**TP8**
**Programmation C L2.1**
**Trier des points et visualisations**

---

Dans cette séance de travaux pratiques, nous allons coder un programme triant des points dans une fenêtre graphique et proposant quelques visualisations selon la manière choisie de trier. On utilisera la libMLV pour les parties graphiques de ce TP.

---

Nous allons travailler avec une fenêtre de taille 512 par 512 pixels et avec des tableaux de points d'au plus 256 pixels. Nous allons utiliser la structure suivante, classique pour les points. Mais aussi une structure `PointDistance` qui rassemblera un point avec une distance au carré (pour éviter de calculer des racines carrées).

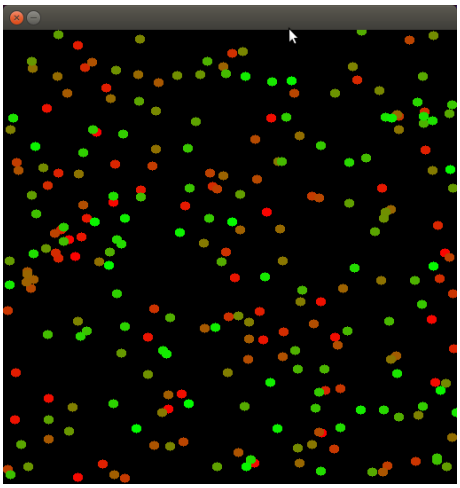
```
#define TAILLE_X 512
#define TAILLE_Y 512
#define NB_MAX_POINT 256

typedef struct{
  int x;
  int y;
}Point;

typedef struct{
  Point point;
  int distcarre;
}PointDistance;

typedef struct{
  PointDistance tab[NB_MAX_POINT];
  int nb_point;
}TabPoint;
```

1. **Fabriquer des tableaux de points aléatoires**
   Écrivez une fonction `void genererTabPoint(TabPoint* t, int nb_point)` qui remplit la structure `t` de type `TabPoint` de `nb_point` aléatoire tiré dans la fenêtre. Pour le moment, on ne calcule pas de distance.

2. **Afficher un tableau de point en faisant un dégradé**
   Voici une fonction qui retourne une couleur exploitable pour la libMLV à partir d'un entier. La fonction retourne la couleur rouge pour $i = 0$ et dégrade cette couleur vers le vert pour $i = 255$.

   ```
   MLV_Color fabriqueCouleur(int i){
     return MLV_rgba(255-i, i, 0, 255);
   }
   ```

   Écrivez une fonction `void dessineTabPoint(TabPoint* t)` qui dessine les points contenus dans `t` comme des cercles dans la fenêtre graphique de rayon 5 pixels tels que le $i^{\text{ème}}$ point de `t` ait pour couleur `fabrique_couleur(i)`.



Exemple d'affichage d'un tableau de points aléatoires.

3. **Calculer les distances**
   Écrivez une fonction `void calculeDistances(TabPoint* t, Point origine)` qui, pour chaque point de `t`, calcule le carré de la distance euclidienne du point au point `(origine)` donné en argument. Une fois calculées, les distances sont placés aux bons endroits de la structure `t`.

4. **Comparer des points**
   Écrivez une fonction `int comparePoint(const void* a, const void* b)` qui prend en argument deux adresses (qui sont en fait des adresses de structure de type `PointDistance`) et retourne un entier qui est :

   - positif si `a` pointe vers un point qui a une plus grande distance que le point pointé par `b`,
   - nul si `a` et `b` pointent vers des points qui ont une distance égale,
   - négatif si `a` pointe vers un point qui a une plus petite distance que le point pointé par `b`.

   Votre fonction de comparaison doit commencer comme il suit... C'est à vous de trouver le transtypage (cast) adéquat permettant de récupérer les points.

```
int comparePoint(const void* a, const void* b){
  PointDistance p = *( (Bon transtypage) a);
  PointDistance q = *( (Bon transtypage) b);
  ...
  ...
}
```

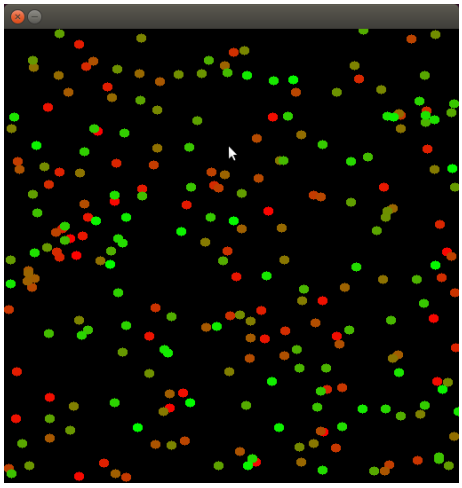
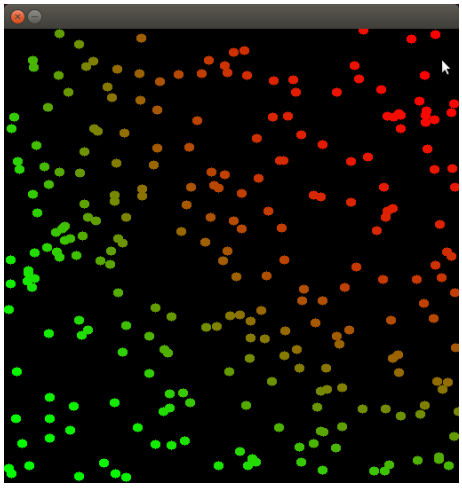
5. **Trier les points selon leur distance**
   En utilisant la fonction `qsort` qui prend pour dernier argument une fonction de comparaison (il suffit de donner à `qsort` le nom de la fonction de comparaison que vous avez programmé), trier les éléments d'un tableau de type `TabPoint` selon leur distance (second champ de la structure).

   Expliquez pourquoi la fonction de trie avait un prototype aussi compliqué avec des adresses `const void*` (génériques) alors que l'on veut juste trier des points.

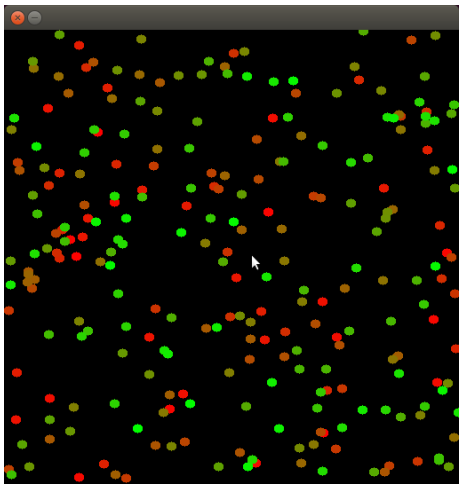
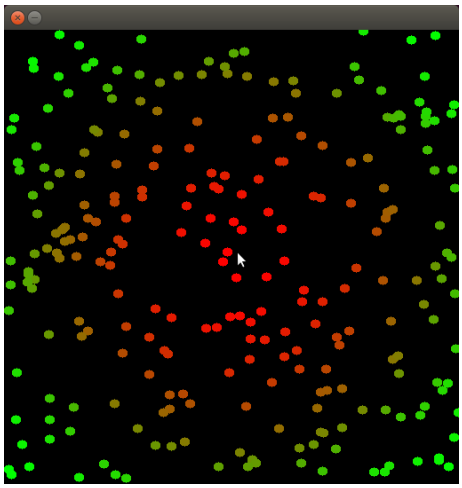
6. **Un programme complet de coloration de points selon la distance**
   Faire un programme qui génère un tableau aléatoire de type `TabPoint` contenant 256 points. Votre programme affiche ce tableau et attend un clic de l'utilisateur. Le clic de l'utilisateur va

définir un point sur la fenêtre, recalculez toutes les distances par rapport à ce point et trier les points selon leur distance. Refaire ensuite un nouvel affichage.

Voici un exemple en cliquant en haut et à droite de la fenêtre:

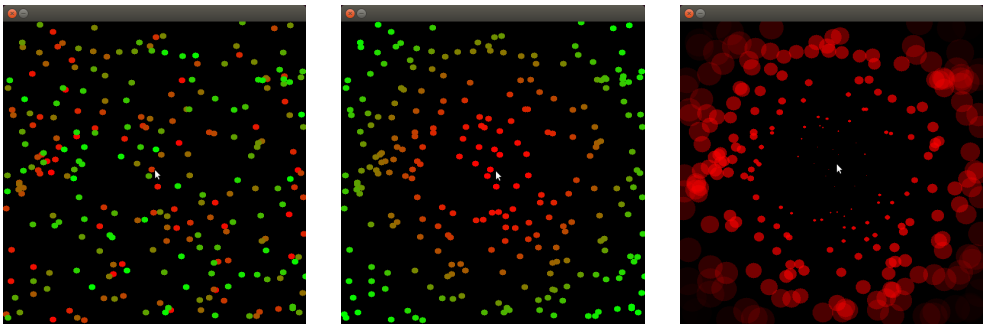
Voici un autre exemple en cliquant au milieu :

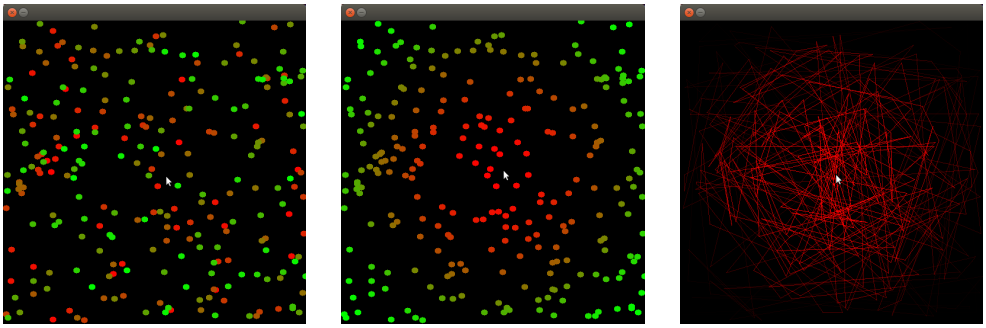
7. **Un peu plus loin avec le graphisme...**
   La fonction `MLV_rgba` fabrique une couleur avec transparence à partir de quatre entiers compris entre 0 et 255 inclus. Dans l'ordre, on lui donne le rouge, le vert, le bleu et l'indice de transparence. Définir et tester les colorations suivantes :

   - un dégradé du rouge au bleu,
   - un rouge du prononcé au effacé (on utilise ici la transparence...),
   - un mélange des deux.

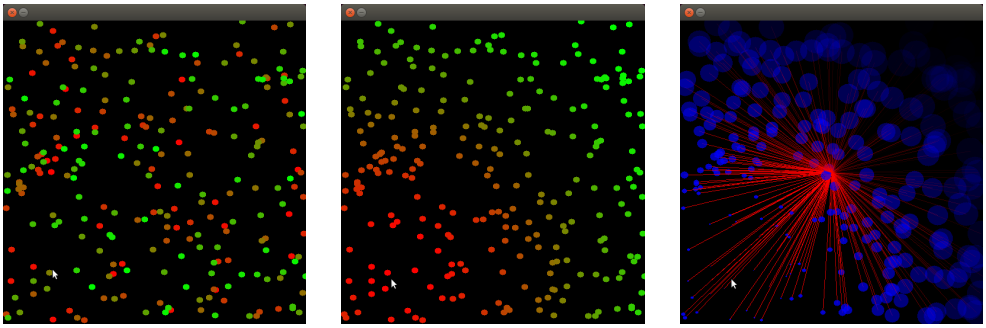
   Essayez de générer les images suivantes :



Cercles disparaissant par transparence mais de rayon croissant avec la distance...



Lignes disparaissant par transparence reliant les points consécutifs du tableau de point...



A vous de deviner...