Jean Perrin
Faculté des Sciences
Université d'Artois

C avancé
TD n° 5

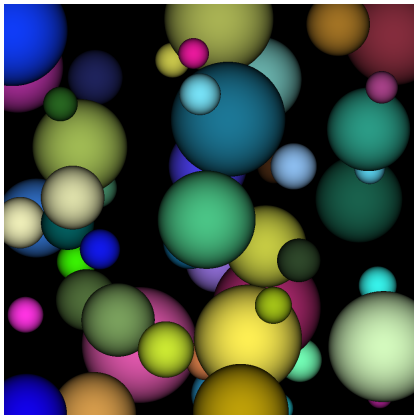
UNIVERSITÉ D'ARTOIS
Licence 3 – semestre 1
2022 – 2023

LES STRUCTURES

1 Ray Tracing

Le but de cet exercice est d'avoir un programme qui permet de faire du *ray tracing* sur un ensemble de sphères. Le *ray tracing* est une technique de calcul d'optique par ordinateur, qui consiste à simuler le parcours inverse de la lumière, c'est-à-dire qu'on calcule les éclairages de la caméra vers les objets puis vers les lumières, alors que dans la réalité la lumière va de la scène vers l'œil. Plus précisément, pour chaque pixel de l'image à générer, cela consiste à lancer un rayon depuis le point de vue dans la scène 3D (la caméra). Le premier point d'impact du rayon sur un objet définit l'objet concerné par le pixel correspondant.

Dans le contexte de ce TP nous verrons une version très simplifié du *ray tracing* qui consiste à regarder la scène du dessus et où nous ne considérons pas la réflexion. La figure suivante donne un exemple :



Dans un premier nous allons fixer la taille de la scène à `DIM` × `DIM` :

```
const unsigned DIM = 1024;
```

Ensuite il faut générer de manière aléatoire un ensemble de sphères.

```
const unsigend NB_SPHERE = 100
```

Une sphère est définie par ses coordonnées dans le plan, par son rayon et par sa couleur (donnée par la quantité de rouge, vert et bleu en pourcentage (entre 0 et 1)). Proposez une structure pour représenter une sphère et utiliser l'instruction `typedef` pour la nommer `sphere`.
Une fois la représentation de la sphère en mémoire posée, écrivez la fonction `generateRandomSpheres` qui génère un ensemble de `nbSphere` dans le plan de taille `dim` × `dim`. Lors de la génération il faut bien faire attention à ce que les sphères ne s'intersectent pas. Voici l'en-tête de la fonction :

```
void generateRandomSpheres(sphere *spheres, unsigned nbSphere, unsigned dim);
```

Il faut ensuite une structure pour stocker chaque pixel de la scène. La scène sera quant à elle représentée dans une matrice (en représentation en ligne par exemple).

```
#include <sys/types.h>

typedef struct {
  u_int8_t r, g, b;
} pixel;

pixel *images;
```

Pour calculer la valeur de chaque pixel vous devrez parcourir pour chaque position la liste des sphères, et vous devrez retourner la couleur associée à la sphère heurtée si elle existe, et `-INF` sinon. Il faudra aussi appliquer un dégradé en fonction de la distance par rapport au centre de la sphère. Pour réaliser cela vous aurez besoin de la fonction `hit` suivante :

```
float hit(sphere s, float ox, float oy, float *n) {
  float dx = ox - s.x;
  float dy = oy - s.y;

  if (dx * dx + dy * dy < s.radius * s.radius) {
    float dz = sqrtf(s.radius * s.radius - dx * dx - dy * dy);
    *n = dz / sqrtf(s.radius * s.radius); // pour l'ombre
    return dz + s.z;
  }

  return -INF;
} // hit
```

Une fois que vous aurez calculé la valeur de chaque pixel vous devrez créer un fichier permettant de contenir l'image. Pour cela, vous allez utiliser le codage `ppm` qui permet de stocker des images graphiques. Plus précisément, vous utiliserez le format P3 qui se décrit ainsi :

```
P3
# Le P3 signifie que les couleurs sont en ASCII, et qu'elles sont en RGB.
# Par 3 colonnes et 2 lignes :
3 2
# Ayant 255 pour valeur maximum :
255
255  0   0     0  255  0       0   0  255
255 255  0    255 255 255      0   0   0
```

Le format limite le nombre de pixel par ligne, afin de vous simplifiez la vie, écrivez un pixel par ligne.

2 Tic-Tac-Toe

Le but de cet exercice est encore de réaliser un programme permettant de jouer au Tic-Tac-Toe. Cependant, cette fois nous seront encore plus économe puisque nous souhaitons n'utiliser que 4 octets de mémoire. Pour rappel, le Tic-Tac-Toe est un jeu à deux joueurs. Ils doivent remplir chacun à leur tour une case de la grille avec le symbole qui leur est attribué : O ou X. Le gagnant est celui qui arrive à aligner trois symboles identiques, horizontalement, verticalement ou en diagonale.

```
   |   |
-----------
   |   |
-----------
   |   |
Ligne: 1
Colonne: 1

 X |   |
-----------
   |   |
-----------
   |   |
Ligne: 2
Colonne:
```

Il est toujours possible de demander à l'utilisateur un `unsigned char` à l'aide de la séquence `%hhu` de la fonction `scanf`.