Programmation C avancée
TP 7

Hélas ! Que ce code est sale !

Vous êtes offert d'un fichier de source en C (à télécharger sur e-learning), qui a été écrit par un ancien élève en BAC+2. Clairement, cet élève n'avait pas été entraîné pour écrire du bon code. Son code n'est ni lisible, ni modulaire, ni facile à maintenir. Ce que vous aller faire dans ce TP, c'est de lui montrer comment écrire du bon code, en le modifiant et en y ajouter des nouvelles fonctionnalités.

Pour vous aider, voici quelques détails sur ce code. Ce programme essaie d'approximer une image donnée par des triangles transparents. Il commence par lire les données de l'image dans le fichier `data.txt`. Puis, le programme applique un algorithme génétique sur des entités appelés "amibe" (*amoeba* en anglais) dans le code, chacune représent un ensemble de nombre fixé de gènes, qui sont des triangles transparents. L'algorithme génétique dans ce code prend plusieurs étapes :

1. Constituer une population d'individu aléatoires, qui forme la génération initiale.
2. Pour la génération courante, on évalue chaque individu. Puis on divise les individus en trois tiers par leur évaluation. Les tiers sont numérotés 1, 2 et 3, avec le tier 1 le meilleur et le tier 3 le pire.
3. On vire tous les individus du tier 3, et recopier les individus du tier 1 dans l'espace libéré. Puis on fait des accouplements pour chaque nouvel individu, en prenant un individu aléatoire dans le tier 2, duquel on recopie un nombre fixé de gènes aléatoires.
4. Après avoir généré tous les individus, chaque gène dans chaque individu a une chance faible de muter, en changeant ses paramètres (coordonnées des trois sommets, les composantes de couleur).
5. On revient à l'étape 2 jusqu'à avoir itérer sur un nombre de générations donné.

Le nombre de générations est entré par l'utilisateur au début de l'exécution du programme.

Le format utilisé dans le fichier `data.txt` est proche de PPM P3, qui donne d'abord la longueur h et la largeur w de l'image, puis les $h * w$ pixels, chacun représenté par trois entiers entre 0 et 255 pour les composantes de sa couleur.

Figure 1 contient un exemple du meilleur individu aux générations différentes pendant une approximation pour l'ancien logo de Firefox avec 100 triangles transparents.

Ce code a l'air intéressant, mais il y a quelques faiblesses apparentes. Le code est non modularisé, obscur et difficile à lire, et l'application n'a pas d'interface graphique, et ne peut pas lire une image de format plus utilisé. Dans ce TP, vous allez améliorer tous ces points. Voici une liste de vos tâches :

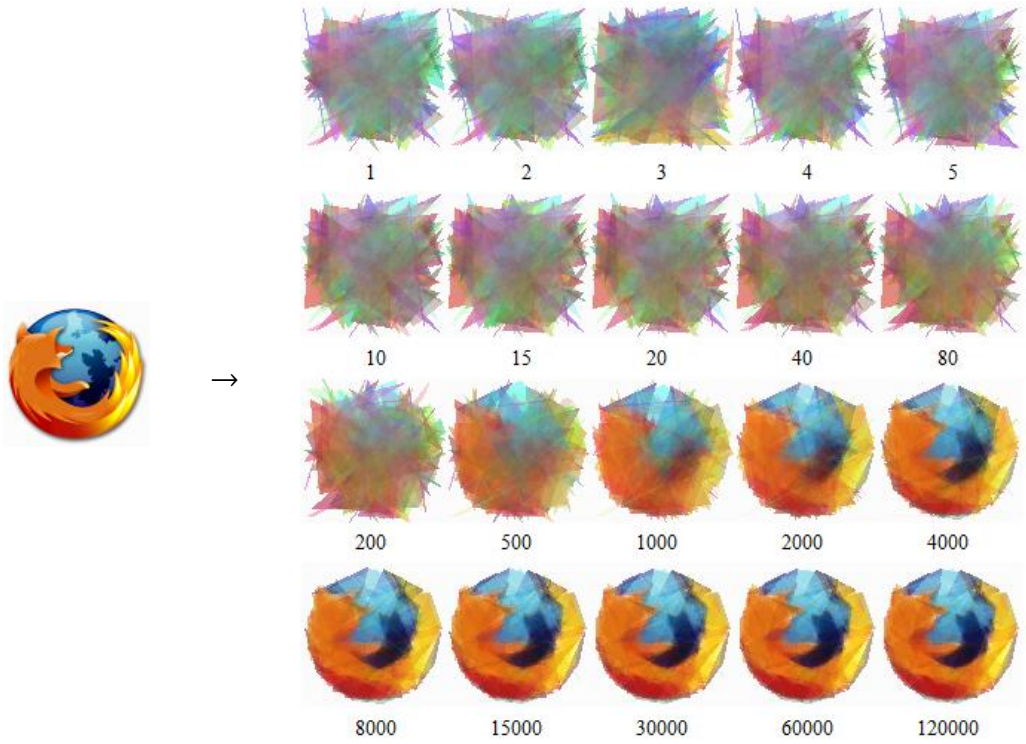

Figure 1: Exemple du meilleur individu aux générations différentes

- Séparer le code en quelques modules de façon appropriée.
- Ajouter les fonctionnalités de passage de paramètre, de chargement d'un fichier image et d'affichage par `libMLV`.
  - Dans la ligne de commande, le premier paramètre sera le nombre de générations à itérer, et le deuxième le fichier d'image à lire.
  - Le nouveau programme doit charger le fichier donné pour le calcul en utilisant `libMLV`.
  - Tous les quelques générations, le nouveau programme doit afficher le meilleur individu de la génération courante avec `libMLV`.
  - Optionnellement, vous pouvez changer le nombre de gènes (triangle) pour chaque individu en fonction de la taille de l'image. Vous pouvez inventer une formule ou bien ajouter un troisième paramètre en ligne de commande.
- Nettoyer le code pour qu'il ne pique pas aux yeux. Simplifier le code au maximum.
- Ajouter des commentaires pertinents, en mettant ce que vous avez compris pour chaque structure de données et chaque function du code.

Il n'est pas suffisant que toute fonctionnalités demandeés soient ajoutées, il faut aussi que le code soit de bonne qualité, bien modularisé et facile à lire. Allez-y, donnez une leçon sur comment écrire du bon code à ce petit élève !