Projet IGSD

Rui CESISTA, Rayan BENTOUATI

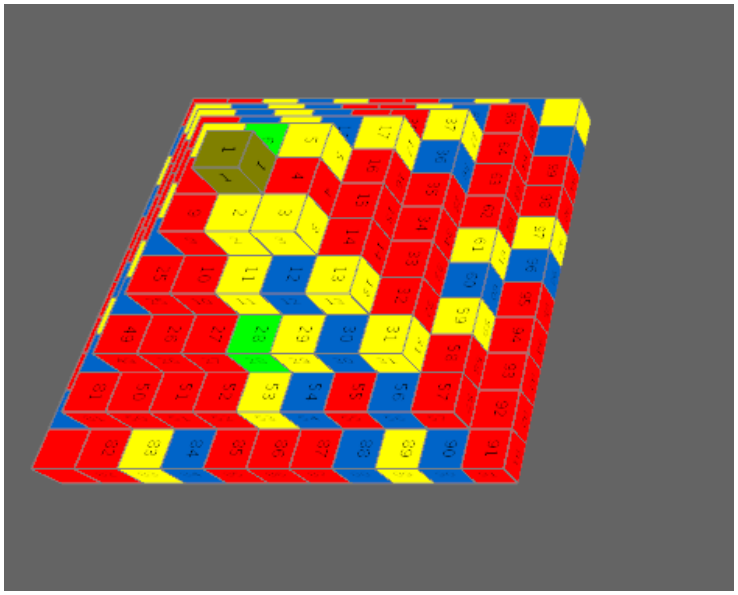
Dans ce rapport, seront décrites les éléments qui ont été implémentés, le code ainsi qu'une description de ce dernier, le tout accompagnés d'une illustration pour mieux comprendre.

Nombres premiers, abondant, parfaits .(Rayan)

Tout d'abord, nous avons implémenté une fonction somme directe, pour nous permettre de vérifier si un nombre est premier, abondant, parfait ou déficient.

```
int sd(int n)
{
  int somme = 0;
  for(int i = 1; i <= sqrt(n); i++)
  {
    if(n % i == 0)
    {
      if(n/i == i)
      {
        somme += i;
      }
      else
      {
        somme += i;
        somme += (n/i);
      }
    }
  }
  return somme;
}
boolean estNeg(int n){
  return n <= 1;
}

boolean estAbondant(int n)
{
  return (sd(n) > 2*n);
}

boolean estPremier(int n)
{
  return sd(n) == n+1;
}
boolean estParfait(int n)
{
  return sd(n) == 2*n;
}
boolean estDefaillant(int n){
  return sd(n) < 2 * n;
}
```

Avec ces fonctions, est nos variations de couleur, les nombres parfaits apparaissent vert, les premiers apparaissent jaune, les abondants bleus, les déficients rouges, et les négatifs et le 1 apparaissent jaune foncé.



Figures :

Deux figures différentes ont été réalisées pour ce projet. La première étant une pyramide, la seconde étant un cône. Ces deux figures sont composées de cubes car cela était plus pratique pour l'affichage des nombres et le placement du cube sélectionné.

La pyramide a été réalisée en suivant le modèle de la spirale d'Ulam pour l'affichage des nombres, voici par exemple le pattern utilisé pour la réalisation de la pyramide (Rayan):

```
for(int i = 1; i<boucle1; i++){
  if(i == boucle1-1)a--;
  for(int j = 0; j < a ;j++){
    ps = myBoxG1(20, n);
    if(dir == 0)translate(20,0);
    if(dir == 1)translate(0,-20);
    if(dir == 2)translate(-20,0);
    if(dir == 3)translate(0,20);
    if(k == 0 && a%2 ==1 && j == a-1)translate(0,0,-20);
    c = couleur1(nbfig1[n-1]);
    if(n < 100){
        shader(norm);
        ps.setFill(c);
        shape(ps);
        resetShader();
        psh = myBoxG1(20,1);
        shader(text);
        txtImg.beginDraw();
        txtImg.clear();
        s = "" + nbfig2[n-1];
        txtImg.background(c, 0);
        txtImg.textAlign(CENTER);
        txtImg.fill(0);
        txtImg.textSize(15);
        txtImg.text(s,20,20);
        txtImg.endDraw();
        psh.setTexture(txtImg);
        shape(psh);
        resetShader();
    }else{
      shader(norm);
      ps.setFill(c);
      shape(ps);
      resetShader();
    }
    n++;
  }
  k++;
  if(k ==2){
    a++;
    k = 0;
  }
  if(dir == 3){
     dir = 0;
   }else{
      dir++;
    }
```

Au niveau du cône on a décidé de le faire en couche avec les cos et les sin (Rui).

Modification des valeurs :

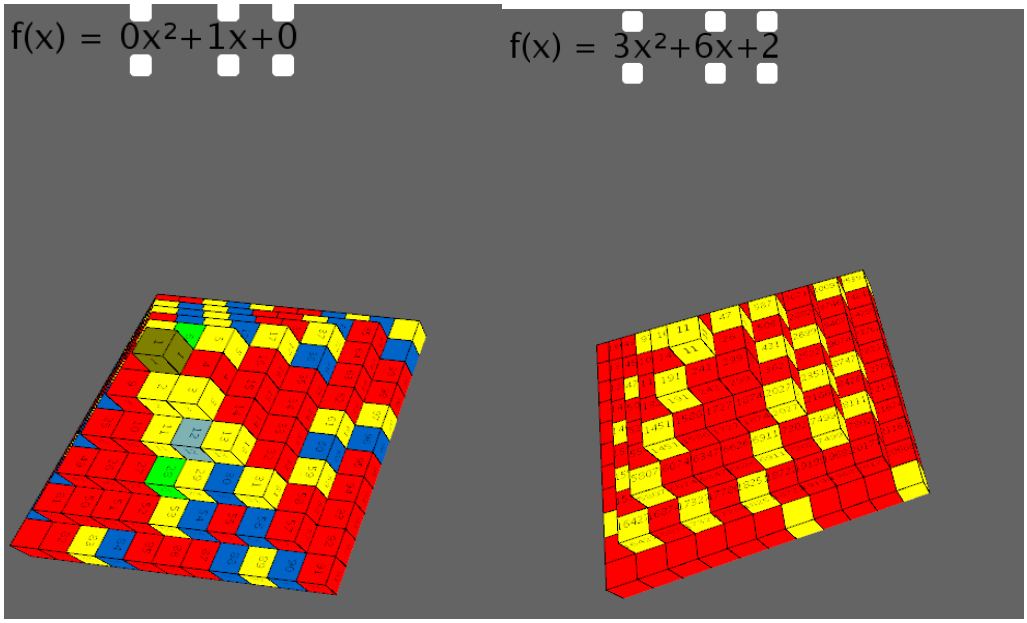
Pour la modification des valeurs des figures, deux polynômes ont été mis en place avec des boutons permettant de faire varier les nombres (Rayan):

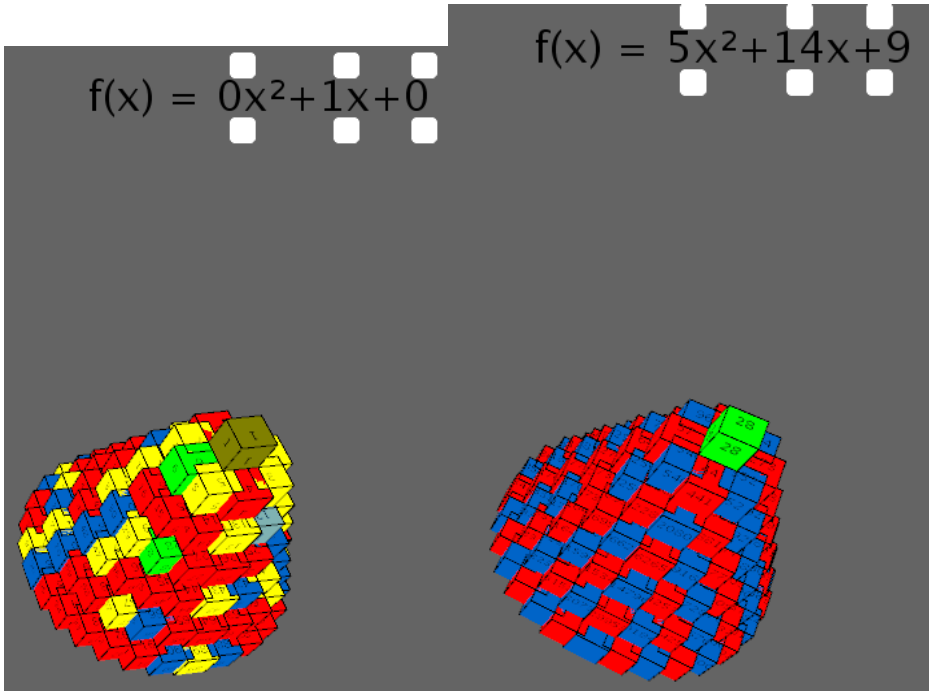
```
void drawF(){
    fill(100);
    noStroke();
    rect(0,0,width,80);
    fill(0);
    text("f(x) = ", 10, 50);
    text(n1+"x²+"+n2+"x+"+n3, 110, 50);
    fill(0);

    text("f(x) = ", width-320+10, 50);
    text(n4+"x²+"+n5+"x+"+n6, width-320+110, 50);
    //affiche carrés
    fill(255);
    rect(120,5,20,20,5);
    rect(120,55,20,20,5);
    rect(200,5,20,20,5);
    rect(200,55,20,20,5);
    rect(250,5,20,20,5);
    rect(250,55,20,20,5);
    rect(width-200,5,20,20,5);
    rect(width-200,55,20,20,5);
    rect(width-120,5,20,20,5);
    rect(width-120,55,20,20,5);
    rect(width-60,5,20,20,5);
    rect(width-60,55,20,20,5);
}
```

Affichage du polynôme(Rayan et Rui)

```
void mouseClicked()
{
  if(mouseX >= 120 && mouseX < 120+20 && mouseY >= 5 && mouseY < 5+20)
  {
    n1++;
    for(int i = 0; i < N1; i++){
      nbfigl[i] += (i + 1) * (i +1);

    }
  }
  if(mouseX >= 120 && mouseX < 120+20 && mouseY >= 55 && mouseY < 55+20)
  {
    n1--;
    for(int i = 0; i < N1; i++){
      nbfigl[i] -= (i + 1) * (i +1);
    }
  }
  if(mouseX >= 200 && mouseX < 200+20 && mouseY >= 5 && mouseY < 5+20)
  {
    n2++;
    for(int i = 0; i < N1; i++){
      nbfigl[i] += i + 1;
    }
  }
  if(mouseX >= 200 && mouseX < 200+20 && mouseY >= 55 && mouseY < 55+20)
  {
    n2--;
    for(int i = 0; i < N1; i++){
      nbfigl[i] -= (i + 1);
    }
  }
```

Extrait de la fonction pour l'affichage des boutons.





Rendu des nombres en fonction de la valeur des polynômes(Rui).

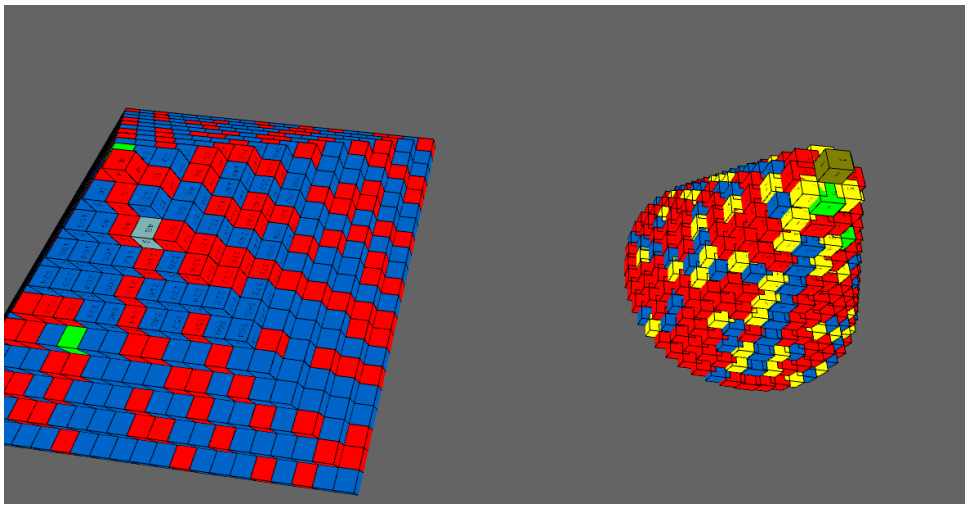
Pour de la rapidité on a fait 2 tableaux d'entier pour avoir rapidement la valeur du cube sans la recalculer à chaque fois. On a fait aussi une table de Hachage pour avoir chaque couleur selon si on a déjà calculé la valeur.

Animation (Rayan):

Les deux figures effectuent une rotation sur elles-mêmes, pour cela nous avons simplement utilisé un rotate, permettant aux figures de réaliser des rotations assez fluides.

On peut arrêter de tourner ou tourner des 2 côtés. On peut aussi contrôler la vitesse de rotation.

On a aussi rajouté l'agrandissement et les rapetissements des figures.



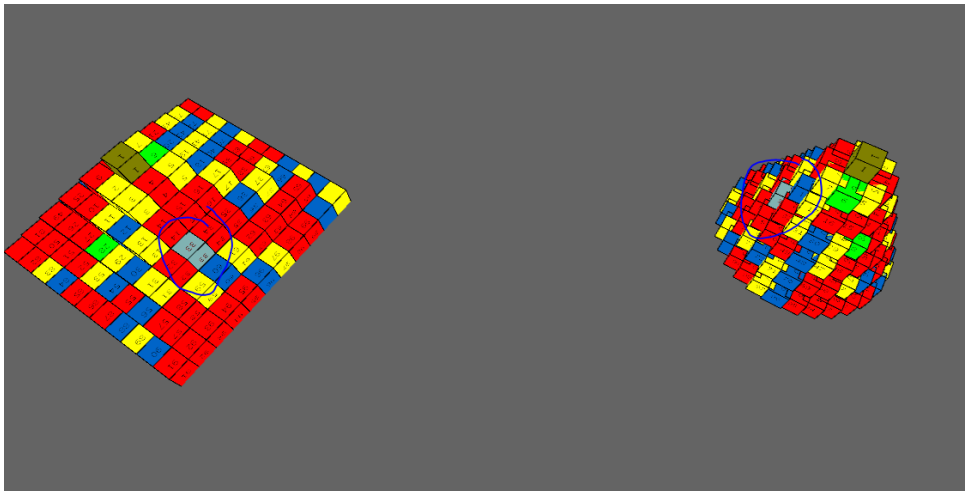
Textures (Rui):

Pour les textures on a décidé mettre dans un cube par-dessus le premier cube déjà fait. La raison est qu'on n'a pas réussi à faire le picking des cubes avec des textures à l'intérieur.

Picking (Rui):

On a fait 3 shaders dont un pour le picking un pour les cubes et le dernier pour les textures.

Au niveau du picking on a fait un PGraphics dans la fonction mouseClicked() pour avoir un cube choisi ou l'enlever. Et on le voit au niveau du shader pour les cubes



On peut voir dans le PGraphics que pour identifier chaque cube on les mit toutes de couleur différente et on a aussi un système de picking dans cette partie pour tester. Pour ce faire on a rajouté un attribue idnum pour chaque Pshape/Cube pour qu'ils puissent être répartie de couleur différente dans le shader. Et lorsqu'on clique sur un des cubes il change un attribut uniform de shader des cubes qui idselect = idnum clické pour qu'il puisse reconnaître lequel mettre en cyan.