre $t10^{23}$ par deux entiers consécutifs, on re-divise par 10^{23}

On a ici un ennui : D_7 est plus haut que large, cela va dissymétriser les formules. Alors je remplace D_7 par $DD = D_7 \bigcup (D_7 + 8)$ (8 c'est $-c^4$) si bien que le pavage de $\mathbb{G}$ vu ci-dessus s'énonce maintenant $DD + 16\mathbb{G} = \mathbb{G}$

Je m'empresse bien sûr de ré-injecter cette formule dans elle-même : $DD + 16(DD + 16\mathbb{G}) = \mathbb{G}$ et on recommence : $DD + c^8DD + c^{16}DD + c^{24}\mathbb{G} = \mathbb{G}$ et on pourrait bien sûr continuer avec un indice n

Soit maintenant un z de $\mathbb{C}$, je considère z'=c^{24}z, et j'approche z' par un g de $\mathbb{G}$: $|z' - g| \leq \sqrt{2}/2$, en multipliant par b^24 on a alors :

$$|z - (dd_0\ b^{24} + dd_1\ b^{16} + dd_2\ b^8 + g)| \leq \frac{\sqrt{2}}{2}(\frac{1}{\sqrt{2}})^{24}$$

Il reste à continuer avec un n quelconque, tout z de module inférieur à 100 (par exemple) est donc limite d'une suite d'issue d'un nombre fini de translatés de D, tout cela étant compact, la limite est dedans : on peut paver notre disque avec des translatés de D

3 les d(b) sont compacts

Au dernier § il y en a une autre preuve

Dans ce § une approche "directe" : on va prouver que les d(b) sont compacts en tant qu'image continue de l'ensemble de paries $\boldsymbol{\mathcal{P}}$ que voilà :

On note $\boldsymbol{\mathcal{P}}$= $\boldsymbol{\mathcal{P}}(\mathbb{N}^*)$ l'ensemble formé des parties de $\mathbb{N}^*$, il est muni de la distance $d(A, B) = 2^{-\min(A\Delta B)}$ où $A\Delta B = (A\backslash B) \bigcup (B\backslash A)$ On va prouver que $\boldsymbol{\mathcal{P}}$ est compact en prouvant qu'il est précompact (pour tout $\varepsilon > 0$ on peut le recouvrir par un nombre fini de boules de rayon ε) et complet (toute suite de Cauchy converge) : quand on aura cela, la compacité se prouve "comme pour [0..1]", la dichotomie étant généralisée à des recouvrements par des boules de rayons 1,1/2,1/3,...,1/n

a $\boldsymbol{\mathcal{P}}$ est précompact : soit $p \in \mathbb{N}^*$: alors $\boldsymbol{\mathcal{P}}$ est l'union des 2^p parties de la forme $D \bigcup \boldsymbol{\mathcal{P}}((p+1)...\infty)$ pour D partie de $\{1...p\}$ et pour un D donné deux parties comme $D \bigcup U$ et $D \bigcup V$ sont à une distancve au plus $2^{-(p+1)}$

b $\boldsymbol{\mathcal{P}}$ est complet : quand $(A_n)_{n\in\mathbb{N}^*}$ est une suite de Cauchy dans $\boldsymbol{\mathcal{P}}$, dès que ses termes sont distants de moins que 2^{-p} leurs portions dans [1..p] sont égales, la limite de la suite des (A_n) est "l'union" de ces limites

Les d(b) sont des parties bornées de $\mathbb{C}$ (car $|b| < 1$, quand A et B sont deux parties bornées on peut définir $\lambda(B)$=$Sup_{a\in A}$distance(a, B) puis $\lambda(A)$=$Sup_{b\in B}$distance(b, A) et enfin

$$H(A, B) = \max(\lambda(A), \lambda(B))$$

: une fois cela fait on prouve que H est une distance ("la distance de Hausdorff") sur l'ensemble des parties bornées de $\mathbb{C}$

Les définitions précédentes s'emboîtent comme il faut : l'application qui part d'une partie M de $\mathbb{N}$ et lui associe $\varphi(M) = \sum_{k\in M} b^{-k}$ est continue (pour les deux distances citées ci-dessus). On part d'un compact, l'image qui est justement notre d(b) est compacte

4 les d(b) comme points fixes

Ce qui suit formalise ce que l'on a déjà expérimenté et vu sur le cas des dragons : "plus on avance dans la suite de D_k, moins ça varie"; dit de façon un peu moins floue : si on a déjà fabriqué deux morceaux A et B avec exposants de b majorés par 14, quand on passera à A' et B' (exposants majorés par 15), alors A' et B' sont plus proches que A et B l'étaient

Je note $\mathcal{K}$ l'ensemble des parties compactes de $\mathbb{C}$ muni de la distance de Hausdorff et je vais utiliser le fait que cet espace $\mathcal{K}$ est complet.

Pour un b de $\mathbb{C}$ ($|b| < 1$) je définis T_0 et T_1 par $T_0(z) = bz$ et $T_1(z) = b + bz$, ces T_0 et T_1 sont des contractions de $\mathbb{C}$ (ce sont des similitudes de rapport < 1) et la définition de la distance dans $\mathcal{K}$ fait que T_0 et T_1 sont encore des cvontractions de $\mathcal{K}$

On définit alors T=$T_0 \bigcup T_1$ par $T(X) = T_0(X) \bigcup T_1(X)$ et en utilisant le lemme qui dit que $H(X_1 \bigcup X_2, Y_1 \bigcup Y_2) \leq \max(H(X_1, Y_1), H(X_2, Y_2))$ on arrive à T est encore une contraction de rapport |b|

Cette T a donc un unique point fixe (c'est d(b)) qui est la limite des $T^n(K_0)$ pour toute partie K_0 : c'est ce qui a été fait ci-dessus en partant de $K_0 = \{0\}$, les $T^n(K_0)$ sont les complexes sommes de puissances de b inférieures à n

5 la connexité des d(b)

On utilise les notations et résultats du § précédent : B = d(b) est la limite précédente, donc T(B)=B, donc $B = (bB) \bigcup (b + bB) = B_0 \bigcup B_1$, il y a alors deux cas :

a B_0 **et** B_1 **disjoints**

soit m dans B. Je nomme M sa composante connexe dans B. Comme B_0 et B_1 sont compacts (images de B par T_0 et T_1), si m est dans B_{i_1} (i_1 vaut 0 ou 1), M est contenue dans B_{i_1}, mais ce B_{i_1} est lui même $T_0(B_{i_1}) \bigcup T_1(B_{i_1})$, donc m est dans l'un des deux et M est contenu dans un $B_{i_1\ i_2}$. On peut continuer ainsi et M est contenu dans l'intersection d'une suite décroissante de parties dont les diamètres tendent vers 0 (chaque T_k contracte d'un rapport |b|) : M est donc réduite à un point et B est totalement discontinu

b B_0 **et** B_1 **non disjoints**

soit alors a dans B_0 et B_1, donc a=$ba_0 = b + ba_1$. B étant (métrique) compact, pour prouver qu'il est connexe il suffit de prouver que pour tout $\varepsilon > 0$ et pour tout z de B on peut trouver une ε-chaine reliant z à a (c'est une liste : $a = c_0, c_1, ...c_n = z$ avec deux c_k consécutifs distant de moins de ε)

Je nomme G le diamètre de B, et λ= |b|. Pour tout z de B on peut trouver une G-chaîne : a, z. Mais à partir des ε-chaine reliant a aux divers z on fabrique facilement des $\lambda\varepsilon$-chaines :

1. si z est dans B_0, $z = bz_0$, on ε-relie z_0 à a on multiplie par b pour aller de bz_0 à b a (donc de z à ba) en fin on on va de même de b a à ba_0
2. si z est dans B_1, $z = b + bz_1$ on fait pareil avec T_1 à la place de T_0

6 quels d(b) sont connexes ?

quand $|b| < 1/2$: les z de B sont dans B_0 ou dans $B_1 = b + B_0$. La distance bu à b + bv (où u et v sont dans B) est celle de b à b(u-v), c'est donc |b| fois celle de 1 à (u-v), or (u-v) est une somme

de puissances de b (à coefficients 1 ou 0 ou -1), son module est donc strictement inférieur à 1 (puisque $|b| < 1/2$) : on a donc B_0 et B_1 disjoints, B est totalement discontinu

Quand, pour un argument donné, |b| croît, les B_0 et B_1 "augmentent" et vont finir par se rencontrer

On imagine alors que l'ensemble des b pour lesquels d(b) est connexe va constituer un nouveau splendide ensemble "à la Mandelbrot" .... déception, on obtient une sorte d'oeuf poilu moche (il aurait peut-être fallu le microscoper ?). Dans [2] cet ensemble est étudié et on y prouve que B est connexe dès que $|b| > \sqrt{2}/2$)

Parmi les d(b) dessinés ci-dessus ? Le Tartan est connexe (c'est un rectangle rempli) cela se prouve en considérant (par exemple) les parties réelles qui forment un ensemble du même type avec b'=$(0.931i)^2$: le module est > 0.5 on a donc un intervalle. (En fait il n'y en a que deux de disconnexes : chou-fleur et squelette)

B concernant le serpent

1 re-définition du serpent

L'efficacité des preuves par point fixe est telle que l'on va décrire S aussi de cette façon. Je note r=$\frac{1+i}{2}$ (c'est le b du § précédent) c'est le complexe qui décrit la similitude d'angle $\frac{\pi}{4}$ et de rapport $\frac{\sqrt{2}}{2}$: celle qui transforme la diagonale du carré [0..1].[0..1] en [0..i]

On définit alors deux similitutes : $s_0 = z \to rz$ et $s_1 = z \to r + \bar{r}z$ et comme ci-dessus nous les étendons aux parties de $\mathbb{C}$, et concluons par

$$s(A) = s_0(A) \bigcup s_1(A)$$

Comme s est l'union de deux contractions, c'est une contraction, ses itérés pour n=1,2,3,4,5... en partant du segment [0..1] donnent justement les serpents vus au début, il y a un unique point fixe : c'est S qui vérifie donc $S = s_0(S) \bigcup s_1(S)$

2 propriétés faciles des S_n

On a donc $S_0 = I = [0..1]$ et $S_{n+1} = s_0(S_n) \bigcup s_1(S_n)$. J'affirme que tout ce qui suit est facile par récurrence (ou mieux par évidence)

1. S_n est constitué de 2^n segments, tous de longueur $\lambda_n = (\frac{\sqrt{2}}{2})^n$
2. Tout point qui était une extrémité dans S_n l'est encore dans S_{n+1}
3. Si on paramètre S_n de façon affine par morceaux par $s_n : [0..1] \to S_n$ alors $||s_n - s_{n+1}|| \leq \lambda_n/2$ ... et donc la suite s_n converge uniformément vers une $s = s_\infty$ dont l'image est $S = S_\infty$ = le serpent
4. Les directions des segments consécutifs de S_n sont orthogonales
5. Si je prend (0,0) et (1,0) comme premier segment de S_n alors toutes les extrémités de segments sont des (p_k, q_k) de $\mathbb{Z}^2$ et la parité de $p_k + q_k$ change à chaque k
6. quand deux segments perpendiculaires du parcours partagent une extrémité, "ils s'enchaînent" = l'un arrive et l'autre part; cela résulte de la réflexion de parité que l'on vient juste de voir
7. Si un M de $\mathbb{Z}^2$ intervient deux fois comme origine : comme on a même abcisse il y a eu un nombre pair de déplacements horizontaux, ... mais donc aussi un nombre pair de déplacements verticaux, donc un nombre pair de déplacement, donc les deux segments partant de M sont tous deux horizontaux ou verticaux : ou ils se tournent le dos ou ils sont confondus (on prouvera que ce n'est jamais ce denier cas)

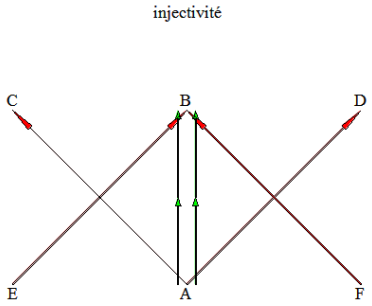
8. S_{n+1} a pour sommets ceux de S_n plus des pointes situées alternativement à droite et à gauche du chemin suivi par S_n

3 les S_n sont injectifs "sauf aux coins"

(Les idées de ce § sont de P.Charlat étudiant à Fauriel vers 1990)

Pour n=1,2,3,4... on le vérifie visuellement : il y a des coins en commun mais on n'a pas deux segments de S_n joignant les mêmes A et B

On attaque S_{n+1} et on suppose avoir deux indices différents tels que l'on aille de A à B. Les reflexions sur la parité du § précédent disent alors que c'est deux fois dans le même sens :

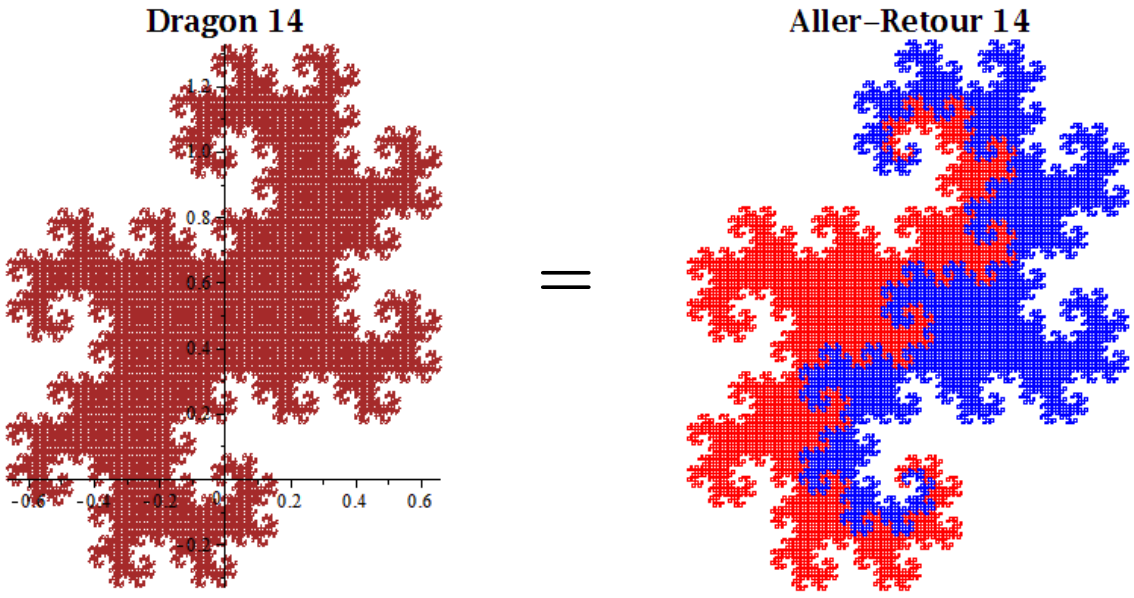


Ce A → B ne peut provenir (dans S_n) que de A → C ou E → B ou A → D ou F → B (avec C → A ce serait à l'envers), mais on ne peut avoir ni (A → C et E → B) dans S_n, ni (A → D et F → B), ni (A → C et A → D) (voir le 6. ci-dessus), ni (E → B et F → B) (voir le 6. ci-dessus), ni (A → C et B → F) (les sommets ne seraient plus dans le même quadrillage entier), ni (E → B et A → D) (les sommets ne seraient plus dans le même quadrillage entier) : nos "deux A → B) proviennent donc du même segment : c'est exclu par l'injectivité supposée de S_n

4 S est d'intérieur non vide

On peut le prouver "purement en serpent" en trouvant une grille 4×4 dans S_{10} et en en déduisant l'existence d'une grille 2 fois plus fine dans S_{12} etc .. mais comme on va prouver "S+S=D" et comme on sait que les dragons pavent (dénombrablement) le plan, S ne peut être d'intérieur vide

C serpents et dragons : D = S+S



Les 14 sont des indications de programmation : D_{14} est l'ensemble des sommes de puissances au plus 14 de b, et les S ont été obtenus à partir de S_0=un segment en pliant 14 fois (comme programmation c'est la version "transducteur 14")

La grosse difficulté ici est la précision on ne veut pas traiter "un D et un S" et des "ressemblances de formes", on veut une preuve, il faut spécifier soigneusement le D et le S

Je nomme ici D = l'ensemble des sommes de puissances strictement négatives de b=$\frac{1+i}{2}$. On sait donc que $D = bD \bigcup (b + bD)$

Je nomme ici S = le serpent qui va de l'origine 0 à l'extrémité i, c'est la limite de la famille qui part du segment [0..i], qui continue par [0..(-1+i)/2][(-1+i)/2..i] ... et qui finit par vérifier $S = bS \bigcup (1 - i\bar{b}S)$

Je nomme ici S' = le serpent-retour = i - S

L'objectif est $D = S \bigcup S'$. Je nomme momentanément $\Delta = S \bigcup S'$, pour prouver D=Δ je vais prouver que Δ vérifie l'équation qui caractérise D : $\Delta = b\Delta \bigcup (b + b\Delta)$

Sur le dessin on voit que : D est constitué d'une moitié DSud = (bD) et d'une moitié DNord (b+bD); la moitié DSud est constituée des 3/4 de S et du dernier quart de S', tandis que DNord est fait de la fin de S et des 3/4 de S'

On va ci-dessous utiliser "tout ce que fait b" à savoir

$$b^2 = i/2 \quad b + bi = i \quad i - \bar{b}i = bi \quad i - bi = b \quad \bar{b}i = b \quad b\bar{b} = \frac{1}{2}$$

Au boulot !

$$b\Delta = bS \bigcup (bi - bS)$$

on remplace S par $bS \bigcup (1 - i\bar{b}S)$:

$$= b^2S \bigcup (bi - \frac{1}{2}S) \bigcup (bi - \frac{i}{2}S) \bigcup \frac{1}{2}S$$

on recopie dans $b + b\Delta$:

$$= (b + b^2S) \bigcup (b + bi - \frac{1}{2}S) \bigcup (b + bi - \frac{i}{2}S) \bigcup (b + \frac{1}{2}S)$$

On vient d'obtenir $b\Delta \bigcup (b + b\Delta)$ en 8 morceaux que je nomme D_1 ..D_4 pour la première ligne et D_5 ..D_8 pour la seconde

On attaque maintenant $\Delta = S \bigcup (i - S)$ et on y reporte $S = bS \bigcup (1 - i\bar{b}S)$ dans lequel on reporte encore $S = bS \bigcup (1 - i\bar{b}S)$

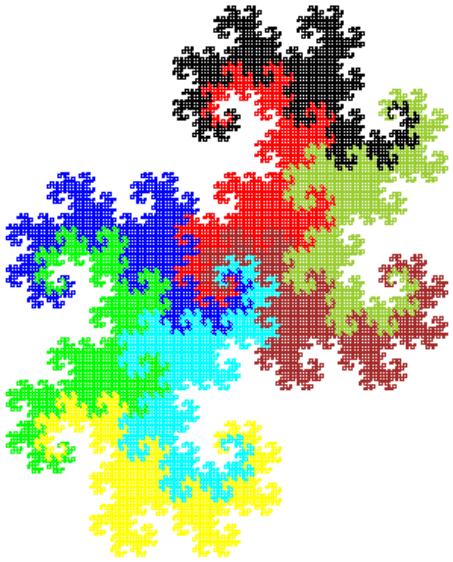
$$S = bS \bigcup (1 - i\bar{b}S)$$

$$S = b^2S \bigcup (bi - \frac{1}{2}S) \bigcup (i - \frac{1}{2}S) \bigcup (i - \bar{b}i - \frac{i}{2}S)$$

$$(i - S) = (i - b^2S) \bigcup (i - bi + \frac{1}{2}S) \bigcup (\frac{1}{2}S) \bigcup (\bar{b}i + \frac{i}{2}S)$$

On vient d'obtenir Δ en 8 morceaux que je nomme G_1 ..G_4 pour la première ligne et G_5 ..G_8 pour la seconde

On a maintenant $G_1 = D_1$, $G_2 = D_2$, $G_3 = D_6$, $G_4 = D_3$, $G_5 = D_7$, $G_6 = D_8$, $G_7 = D_4$, $G_8 = D_5$: OUF



Note 1 : pour rédiger cette preuve, j'ai voulu utiliser celle publiée dans mon premier article (Quadrature numéro 10), mais mes dessins avaient des serpents verticaux et mes calculs traitaient le Serpent allant de 0 à 1 au lieu de 0 à i

J'ai essayé à "de nombreuses reprises" d'adapter : pas moyen! Il y a en plus plein de manifestes "fautes de frappe" dans Quadrature (à l'époque l'article était écrit au stylo sur papier et l'éditeur retapait .. comme il pouvait). J'ai donc tout recommencé ... et j'ai dû m'y prendre à "de nombreuses reprises"

Note 2 : la couleur fait voir que Rouge + Bleu Foncé + Bleu Clair + Marron constitue une nouvelle forme d'ensemble paveur

III Les programmations

A versions caml

1 S

J'ai écrit dans des répertoires caml des "bibliothèques" qui font les tâches courantes : ci-dessous j'en charge deux. Une fait les opérations concernant le graphisme avec "mes choix" : le centre de l'écran

est (0,0), on représentera les points de $[-10..10] \times [-10..10]$, etc. L'autre effectue les opérations usuelles concernant les complexes.

La programmation traduit simplement les définitions de "Serpent transducteur" : D et G sont codés comme 0 et 1 (cela me permet d'utiliser 1-x pour les échanger)

```
(* dessin des serpents *)

(*   chargement de bibliothèques   *)
#open "graphics";;

directory "D:/Info/OPTION/MODULES/graphisme";;
directory "D:/Info/OPTION/MODULES/nombres";;

#open "Ini_gr";;  load_object "Ini_gr";;
#open "complexes";; load_object "complexes";;

(* l'instruction #infix ne se transmet pas à l'analyseur de l'éditeur caml *)

#infix "aC";; (* ajoute dans C *) #infix "sC";; (* soustrait dans C *)
#infix "pC";; (* produit dans C *) #infix "dC";; (* divise dans C *)

(*   des initialisations pour un premier essai    *)
let rec Ser = function  |1->[1]
                        |n-> let old = Ser(n-1) in old@[1]@(rev (map (fun x-> (1-x)) old));;

(* On utilise les choix par défaut de module graphique : on trace dans [-10..10]^2 *)
open_graph "";; vide_ecran();; (* initialisation, mode par défaut *)

let axes()=va_en (-10.,0.);joins_a (10.,0.);va_en (0.,-10.);joins_a (0.,10.);;

(* la fonction pose est comme "premier_point", avec un rapport d'homothétie *)
let pose rapport (a,b)=premier_point (rapport*.a,rapport*.b);;

type directions = N|E|S|O;;

let td = function |N -> E|E->S|S->O|O->N;;
let tg = function |N -> O|O->S|S->E|E->N;;

let DirSuivante = function |1 -> td |0->tg;;

let PtSuivant long (x,y) d = match d with
        |N      ->      (x,y+.long)
        |E      ->      (x+.long,y)
        |S      ->      (x,y-.long)
        |O      ->      (x-.long,y);;

let sym (x,y) = (-.x,-.y);;
let trace cou Ser long (x0,y0) d0 =
```

```
        let rec  boulot liste Pt d = match liste with
                |[]        ->   ()
                |t::q      ->   let but = (PtSuivant long Pt d)
                                in joins_a but; boulot q  but ((DirSuivante t) d)
        in set_color cou ;premier_point (x0,y0);boulot Ser (x0,y0) d0
;;
(* de bons réglages sont :
        (Ser 14) 0.05 Omega E
        (Ser 16) 0.025 Omega N
        (Ser 17) 0.0125 Omega O   *)

(* Pour " 1 dragon = 2 serpents" : on définit ecart qui trace-à-l'envers *)

let ecart cou Ser long (x0,y0) d0 =
             let rec  boulot liste Pt d = match liste with
                     |[]      ->   ()
                     |t::q    ->   let but = (PtSuivant long Pt d)
                                   in joins_a (sym but); boulot q  but ((DirSuivante t) d)
            in set_color cou ;premier_point (sym (x0,y0));boulot Ser (x0,y0) d0
;;
vide_ecran();;
trace red (Ser 16) 0.0125 (0.,0.) N;;
ecart green (Ser 16) 0.0125 (-0.,-3.18) N;;
```

2 les d(b)

Un intérêt du graphisme caml est la memoire d'écran : on peut lire la couleur d'un pixel avant d'en mettre un. Le programme qui suit joue la dessus : j'ai fixé une liste de couleurs (plutôt vert pour le chou-fleur, rouge-orangé pour le flamenco). Quand on affiche sur un pixel blanc (= la première fois) on prend la première des couleurs, quand on affiche de nouveau on lit la couleur et on prend la suivante dans la liste. Cet empirisme ne permet pas de distinguer les non-injectivité vraies de celles dues aux arrondis de la pixelisation, on obtient souvent de jolis moirages

```
(* dessin des ensembles unité en base complexe *)
(* il y a 10 lignes analogues à celles du programme Serpents *)
(*   des initialisations pour un premier essai    *)
let max=10;; let alfa= {Re = 0.5 ;Im=0.5 };;
let code=make_vect max false;;
let puiss_alfa=make_vect max unC;;

(* la fonction naïve de comptage, on fait mieux avec un code de Gray *)
let ajoutun()=
        let retenue=ref true and k=ref 0
        in        while !retenue do
                        code.(!k)<-(not code.(!k));
                        if (not code.(!k))
                        then incr k
```

```
                        else retenue:=false
                                done
;;

(* on stocke dans un vecteur les puissances de alfa *)
let calcule_puissances=
        for k=1 to (max-1) do puiss_alfa.(k)<- alfa pC puiss_alfa.(k-1) done;;

(* le "produit scalaire" du vecteur de booléens et de celui des puissances de alfa donne un
let numerise vecteur=
        let somme= ref zeroC in
        for k=0 to (max-1) do if vecteur.(k) then somme := !somme aC puiss_alfa.(k); done;
!somme ;;

(* On utilise les choix par défaut de module graphique: on va tracer dans [-10..10]^2
open_graph "";; vide_ecran();;          (* initialisation, mode par défaut *)

let axes()=va_en (-10.,0.);joins_a (10.,0.);va_en (0.,-10.);joins_a (0.,10.);;

(* la fonction pose est comme "premier_point", avec un rapport d'homothétie *)
let pose rapport (a,b)=premier_point (rapport*.a,rapport*.b);;

(*  au boulot! noir et blanc
axes();;
while code.(max-1)=false
do
        let {Re=u;Im=v}=(numerise code) in (pose 3. (u,v));
        ajoutun();
done;; *)

(*  au boulot! couleur *)
let couleurs=[black;blue;cyan;green;yellow;red;white;black];;

let rec suivant mot =function
        |[]                             ->        failwith "catastrophe impossible"
        |a::t::q        when (a=mot)    ->        t
        |t::q                           ->        suivant mot q;;

(* ci-dessous la version à petits points *)
let pose_petit_couleur rapport (a,b)=
        let         ea=virtuel_a_ecran_x (rapport*.a)
        and        eb=virtuel_a_ecran_y (rapport*.b)
        in
        set_color (suivant (point_color ea eb ) couleurs);
        premier_point (rapport*.a,rapport*.b)
;;
```

```
(* ci-dessous la version à gros points *)
let pose_gros_couleur rapport (a,b)=
        let         ea=virtuel_a_ecran_x (rapport*.a)
        and        eb=virtuel_a_ecran_y (rapport*.b)
        in
        begin
                set_color (suivant (point_color ea eb ) couleurs);
                premier_point (rapport*.a,rapport*.b);
                set_color (suivant (point_color (ea+1) eb ) couleurs);
                premier_point (rapport*.a,rapport*.b);
                set_color (suivant (point_color ea (eb+1) ) couleurs);
                premier_point (rapport*.a,rapport*.b);
                set_color (suivant (point_color (ea-1) eb ) couleurs);
                premier_point (rapport*.a,rapport*.b);
                set_color (suivant (point_color ea (eb-1) ) couleurs);
                premier_point (rapport*.a,rapport*.b)
        end;;

vide_ecran();;
while code.(max-1)=false
        do
        let {Re=u;Im=v}=(numerise code) in (pose_petit_couleur 3. (u,v));
        ajoutun();
done;;
```

B versions Maple

Elles sont collées à la fin de ce document (merci à `pdfsam` !)

Ce sont des versions entre Maple 12 et Maple 14 (mais je ne crois pas avoir utilisé de commandes spécifiques postérieures à Maple 6

IV Divers

A d'où vient l'idée de D = S+S ?

Quand j'ai écrit l'article sur les dragons (pour l'American Mathematical Monthly) j'ai mentionné au referee que je trouvais que le "Dragon de Knuth" ressemblait au Serpent. Le referee m'a répondu "je vais demander à un spécialiste des fractals" .. et 10 jours après j'avais un mail " ... il a dit que ça n'avait aucun rapport, que c'est une idée ridicule ...".

Cette réponse m'ayant bien énervé, je me suis accroché pour essayer de trouver quelque chose ... un jour j'ai raté une impression de Serpent, j'en ai fait une autre : j'ai eu les deux feuilles en main "presque superposées", l'idée de l'emboîtement était là, une fois emboîtés l'idée du Dragon était renforcée ... il n'y avait plus qu'à trouver comment faire la preuve

B et si on remplace $\mathbb{C}$...

par une autre algèbre que va-t-on avoir?

J'avais essayé en 1889 avec $(a, b) \bullet (c, d) = (ac + bd, ad + bc)$ et je n'ai rien obtenu de montrable. J'avais ré-essayé avec des "parties vectorielles de quaternions", comme ça n'est pas interne je tronquais ... ce fut encore pire

`http://www.skytopia.com/project/fractal/mandelbulb.html` est fait de ce genre d'idée : on représente $\mathbb{R}^3$ en sphériques, et on considère que les (r,θ) ET que les (r,φ) sont des représentations polaires de complexes : donc on multiplie les r, on ajoute les θ, on ajoute les φ ! Et on cherche "comme Mandelbrot" pour quels c la suite partant de 0 et définie par $M_{n+1} = M_n^2 + c$ est bornée ...

Selon moi, ça ne veut plus rien dire du tout, mais c'est d'une beauté qui écrase tout

C et si on remplace $\{0, 1\}$ par un autre ensemble de chiffres ?

Des essais (pas terribles) furent faits par moi (+ un lecteur du journal Lyonnais Singularité) avec $\{0, 1, 2\}$ et un b d'argument $\frac{\pi}{6}$ on avait un petit quelque-chose

C'est Gerard Rauzy qui a eu la bonne idée : associer $\{0, 1, 2\}$ à des b qui vérifient une équation à coefficients entiers de degré 3 (avec des variantes ... si vous cherchez `fractal de Rauzy` sur un moteur de recherche vous en aurez beaucoup)

D et la frontière ?

si vous cherchez `frontière du fractal de Rauzy` vous verrez que celle-là est paramétrée... C'est peut-être adaptable aux Dragons et Serpents

V Références

Je ne met ici que les références utilisées vers 1990. La excellente est maintenant de chercher sur internet

[1] Michel Zinsmeister *Problème de l'E.N.S.E.T. 1988.*

[2] Michael Barnsley *Fractals everywhere.* Academic Press, 1988

[3] M.Barnsley et A.N.Harrington *A Mandelbrot set for pairs of linear maps.* Physica 15D (1985) 421-432

[4] J.Dieudonné *Fondements de l'analyse moderne, Vol 1.* Gauthier Villars, 1965

[5] J.E. Hutchinson *Fractals and self-similarity.*Indiana Univ. Math J. 30 (1981) 713-747

[6] D.E.Knuth *The Art of Computer Programming, Vol 2, Seminumerical Algorithms.* Addison-Wesley, 1981, 190 and 563

La seconde partie de cet article (les d(b)) est :

`Daniel Goffinet`

`The American Mathematical Monthly, Vol. 98, No. 3 (Mar., 1991), pp. 249-255`

La partie "Serpents et Dragons" est parue dans Quadrature numéro 10 (mais tous les détails étaient faux)

---

pour toute question : `daniel.goffinet@wanadoo.fr`

Serpents et Dragons

Le Serpent seul

> On définit la la liste V des virages (1 à droite, 0 à gauche). Pour cela on initialise V(1), puis on définit V(n) en fonction de V(n-1) ... selon le modèle du pliage répété de rubans : c'est cela que fait la fonction suit

```
> restart;
> V(1) := Array(1..1) : V(1)[1] := 1; suit(1) := [1];
> suit :=proc(n) global V;local k, long;
                long := ArrayNumElems(V(n-1)); V(n) := Array(1..(1+2*long)); for k
      to long do V(n)[k] := V(n-1)[k]; V(n)[2*long+2-k] := 1- V(n-1)[k] od;
       V(n)[1+long] := 1; V(n) end:
> suit(2); suit(3);
```

> Ci dessous map(suit,[$3..16]): fait "tout le travail", il ne restera plus qu'à traduire les 1 et 0 en "DirectionSuivante", cela au moyen des Vec qui donnent les 4 directions et des TourneDroite ou TourneGauche

```
> map(suit, [$3..16]) :
> Origine := [0, 0]; Vec := [[1, 0], [0, 1], [-1, 0], [0,-1]];
> td :=proc(x) if x = 1 then 4 else (x-1) fi end; tg :=proc(x) if x = 4 then 1 else (x+1)
      fi end;
> DirSuivante :=proc(d, bit) if bit = 1 then td(d) else tg(d) fi end;
> DirSuivante(1, 1);
> with(plots) : with(plottools) :
```

> CodeAListeLignes transforme un code de tournage fourni par suit en un liste de lignes. Il n'y a pas de mise à l'échelle, c'est la fonction d'affichage de Maple qui fera le travail. L'affichage de Serpent 16 prend 1/4 d'heure. Jusqu'à Serpent 15 on a encore des trous blancs dans le Serpent noir

```
> CodeAListeLignes :=proc(a, d) local dep, dir, arr, lt, k;
                          dep := Origine; dir := d; arr := dep + Vec[dir]; lt := line(dep,
      arr);
    for k from 1 to ArrayNumElems(a) do dep := arr ; dir := DirSuivante(dir, a[k]); arr
       := dep + Vec[dir]; lt := line(dep, arr), lt end; lt end:
>
```

production des images

```
> CodeAListeLignes(suit(2), 1);
> display(%, scaling = constrained, axes = none, titlefont = [Times, bold, 16], scaling
      = constrained, title = "Serpent 2");
> CodeAListeLignes(suit(4)) :
> display(%, scaling = constrained, axes = none, titlefont = [Times, bold, 16], scaling
      = constrained, title = "Serpent 4");
> CodeAListeLignes(suit(8)) :
```

```
> display(%, scaling = constrained, axes = none, titlefont = [Times, bold, 16], scaling
      = constrained, title = "Serpent 8");
> # CodeAListeLignes(suit(16)) : display(%, scaling = constrained, axes = none, title
      = "Serpent 16");
```

Le Dragon seul

> L'idée de la programmation est de ne faire que des produits scalaires : pour un b donné on (pré-)calcule les puissances de b, et on fait le produit scalaire de cete liste avec "toute" les listes de 0 et de 1 pour ajouter certaines des puissances de ce b

```
> with(LinearAlgebra) : with(plottools) : with(plots) :
> b := (1 + I)/2;
> puissb := map(k→b^k, [$0..20]);
```

> prefixe est un intermédiaire technique pour écrire code(), ensuite on convertit les termes de la liste code(n) en Vector pour pouvoir utiliser la fonction toute prête DotProduct (notée par un simple .)

```
> prefixe := proc(z, liste) map(x→[z, op(x)], liste) end; prefixe(8, [1, 5, 9]);
> code := proc(n) local old; option remember; if n = 1 then [[0], [1]] else old := code(n
      − 1); [op(prefixe(0, old)), op(prefixe(1, old))] fi end:
> map(x→convert(x, Vector), code(4));
> Vcode := proc(n) map(x→convert(x, Vector), code(n)) end;
```

> CodeAComplexes fait ce qui a été dit ci-dessus, elle part de code(n) qui est une liste de 2^n Vecteurs, et elle fournit une liste de complexes. Ces complexes vont alors devenir des point quand ils seront passés par la fonction ComplexeAPoint. Enfin .. il reste à afficher

```
> CodeAComplexes := proc(n) local puiss; puiss := map(k→b^k, ⟨$1..n⟩); map(x→x
      .puiss, Vcode(n)) end:
> Dragon4 := CodeAComplexes(4);
> ComplexeAPoint := proc(z) point([ℜ(z), ℑ(z)], color = red, symbol = solidcircle,
      symbolsize = 12) end;
>
> D4 := map(ComplexeAPoint, CodeAComplexes(4)) :
> display(%, title = "Dragon 4", titlefont = [Times, bold, 16], scaling = constrained);
>
> D5 := map(ComplexeAPoint, CodeAComplexes(5)) :
> display(%, title = "Dragon 5", titlefont = [Times, bold, 16], scaling = constrained);
>
>
> D7 := map(ComplexeAPoint, CodeAComplexes(7)) :
> display(%, title = "Dragon 7", titlefont = [Times, bold, 16], scaling = constrained);
> D10 := map(ComplexeAPoint, CodeAComplexes(10)) :
> display(%);
> D12 := map(ComplexeAPoint, CodeAComplexes(12)) :
```

```
> display(%);
> D14 := map(ComplexeAPoint, CodeAComplexes(14)) : display(%, scaling = constrained)
```

Les Dragons positifs

```
> c := 1 − I;

> with(LinearAlgebra) : with(plottools) : with(plots) :
>
> puissc := map(k→c^k, [$0..7]);
```

> prefixe est un intermédiaire technique pour écrire code(), ensuite on convertit les termes de la liste code(n) en Vector pour pouvoir utiliser la fonction toute prête DotProduct (notée par un simple .)

```
> prefixe := proc(z, liste) map(x→[z, op(x)], liste) end; prefixe(8, [1, 5, 9]);
> code := proc(n) local old; option remember; if n = 1 then [[0], [1]] else old := code(n
      − 1); [op(prefixe(0, old)), op(prefixe(1, old))] fi end:
> map(x→convert(x, Vector), code(4));
> Vcode := proc(n) map(x→convert(x, Vector), code(n)) end;
```

> CodeAComplexes fait ce qui a été dit ci-dessus, elle part de code(n) qui est une liste de 2^n Vecteurs, et elle fournit une liste de complexes. Ces complexes vont alors devenir des point quand ils seront passés par la fonction ComplexeAPoint. Enfin .. il reste à afficher

```
> CodeAComplexes := proc(n) local puiss; puiss := map(k→c^k, ⟨$0..n − 1⟩); map(x→x
      .puiss, Vcode(n)) end:
> Dr3 := CodeAComplexes(4);
> ComplexeAPoint := proc(z, t) point([ℜ(z), ℑ(z)], symbolsize = t) end;
> D3 := map(x→ComplexeAPoint(x, 12), CodeAComplexes(4)) :
> display(%, scaling = constrained, title ='C[3]', tickmarks = [3, 2]);
>
> D4 := map(x→ComplexeAPoint(x, 8), CodeAComplexes(5)) :
> display(D3, D4, color = [blue, black], scaling = constrained, title ='C[4]', tickmarks = [3,
      2]);
>
> D7 := map(x→ComplexeAPoint(x, 10), CodeAComplexes(8)) :
> display(%, scaling = constrained, title ='C[7]', tickmarks = [3, 2]);
>
> D12 := map(ComplexeAPoint, CodeAComplexes(12)) :
> display(%);
> D14 := map(ComplexeAPoint, CodeAComplexes(14)) : display(%, scaling = constrained)
>
>
```

L'image pour "injectivité"

```
> with(plots) : with(plottools) :
```

```
> A := [0, 0] : B := [0, 10] : C := [-10, 10] : De := [10, 10] : E := [-10, 0] : F := [10,
      0] :
> ptA := point([op(A)], color = black, symbol = solidcircle, symbolsize = 2) :
> tA := textplot([op(A), "A"], font = [TIMES, ROMAN, 14], align = {below}) :
> ptB := point([op(B)], color = black, symbol = solidcircle, symbolsize = 2) :
> tB := textplot([op(B), "B"], font = [TIMES, ROMAN, 14], align = {above}) :
> ptC := point([op(C)], color = black, symbol = solidcircle, symbolsize = 2) :
> tC := textplot([op(C), "C"], font = [TIMES, ROMAN, 14], align = {above}) :
> ptD := point([op(De)], color = black, symbol = solidcircle, symbolsize = 2) :
> tD := textplot([op(De), "D"], font = [TIMES, ROMAN, 14], align = {above}) :
> ptE := point([op(E)], color = black, symbol = solidcircle, symbolsize = 2) :
> tE := textplot([op(E), "E"], font = [TIMES, ROMAN, 14], align = {below}) :
> ptF := point([op(F)], color = black, symbol = solidcircle, symbolsize = 2) :
> tF := textplot([op(F), "F"], font = [TIMES, ROMAN, 14], align = {below}) :
>
> fABdb := arrow([0.5, 0], [0.5, 5], 0.05, 0.3, 0.1, color = green) : fABgb := arrow([-0.5,
      0], [-0.5, 5], 0.05, 0.3, 0.1, color = green) :
  fABdh := arrow([0.5, 5], [0.5, 10], 0.05, 0.3, 0.1, color = green) : fABgh := arrow([
      -0.5, 5], [-0.5, 10], 0.05, 0.3, 0.1, color = green) :
> fAC := arrow(A, C, 0.02, 0.3, 0.1, color = red, linestyle = dot) : fAD := arrow(A, De, 0.05,
      0.3, 0.1, color = red) :
  fEB := arrow(E, B, 0.05, 0.3, 0.1, color = red) : fFB := arrow(F, B, 0.05, 0.3, 0.1, color
      = red) :
>
>
> display(ptA, tA, ptB, tB, ptC, tC, ptD, tD, ptE, tE, ptF, tF, fABdb, fABgb, fABdh, fABgh, fAC,
      fAD, fEB, fFB, scaling = constrained, view = [-10..10, -2..12], font = [TIMES,
      ROMAN, 14], title = "injectivité", axes = none);
>
```