Cours de programmation séquentielle

Lecture de fichiers binaires au format LAS 1.2

1 Buts

— Lecture de spécifications de format de fichier.
— Lecture de fichier binaire.
— Écriture de librairie de manipulation de points en trois dimensions.

2 Énoncé

Un fichier LAS est un format de fichier binaire utilisé pour échanger et archiver des nuages de points "lidar". Il existe différentes versions. Dans le cadre de ce travail, nous nous concentrerons sur la version 1.2 (voir la référence https://bit.ly/2QRcZEO). Un fichier LAS 1.2 est constitué de trois parties:

1. Une entête, ou en bon français, *public header block*, contenant des informations générales sur le fichier comme l'année de création du fichier, le nombre de points stockés, leur localisation dans le fichier, ...
2. Une section de longueur variable, contenant principalement des métadonnées.
3. Les points à proprement parler.

Nous allons à présent décrire brièvement l'entête et les points. La section à longueur variable n'a pas d'intérêt dans le cadre de ce travail. Dans ces sections nous allons uniquement voir quels sont les champs importants, puis détailler la méthode pour lire un fichier binaire (qui est étonnamment ou pas beaucoup plus simple que de lire un fichier texte). Finalement, il faudra créer une petite librairie de manipulations de points en 3 dimensions.

2.1 L'entête des fichiers LAS

Le contenu de l'entête est détaillé à la page 3 des spécifications https://bit.ly/2QRcZEO.

Tous les champs ne sont pas nécessaires pour la réalisation de ce travail, néanmoins il est beaucoup plus simple de tous les lire et de garder que les champs relevants. Nous n'utiliseront que les champs suivants

— `Point Data Format ID` le format de stockage des points. Cette valeur vaut 1 dans ce travail.
— `Offset to point data` qui est un entier non signé désignant l'endroit dans le fichier où sont stockés les points.

— `Number of point records` le nombres de points stockés.
— `X/Y/Z scale factor`, `X/Y/Z offset`, permettant de récupérer les coordonnées des points "géométriques" à l'aides des formules (`Xrecord` est la position de chaque point, cf. la sous-section suivante):
```
Xcoordinate = (Xrecord * Xscale) + Xoffset
Ycoordinate = (Yrecord * Yscale) + Yoffset
Zcoordinate = (Zrecord * Zscale) + Zoffset
```

Les autres champs n'ont pas forcément une utilité immédiate, mais peuvent servir à vérifier que vous n'avez pas fait d'erreur. Par exemple, `File Signature` contient forcément les lettre `LASF`, `Min/Max X/Y/Z` sont les valeur minimales/maximales coordonnées dans les directions `X`, `Y`, et `Z`, et elles peuvent servir à bien vérifier que les transformations ci-dessus sont cohérentes, ...

N'hésitez pas à utiliser ces valeurs pour faire des tests unitaires.

2.2 Les points lidar

Le format de points, version 1, est décrit à la page 9 des spécifications (voir https://bit.ly/2QRcZEO).

Les seules valeurs que nous réutiliseront ici sont les positions `X`, `Y`, et `Z` des points, qui sont des entiers signés de 4 octets et qui correspondent à `X/Y/Zrecord` de la section précédente.

Ces points "lidar" seront transformés en points "géométriques" en trois dimensions à l'aide d'une structure `point_3d` contenant les doubles `x`, `y`, et `z` (voir plus loin dans ce document pour la définition de `point_3d`). Pour obtenir `x`, `y`, et `z`, il faut utiliser la formule

```
x = (X * Xscale) + Xoffset
y = (Y * Yscale) + Yoffset
z = (Z * Zscale) + Zoffset
```

3 Travail à réaliser

Le but de ce travail est de lire des fichier LAS version 1.2 tels que décrits un peu plus haut, et de retourner un tableau avec tous les coordonnées de tous les points qu'il contient. Pour lire le fichier, il faut utiliser deux structures de données.

1. La structure d'entête de fichier LAS
```
typedef struct __attribute__((__packed__)) {
    char signature[4];

    uint16_t source_id;
    uint16_t global_encoding;

    uint32_t pid_data_1;
    uint16_t pid_data_2;
    uint16_t pid_data_3;
    uint8_t  pid_data_4[8];
```

```
    uint8_t version_major;
    uint8_t version_minor;

    char system_id[32];
    char gen_soft[32];

    uint16_t creation_day;
    uint16_t creation_year;

    uint16_t header_size;
    uint32_t off_to_point_data;
    uint32_t num_var_length_record;

    uint8_t point_data_format_id;
    uint16_t point_data_record_length;

    uint32_t num_point_records;
    uint32_t num_points_by_return[5];

    double x_scale_factor;
    double y_scale_factor;
    double z_scale_factor;

    double x_offset;
    double y_offset;
    double z_offset;

    double max_x;
    double min_x;

    double max_y;
    double min_y;

    double max_z;
    double min_z;
} header;
```

2. La structure point “lidar” (point record)

```
typedef struct __attribute__((__packed__)) {
    int32_t x, y, z;
    uint16_t intensity;
    uint8_t flags;
    unsigned char classification;
    unsigned char scan_angle_rank;
    unsigned char user_data;
    uint16_t point_source_id;
    double gps_time;
} point_record_1;
```

Notez qu'on utilise ici les entiers avec des tailles bien définies (contrairenemt au type `int`) pour être certain · e · s que les variables ont toujours la bonne taille en mémoire. Il faut aussi mettre en évidence l'utilisation de l'attribut de la structure `__attribute__((__packed__))`. Cela permet d'éviter des optimisations du compilateur et d'avoir les données qui sont représentées de façon contiguë en mémoire.

Une fois que ces structures sont définies, vous pouvez lire un fichier LAS 1.2 de façon très simple.

3.1 Lecture de l'entête

Pour cette partie vous devez implémenter la fonction

```
header header_from_file(char *fname);
```

Dans un premier temps il faut ouvrir (à l'aide de `fopen()`) le fichier dont le nom est `fname` en mode lecture seule avec l'option binaire `"rb"`. Puis, vous aurez besoin de la fonction `fread()` (`man 3 fread` pour avoir la documentation de cette fonction). Cette fonction peut échouer, n'oubliez pas de vérifier que tout s'est bien passé.

Implémentez également la fonction

```
void print_header(header *h);
```

permettant d'afficher le contenu d'un header. Cela vous sera sans doute utile pour débugger.

3.2 Lecture des points

Une fois l'entête lue, vous devez écrire une fonction qui lit tous les points "lidar" du fichier, et retourner un tableau contenant les points "géométriques" (`point_3d`). Pour ce faire vous devez implémenter la fonction suivante:

```
point_3d *points_from_file(char *fname, header h, uint32_t *num_points);
```

Cette fonction modifiera également la variable `num_points` afin d'avoir le nombre de points contenus dans le tableau `point_3d*` retourné par cette fonction.

Comme pour la lecture de l'entête, il faut ouvrir le fichier `fname` en mode lecture binaire. Puis, vous devez utiliser la fonction `fseek()` vous permettant de déplacer le pointeur fichier d'un nombre d'octets spécifiés en argument (`man 3 fseek` pour la documentation sur cette fonction). Pour lire les points vous devez vous déplacer dans le fichier du nombre d'octets contenus dans la variable `off_to_point_data` du header, puis lire tous les points à l'aide de la fonction `fread()`.

Une fois chaque point lu, n'oubliez pas de le transformer à l'ade de la formule

```
x = (X * Xscale) + Xoffset
y = (Y * Yscale) + Yoffset
z = (Z * Zscale) + Zoffset
```

où `X/Y/Zscale` et `X/Y/Zoffset` se trouvent dans le header.

Implémentez également la fonction

```
void print_point_record(point_record_1 *p);
```

permettant d'afficher le contenu d'un header. Cela vous sera sand doute utile pour débugger.

3.3 Les points géométriques

La structure des points géométriques est la structure `point_3d`

```
typedef struct {
    double x, y, z;
} point_3d;
```

Mathématiquement chaque point, $\vec{p}$, est représenté par un vecteur à trois composantes

$$\vec{p} = (x, y, z). \tag{1}$$

Comme vous l'avez peut-être déjà vu, les vecteurs s'additionnent, se soustraient et se multiplient par des scalaires un peu comme des nombres réels. Soient deux vecteur $\vec{p}$, $\vec{u}$

$$\vec{p} = (p_1, p_2, p_3), \quad \vec{u} = (u_1, u_2, u_3), \tag{2}$$

et un scalaire $a \in \mathbb{R}$.

La somme est définie comme

$$\vec{p} + \vec{u} = (p_1 + u_1, p_2 + u_2, p_3 + u_3). \tag{3}$$

La soustraction est définie comme

$$\vec{p} - \vec{u} = (p_1 - u_1, p_2 - u_2, p_3 - u_3). \tag{4}$$

La multiplication par un nombre, a, comme

$$a \cdot \vec{p} = (a \cdot p_1, a \cdot p_2, a \cdot p_3). \tag{5}$$

On peut également calculer la norme d'un vecteur, notée $||.||$, à l'aide de la formule

$$||p|| = \sqrt{p_1^2 + p_2^2 + p_3^2}. \tag{6}$$

Finalement, on peut calculer la distance, d, entre deux points comme la norme de la différence entre ces deux points

$$d = ||\vec{p} - \vec{u}||. \tag{7}$$

Il vous faudra écrire les fonctions correspondant aux opérations ci-dessus.

Les signatures de ces fonctions sont:

- La fonction qui crée un point à partir de 3 nombres:
  `point_3d point_3d_create(double x, double y, double z);`
- La fonction additionnant deux points en place

```
void point_3d_add_inplace(point_3d *p1, point_3d *p2);
```

— La fonction soustravant deux points en place
```
void point_3d_sub_inplace(point_3d *p1, point_3d *p2);
```

— La fonction multipliant un point avec un nombre en place
```
void point_3d_scale_inplace(point_3d *p1, double a);
```

— La fonction additionant deux points
```
point_3d point_3d_add(point_3d *p1, point_3d *p2);
```

— La fonction soustravant deux points
```
point_3d point_3d_sub(point_3d *p1, point_3d *p2);
```

— La fonction multipliant un point avec un nombre
```
point_3d point_3d_scale(point_3d *p1, double a);
```

— La fonction calculant la norme d'un point
```
double point_3d_compute_norm(point_3d *p);
```

— La fonction calculant la distance entre deux points
```
double point_3d_compute_distance(point_3d *p1, point_3d *p2);
```

— Une fonction affichant un point
```
void point_3d_print(point_3d *p);
```

3.4 Tests

Aucun fichier de test ne sera fourni pour ce travail pratique. A vous d'en écrire quelques uns pour vérifier que tout fonctionne bien. Néanmoins, vous trouverez ci-dessous, les valeurs attendues pour l'entête, ainsi que pour les deux premiers points du fichier LIDAR fourni sur cyberlearn.

Inspirez-vous de la structure de fichiers tests fournis jusque-là pour écrire les vôtres.

Pour la lecture du fiichier LIDAR, l'entête doit contenir les valeurs suivantes:

```
signature = LASF
source_id = 0
global_encoding = 1
pid_data_1 = 0
pid_data_2 = 0
pid_data_3 = 0
pid_data_4[0] = 0
pid_data_4[1] = 0
pid_data_4[2] = 0
pid_data_4[3] = 0
pid_data_4[4] = 0
pid_data_4[5] = 0
pid_data_4[6] = 0
pid_data_4[7] = 0
version_major = 1
version_minor = 2
system_id =
gen_soft = TerraScan
```

```
creation_day = 160
creation_year = 2017
header_size = 227
off_to_point_data = 229
num_var_length_record = 0
point_data_format_id = 1
point_data_record_length = 28
num_point_records = 8212287
num_points_by_return[0] = 7826831
num_points_by_return[1] = 353517
num_points_by_return[2] = 30221
num_points_by_return[3] = 1649
num_points_by_return[4] = 60
x_scale_factor = 0.001000
y_scale_factor = 0.001000
z_scale_factor = 0.001000
x_offset = 2000000.000000
y_offset = 1000000.000000
z_offset = -0.000000
max_x = 2499499.999000
min_x = 2499000.000000
max_y = 1115999.999000
min_y = 1115500.000000
max_z = 526.168000
min_z = 301.708000
```

Le premier point des `point_record_q` contient:

```
x = 499063511
y = 115793589
z = 378956
intensity = 18331
flags = 9
classification = 16
scan_angle_rank = 29
user_data = 1
point_source_id = 37
gps_time = 171641138.632467
```