TP11 – Infographie

Langage C (LC4)

Semaine du 7 avril 2014

Dans ce TP, nous voulons implémenter les deux première parties de TD11. Le langage *C* n'ayant pas de système de gestion de graphique standard, nous allons créer des fichiers d'image qui nous permettront, ensuite, de visualiser les résultats, par exemple dans un navigateur web.

1 Portable Pixmap Format

Nous commençons par créer une matrice bidimensionnelle, comme dans la partie 1 du TP9, ou nous allons stocké les informations relatives à une image. Nous utilisons les structures de donnée suivantes :

```
typedef struct {
  int x, y;
} pixel;

typedef struct {
  int R, G, B;
} color;

typedef struct {
  int width, height;
  color map[1];
} image;
```

Chaque case du tableau `map` contient ainsi les composantes RGB de la couleur du pixel de l'image dont les coordonnées correspondent à l'emplacement de cette case dans la matrice.

Question 1. Ecrivez une fonction `make_canvas(int width, int height, color c)` qui crée une image vide de taille `width` × `height` avec l'arrière plan de couleur `c`.

Question 2. Ecrivez une fonction `put_pixel(image *, pixel, color)` qui colorie un pixel donné d'une image d'une couleur donnée.

Question 3. En modifiant la fonction `\scan_triangle` de TD11 et en utilisant la fonction précédente, écrivez une fonction `draw_triangle(image *, triangle *, color)` qui ajoute un triangle coloré à une image.

Pour sauvegarder nos images, nous utiliserons un format simple de fichiers graphiques, PPM (*Portable PixMap format*), avec un codage ASCII.

Un fichier `.ppm` commence toujours par un "nombre magique" à deux octet qui contient les caractères `P3` ou `P6` qui correspondent respectivement au codages ASCII et binaire. Nous utiliserons, donc, le nombre magique `P3`.
Le nombre magique est suivi d'un séparateur (ou "caractère blanc" : espace, retour de ligne, etc.). Ensuite, il y aura deux entiers en base 10, toujours séparés par un séparateur, correspondant au nombre de pixels qui constituent respectivement la largeur et la hauteur de l'image.
La donnée qui vient compléter l'entête d'un fichier `.ppm`, c'est la valeur maximum que peut prendre chacun des composants RGB de la couleur, typiquement 15 ou 255.
Dans le reste du fichier sont stockées à la suite les composantes RGB de la couleur de chaque pixel, de gauche à droite, du haut en bas. Par convention, nous séparons les coordonnées d'un même pixel par des espaces, et les pixels entre eux par des retours de la ligne. Un fichier `.ppm` peut donc ressembler à cela :

```
P3
1024 768
255

255 255 255
255   0   0
255   0   0
255   0   0
0   127   0
0   127   0
0   127   0
...
```

Question 4. Supposons que quelque part au début de notre code nous avons défini `const int max_color`. Ecrivez maintenant une fonction `void write_to_ppm(char *, image *)` qui ouvre un fichier dont le nom lui est passé en argument pour écrire dedans (la fonction `fopen` de l'entrée sortie standard), qui y stocke une image dont l'adresse lui est passée en argument et qui, enfin, ferme le fichier (`fclose`).

Question 5. A l'aide de la fonction `draw_triangle` et la fonction précédente créez un fichier `.ppm` et visualisez-le.

2 Les coordonnées barycentriques

Question 6. Soit `typedef pixel triangle[3];` un type pour désigner un triangle. Ecrivez une fonction `int is_inside(triangle *, pixel)` qui renvoie 1 si son argument `pixel` se trouve à l'intérieur de son argument `triangle` et 0 sinon.

Les coordonnées barycentriques affines d'un point P par rapport à un triangle ABC sont trois nombre réels α, β et γ satisfaisant les trois équations suivantes :

$$x_P = \alpha x_A + \beta x_B + \gamma x_C$$
$$y_P = \alpha y_A + \beta y_B + \gamma y_C$$
$$\alpha + \beta + \gamma = 1$$

Pour représenter un point par ces coordonnées barycentrique, nous utilisons la structure suivante :

```
typedef struct {
        double alpha, beta, gamma;
        triangle tri;
} point;
```

Question 7. Ecrivez une fonction `pixel pixel_from_point(point)` qui calcule les coordonnées cartésiennes d'un pixel à partir de ces coordonnées barycentriques relatives à un triangle donnée.

Question 8. Ecrivez une fonction `point point_from_pixel(pixel, triangle*)` qui calcule les coordonnées barycentriques d'un pixel en fonction de ses coordonnées cartésiennes par rapport à un triangle donné. [1]

Les coordonnées barycentriques déterminent la position d'un point sur un plan en indiquant sa proximité relative par rapport aux sommets d'un triangle. Quand est-ce que nous avons $\alpha \geq 1$? Quand est-ce que nous avons $\alpha \leq 0$?

Question 9. Pouvez-vous maintenant réécrire la fonction `is_inside` d'une façon plus simple ?

3 Palette de couleurs

Question 10. En utilisant les fonctions précédentes de TD et TP (et les modifiant si besoin est) écrivez une fonction `spectrum(char *file_name, int width, int height, triangle *)` qui crée un fichier `.ppm` dont le nom et la taille lui sont passés en argument et où il dessine sur un fond blanc une palette de couleur dans le triangle qui lui est passé en argument, de façon à ce que le barycentre du triangle soit gris moyen et chacun de ses sommets d'une des couleurs primaires.

1. *Anti-sèche.* En résolvant les équations, nous aurons :

$$\alpha = \frac{(y_B - y_C)(x_P - x_C) + (x_C - x_B)(y_P - y_C)}{(y_B - y_C)(x_A - x_C) + (x_C - x_B)(y_A - y_C)}$$
$$\beta = \frac{(y_C - y_A)(x_P - x_C) + (x_A - x_C)(y_P - y_C)}{(y_B - y_C)(x_A - x_C) + (x_C - x_B)(y_A - y_C)}$$