Visualisation de graph

Auteur : Hobean BAE
Contact : b845548@gmail.com

I. Introduction

L'objectif de ce projet est de developper un programme qui permet de visualiser le graph dans l'espace 2D en C++. L'idée est de simuler un environnement physique qui influence le mouvement des noeuds,. A la fin, la position des noeuds doit s'equilibrer d'une façon lisible

II. Développement

Le fonctionnement à réaliser :
1. construction du graphe avec la bibliothéque Boost
La quantité de noeuds du graphe qu'on visualise est limité entre 50 - 100 noeuds. La construction se fait avec adjacency list et graph non-orienté

2. distribution des coordonées aléatoires
La coordonnée initiale de chaque noeud est la valeur entre 0 - taille d'ecran.

3. visualisation des noeuds avec la bibliothéque OpenGL
On utilise l'example de GL4Dummies pour créer la fenêtre graphique

4. détection de collision entre les noeuds
s'il y a une collision entre 2 noeuds, on rebondi aléatoirement l'un ou l'autre vers la distance entre deux sommets

5. détection d'intersection entre les arréts
s'il y a moins de intersection, le facteur de progression diminue, si non il augumente. on pouse aléatoirement l'un des sommets.

6. attraction entre un noeud et ses adjacents
on ajoute pour chaque sommet la moyenne de la distance entre chaque sommets et ses adjacents

7. répulsion entre les noeuds non-rélié
on ajoute pour chaque sommet une petite valeur si un noeud n'est pas rélié à un autre

8. gravité pour amener l'ensemble de noeuds vers une certaine direction
Même si la fonction 6,7 marche bien, on est dans un espace limité, veut dire que le noeud peut aller au frontier de la fenetre en se deplaçant Ca ne fait pas un beau rendu. Pour cela, on definit une gravité vers le centre, la fonction va ramner l'ensemble de noeuds vers le centre

9. system temperature (densité) qui régularise les coordonnées de noeuds
Il faut créer une matrice qui divise la fenêtre par certaine taille, chaque zone contient un ensemble de noeuds. La fonction verifie s'il y a beaucoup de noeud dans un zone, il va sortir un noued de ce zone et deplacer dans un autre zone où il n'y a moins de noeuds. A la fin, on peut eviter de créer un endroit où il n'y a beaucoup de noeuds, chaque zone contiendra presque la même quantité de noeuds

10. interface pour tester le programme dynamiquement

III. Algorithm

```
collisionMove()
 (P)rogress >=0, <= 1
 (R)ebond = 0.3
  Pour tous les (S)ommet i,
   Pour tous les (S)ommet j,
    s'il y a une collision entre S_i et S_j
     Random
       S_i.xy -= (S_j.xy - S_i.xy) * R * P
     Soit
       S_j.xy += (S_j.xy - S_i.xy) * R * P

attractionMove()
 (P)rogress >=0, <= 1
 (C)enter = 0.01
  Pour tous les (S)ommet i,
   (M)oyenne.xy = 0
   (N)ombreAdjacent = 0
   Pour tous les (A)djacent j,
      M.xy += S_i.xy - A_j.xy
      N += 1
   S_i.xy += M.xy / N * C * P

repulsionMove()
 (P)rogress >=0, <= 1
  Pour tous les (S)ommet i,
   Pour tous les (S)ommet j,
    si S_i S_j ne sont pas réliés
      si S_i.xy - S_j.xy > 0
          S_i.xy -= 0.01 * P
      Soit
          S_i.xy += 0.01 * P

gravityMove()
 (P)rogress >=0, <= 1
 (G)ravity = 0.000007
 (R)esolution.wh = largeur, hauteur
  Pour tous les (S)ommet i,
   Pour tous les (S)ommet j,
     S_i.xy -= (S_i.xy - R.wh/2) * G * P
```

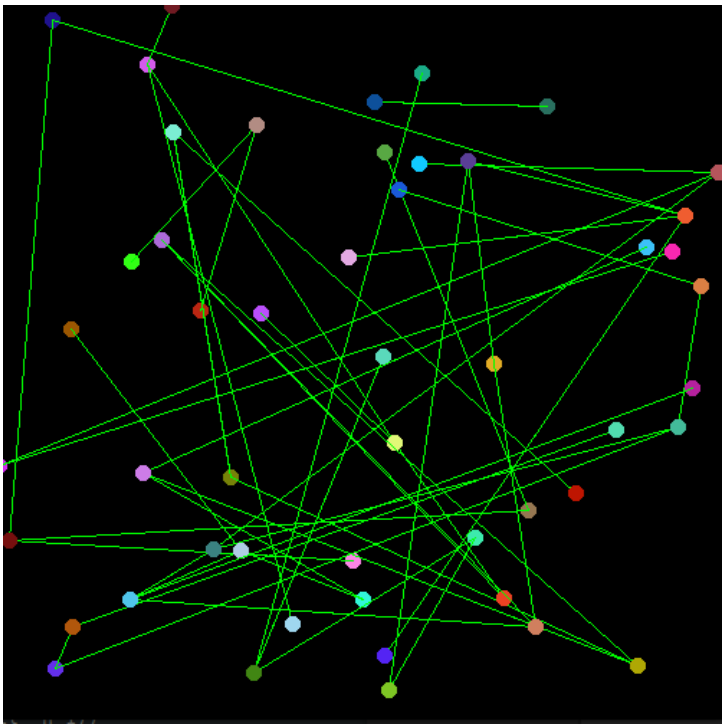
```
intersectionMove()
 (P)rogress >=0, <= 1
 (I)ntersection = 0.3
 Pour tous les (A)rrets i,
  Pour tous les (A)rrets j,
   s'il y a l'intersection entre A_i, A_j,
     Random
       A_i.S_1.xy -= A_i.S_1.xy * I * P
     Soit
       A_i.S_2.xy -= A_i.S_2.xy * I * P
     Soit
       A_j.S_1.xy -= A_j.S_1.xy * I * P
     Soit
       A_j.S_2.xy -= A_j.S_2.xy * I * P

densityMove()
 (R)esolution.wh = largeur, hauteur
 (G)rid [8][8]
    Pour tous les sommet i
      (P)osition.xy=S_i.xy/R.wh * G.wh;
      (G)rid[P.y][P.x].add(S_i);
    min.xy=G_maxSize.xy
    max.xy=G_minSize.xy
    deplacer un element de G[max.y][max.x]
                          à G[min.y][min.x]
```

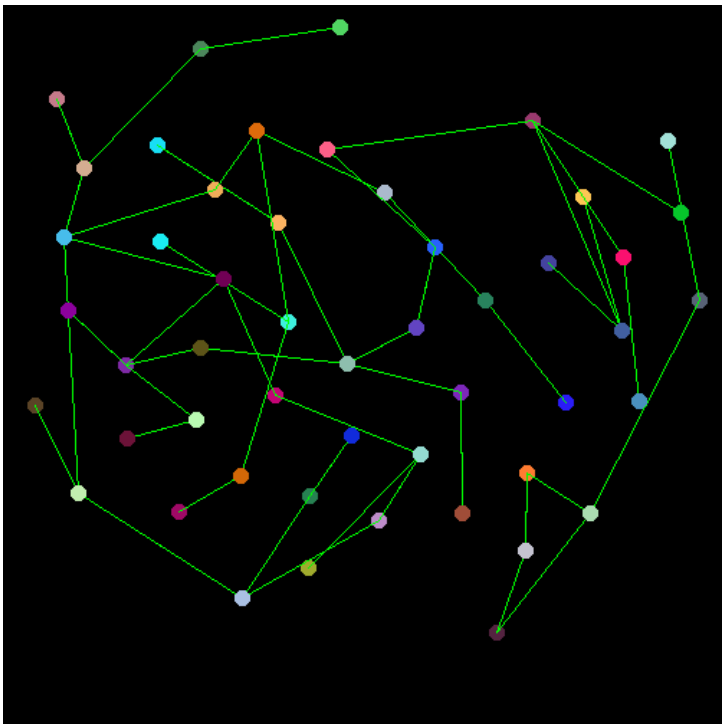
IV. Conclusion

On arrive à obtenir le résultat, les zone de noeud sont bien equilibrés. Tous les noeuds sont prés de ses adjacents. Il y a quelque intersection. Le point à ameliorer est de réduire le temps le programme depense au moin quelque dizaine secondes.

V. Résultat obtenu



(1) graph initial



(2) graph final