

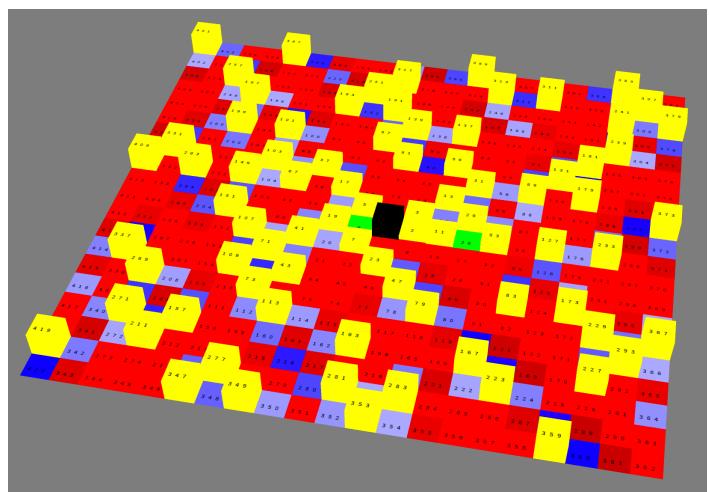
Projet info223

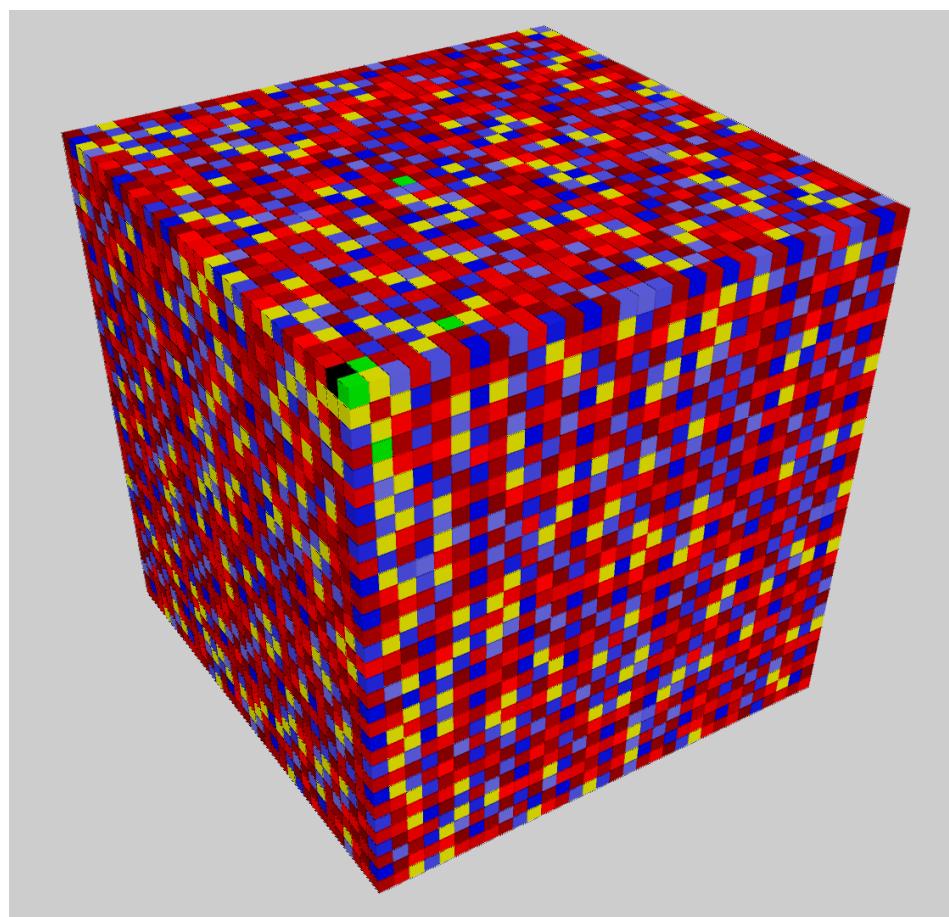
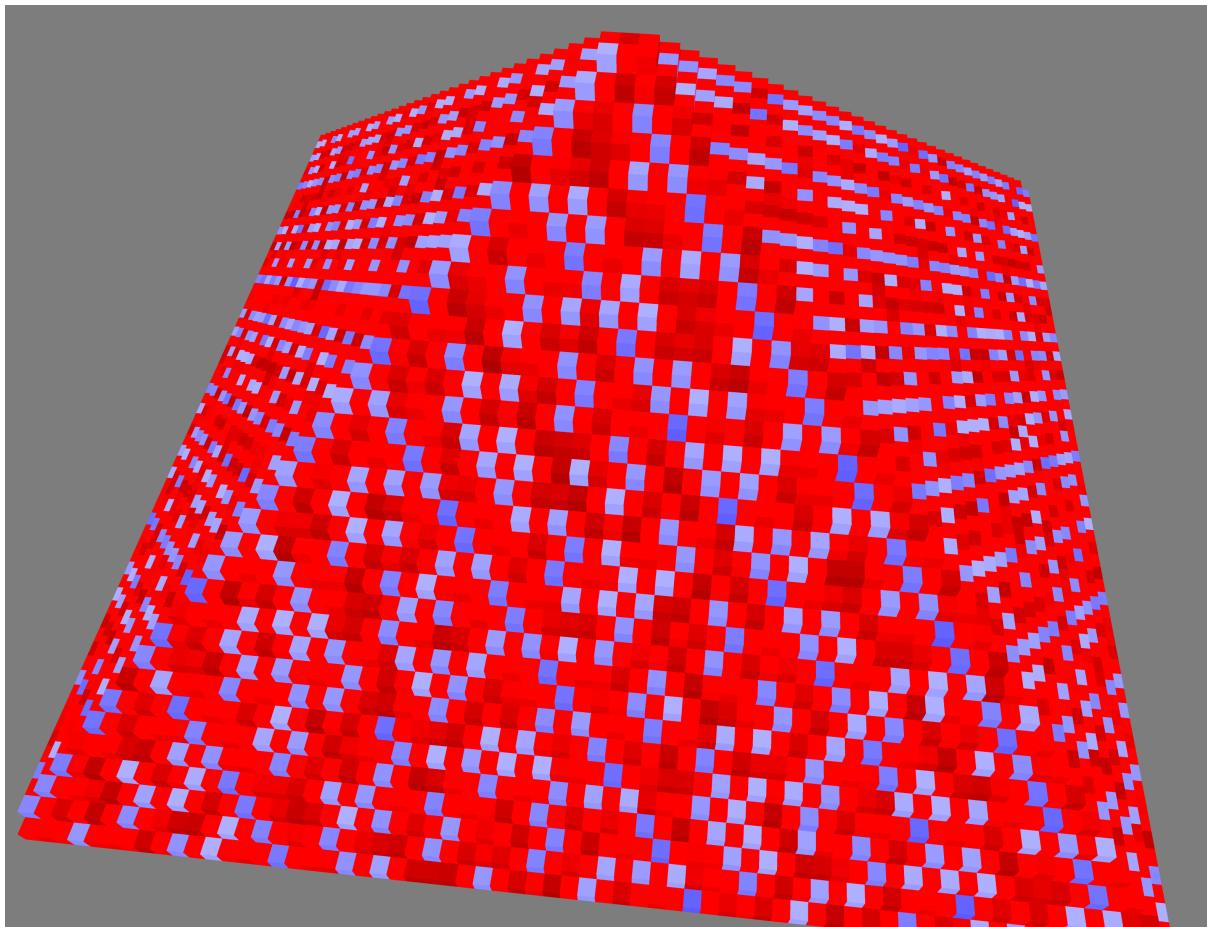
Spirale d'Ulam en 3 dimensions

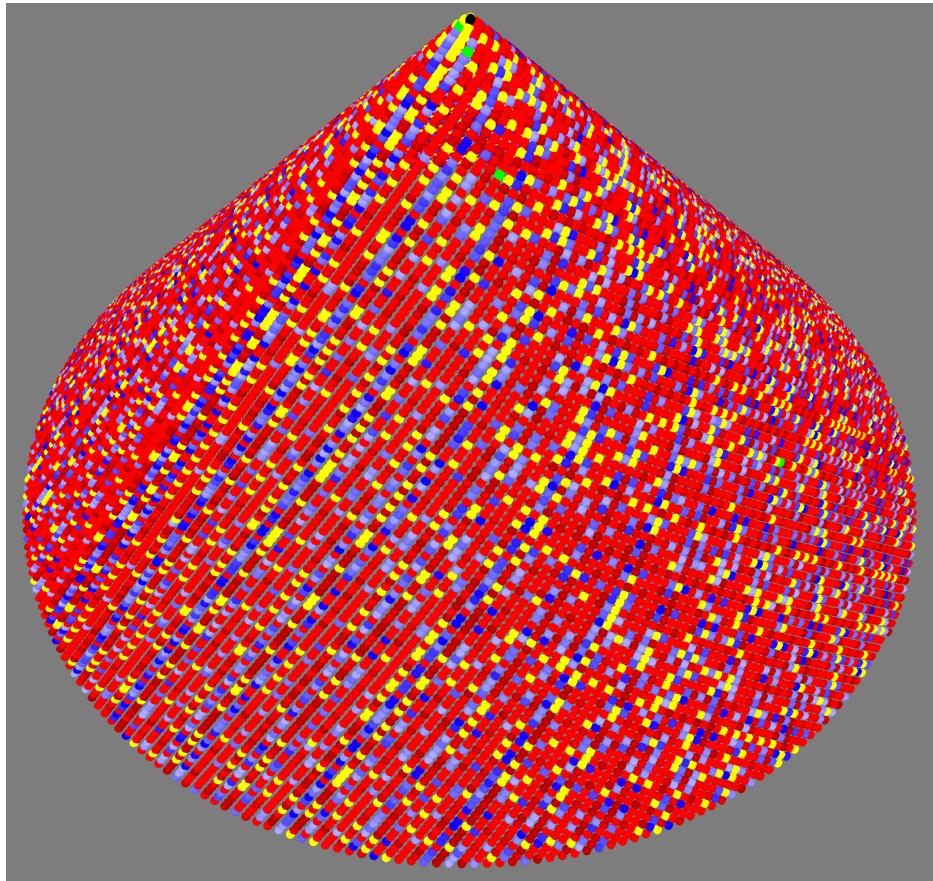
Le but de ce projet est de realiser deux mod les 3D compos es de morceaux (cubes, spheres, cylindres, etc..) dont chacun repr sente un nombre   partir de 1. Chaque morceau est plac  dans l'espace 3D par une technique que vous choisirez et impl menterez. Le morceau sera aussi colori  en fonction du nombre qu'il repr sente, par exemple un nombre premier sera colori  en jaune. Pour diff rencier un peu plus les nombres (premiers ou pas premiers) vous impl menterez d'abord une fonction `sd()` qui calcule la somme des diviseurs. Ainsi $sd(13) = 13+1 = 14$ car 13 ne se divise que par 1 et par lui m me. $sd(9)=1+9+3=12$. Ce nombre est dit d ficient car 12 est plus petit que deux fois le nombre(9) dont il est issu($<2*9$). Un autre exemple $sd(28)=1+28+14+2+7+4= 56 = 2*28$. Ce nombre est dit parfait. Un dernier exemple avec $sd(12)=1+2+4+6+12=25 >2*12$. Ce nombre est dit abondant. Vous pourrez ainsi colorier vos morceaux en fonction de la nature du nombre (premier, deficient, parfait ou abondant) ou m me faire varier la couleur en fonction de l'abondance $sd(n)-2n$ ou toute autre valeur que vous jugerez pertinente.

1 Les formes

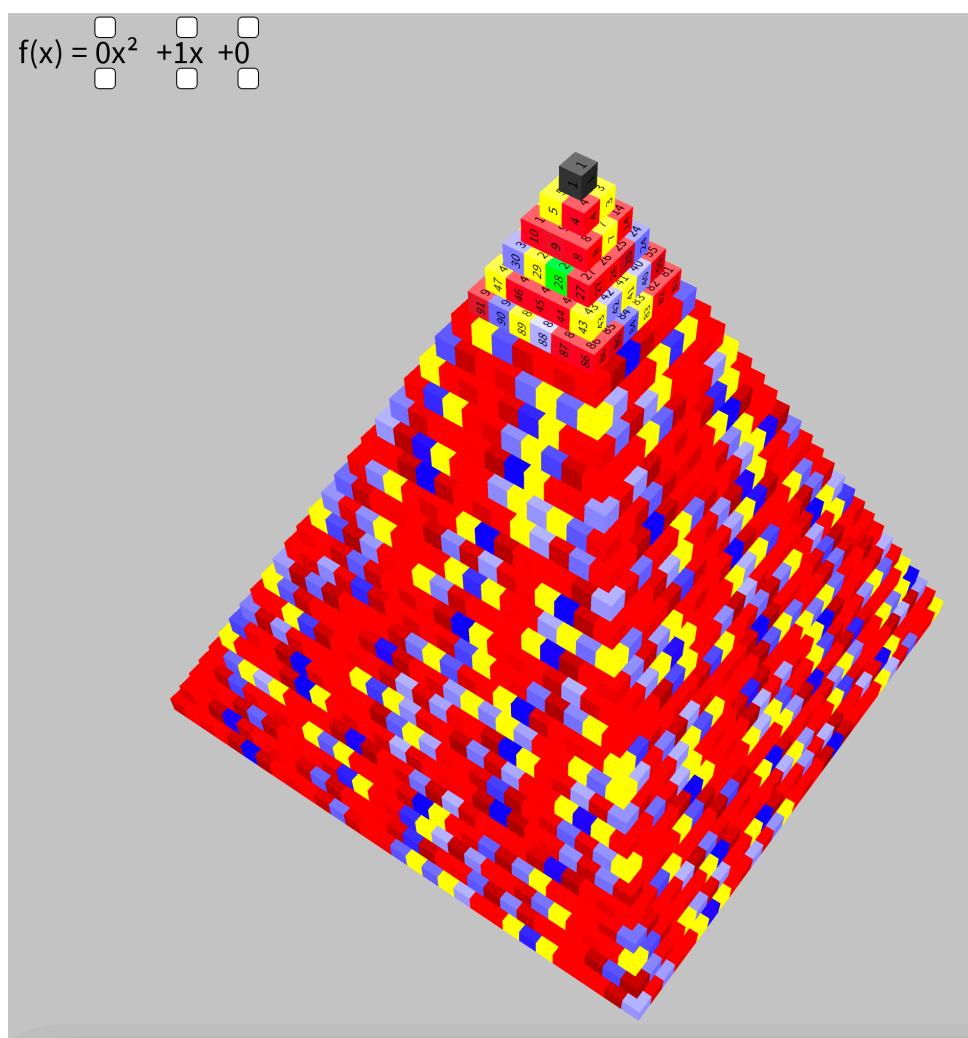
Si la spirale d'Ulam originale est en 2D (il s'agit d'une spirale carr e), de nombreuses autres formes sont possibles en 2D (spirale ronde, hexagonale, triangle) en utilisant la 3i me dimension pour montrer l'abondance ou bien des formes compl tement en 3D (cones, pyramides, cubes, etc)







$$f(x) = \frac{0}{\square} x^2 + \frac{1}{\square} x + \frac{0}{\square}$$



Si vous êtes un binômes vous devrez réaliser 2 formes différentes côte à côte. Si vous faites le projet seul vous pourrez afficher 2 fois la même forme en prenant soin de factoriser la création de la PShape dans une fonction appelée 2 fois.

Les textures

Dans la dernière pyramide ci-dessus, les cent premiers morceaux (ici des cubes) sont texturés avec une image du nombre qu'ils représentent. C'est très utile pour debugger la première partie mais aussi pour faire comprendre à l'utilisateur comment vous avez choisi d'organiser les nombres dans l'espace 3D. Vous pourrez créer une seule texture (un PGraphics) et dessiner dedans à différents endroits. Ie :

```
pg.beginDraw();
pg.text("37", 7*64, 3*32);
pg.endDraw()
```

Bien sûr vous pouvez ne pas remplir la dernière case (ie "99" en 9*64, 9*32) et donner les coordonnées de textures adéquates à chaque face de vos cubes ou sur vos sphères.

Les nombres

Plutôt que d'afficher les nombres de 1 en 1, vous utiliserez un polynôme (ie 2x affichera tous les nombres pairs) que l'utilisateur peut explorer avec 6 boutons du clavier ou 6 rectangles au dessus/au dessous de l'affichage du polynôme. Au final l'utilisateur devra pouvoir augmenter/réduire les 3 termes du polynôme. Comme le calcul peut être un peu long il est préférable de stocker les valeurs de $s(x)$ pour N valeurs de x dans un tableau de N entiers. Ce calcul se fera une seule fois dans setup() ou dans mouseClicked ou keyPressed() quand les termes du polynôme changent. Pour les "boutons" il s'agit de 6 rectangles dessinés dans draw() et de 6 tests dans mouseClicked() qui réagissent si la souris a cliqué exactement dans l'un des 6 carrés.

L'animation

Les 2 objets 3D que vous avez choisi d'implémenter devront être animés (par exemple une rotation autour du centre (du premier nombre). Vous pouvez aussi choisir de contrôler l'animation (sa vitesse, sa direction) avec des boutons du clavier ou d'autres boutons carrés à l'écran.

Le picking

Afin de permettre de tester où se trouve un nombre dans les 2 formes affichées nous souhaitons pouvoir "cliquer" sur un nombre et que ce dernier s'affiche différemment dans les 2 formes. Pour se faire vous donnerez une valeur en attribut sur les vertex qui le représentent mypshape.attrib("idnum", (float)nn);

Il est préférable de passer un flottant pour des raisons de performances mais bien sûr il représente toujours un entier. Vous créerez ensuite un shader qui déclare cet attribut pour chaque vertex et le met en interpolation entre chaque vertex. Si tous les sommets/vertices d'un cube/sphère ont le même nombre l'interpolation au niveau du fragment/pixel donnera forcément le même nombre. Vous passerez par ailleurs une valeur uniforme avec myshader.set("idselect", 28) et vous déclarerez une variable uniforme (ici idselect) dans votre shader. Les valeurs uniformes étant uniformes sur tous les vertex pas besoin de les interpoler. On peut récupérer la valeur directement dans le fragment. Il ne reste plus qu'à tester les 2 nombres avec une petite tolérance (par exemple 0.1) et afficher une couleur spéciale pour les pixels en question. Ainsi il n'y a pas besoin de refaire le modèle quand le nombre cliqué change.

Pour savoir quel nombre est cliqué il faut faire un autre shader plus malin. Ce shader affiche une couleur en fonction du nombre (passé en attribut). Il faut diviser le nombre en trois parties avec la fonction modulo :

```
N0 = mod(n, 256),  
N1 = mod ((n-N0) / 256), 256);  
N2 = mod ((n-N0-N1*256) / 256*256), 256);
```

Sachant que l'on pourra reconstruire $n = N0 + N1 * 256 + N2 * 256 * 256$ il suffit de donner comme r, g et b des pixels fragments $N0/255.0$, $N1/255.0$ et $N2/255.0$

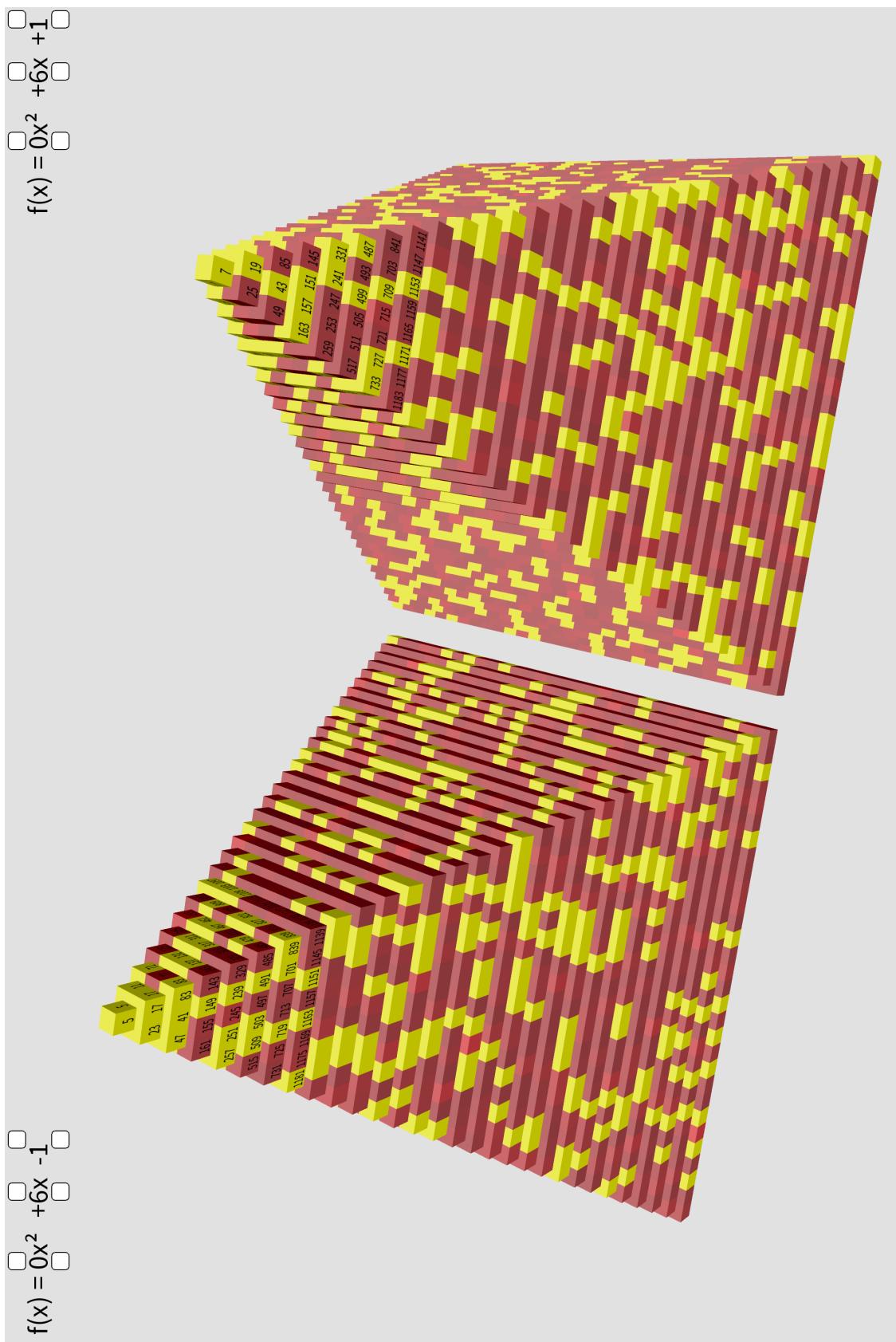
Ainsi dans la fonction mouseCliquée() il faudra créer un PGraphics pour la 3D et tout afficher dedans avec le nouveau shader

```
g1 = createGraphics(1540, 1560, P3D);  
g1.loadPixels();  
g1.beginDraw();  
    // préparation du dessin ici (translate, rotate, etc  
    g1.shader(nouveauShader);  
    // il faudra peut-être recréer les modèles ici  
    g1.shape(monModèle3D);  
    g1.resetShader();  
g1.endDraw();
```

Cette technique n'affiche rien à l'écran mais dessine dans une image où il est ensuite possible de "lire" la couleur des pixels, par exemple sous la souris sous la souris avec

```
int p = g1.get(mouseX, mouseY);
```

Et avec les blue(p), green(p) et red(p) qui par magie sont entre 0 et 255 on peut recréer le nombre n qui a été cliqué avec la souris.



La version finale d'un étudiant seul (sinon il y aurait 2 formes différentes) qui compare les polynômes $6x+1$ et $6x-1$