Algorithmique et Programmation

—

Examen sur machine

—

G1: monasse*(at)*imagine.enpc.fr G2: david.ok*(at)*imagine.enpc.fr
G3: eric.bughin*(at)*gmail.com G4: renaud.marlet*(at)*enpc.fr
G5: drarenij*(at)*imagine.enpc.fr G6: vialette*(at)*univ-mlv.fr

—

10/12/10

—

1 Enoncé

1.1 Visualisation 3D par z-buffer

Nous allons visualiser des volumes 3D agencés dans l'espace sur l'écran et allons les animer. Nous nous restreignons à des polyèdres réguliers, mais le principe serait similaire pour d'autres volumes.

La principale difficulté réside dans la gestion des parties cachées (nos volumes sont opaques), ou occlusions. Nous utiliserons pour cela la méthode classique du *z-buffer*, qui est également utilisée dans les interfaces graphiques 3D openGL et DirectX par exemple, et donc dans la plupart des jeux vidéo.

L'idée de base est simple : pour chaque pixel on enregistre la hauteur (coordonnée z) du point 3D projeté le plus haut. Cela forme une carte des hauteurs appelée le z-buffer. On représente la surface par une série de triangles ; chacun de ceux-ci est projeté dans un ordre quelconque en un ensemble de pixels. On ne met le pixel dans l'image à la couleur du triangle que si la hauteur en ce point est supérieure à celle enregistrée dans le z-buffer jusqu'à présent. Si c'est le cas, on enregistre en plus ce nouveau z dans le z-buffer.

De cette façon, on peut projeter les triangles dans n'importe quel ordre et un pixel est réécrit chaque fois qu'on trouve un nouveau point plus élevé aux mêmes coordonnées x et y.

Précisons enfin que nous utilisons le principe de projection à l'infini, c'est-à-dire qu'un point $\begin{pmatrix} x & y & z \end{pmatrix}$ se projette en $\begin{pmatrix} x & y \end{pmatrix}$.

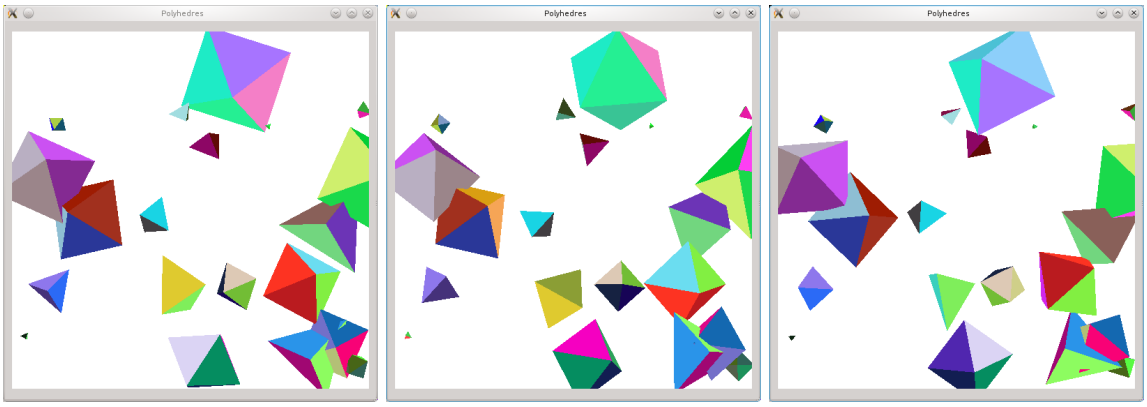

FIGURE 1 – Des tétraèdres et octaèdres en rotation dans l'espace. Aucun traitement particulier n'est fait pour les solides qui s'intersectent.

1.2 Travail demandé

Il est plus important de livrer un code clair et qui compile sans warning, même s'il ne répond pas à toutes les questions. Pour cela, vérifiez à chaque étape que votre programme compile et se lance correctement.

1. Créez un nouveau projet Imagine++ nommé `Polyedres` et écrivez le `main`, qui ouvre une fenêtre de taille fixe.
2. Preparez des fichiers `poly.h`, `poly.cpp`, `buffer.h` et `buffer.cpp`. Les deux premiers serviront aux structures `Point`, `TriangleFace` et `Polyedre` ainsi qu'aux fonctions s'y rapportant, alors que les deux derniers seront spécifiques à la visualisation par z-buffer.

1.2.1 Polyèdres

3. Créez une structure `Point` correspondant à un point ou à un vecteur 3D de l'espace (coordonnées réelles, non entières).
4. Ecrivez les opérateurs d'addition et de soustraction de deux `Point`s et l'opérateur de multiplication par une constante.
5. Ecrire une structure `Polyedre` qui contient :
   (a) un tableau de `Point`s (les sommets), qu'on appellera `sommets`, et son nombre d'éléments. On fera attention au fait que le nombre de sommets n'est pas une constante.
   (b) un tableau de `TriangleFace`s [1] et son nombre d'éléments.
   (c) un point représentant le barycentre.

   La structure `TriangleFace` (que l'on écrira) représente un triangle (face ou morceau de face du polyèdre). Il est constitué d'un tableau de 3 index indiquant les indices des sommets du triangle dans le tableau `sommets`, et d'une couleur `Color` (type d'Imagine++).
6. Ecrire une fonction `set` qui prend en argument un `TriangleFace`, 3 index (des entiers) et une couleur pour les écrire dans le `TriangleFace`.
7. Ecrire une fonction `tetraedre(int w, int h, int rmax)` qui renvoie un `Polyedre` construit de la manière suivante :
   – 4 sommets de coordonnées $\begin{pmatrix}0 & 0 & 1\end{pmatrix}$, $\begin{pmatrix}1 & 0 & 0\end{pmatrix}$ et $\begin{pmatrix}-1/2 & \pm\sqrt{3}/2 & 0\end{pmatrix}$.
   – 4 triangles, joignant tout sous-groupe de 3 parmi ces 4 sommets.
   Complétez cette fonction dans les trois items suivants :
8. Créez une fonction `randCol` qui renvoie une couleur tirée au hasard et l'utiliser pour affecter une couleur à chaque triangle. On rappelle que l'on peut créer une couleur avec la syntaxe suivante : `Color col(r,v,b)`, où `r`, `v` et `b` prennent des valeurs entre 0 et 255. Ne pas oublier d'initialiser le générateur aléatoire dans le `main`.
9. Multiplier les points par un facteur d'échelle aléatoire ne dépassant pas `rmax`.
10. Placez le centre (champ de `Polyedre`, cf 5c) en une position aléatoire (abscisse entre 0 et `w`, ordonnée entre 0 et `h`, $z = 0$) et translater le polyèdre de ce vecteur à l'aide d'une fonction `translate` que vous écrivez.
11. Ecrivez une fonction `detruit` qui prend en paramètre un `Polyedre` et qui libère la mémoire allouée.

1.2.2 Visualisation

12. Ecrivez une structure `Buffer`, qui a comme champs un tableau de hauteurs z et une image couleur `im` de type `Image<Color>` (type Imagine++) qui représente l'image qui sera affichée.
13. Ecrivez la fonction `Buffer creeBuffer(int w,int h)` : elle alloue le tableau des z et initialise les dimensions de l'image :

    ```
    buf.im = Image<Color>(w,h);
    ```

    Par la suite, pour retrouver ces dimensions on pourra utiliser `buf.im.width()` et `buf.im.height()`.
14. Ecrivez la fonction `reinit` qui va mettre le z-buffer à une grande valeur négative et l'image tout en blanc. On rappelle que l'on peut accéder au pixel `(i,j)` d'une image `im` de la manière suivante : `im(i,j)`.

1. ne pas l'appeler `Triangle` pour éviter la confusion avec la structure du même nom dans Imagine++

15. Ecrivez une fonction `affiche` qui se contente de dessiner l'image du buffer dans la fenêtre d'affichage. On pourra utiliser par exemple la fonction `display(im)` qui affiche l'image `im` dans la fenêtre.
16. Ecrivez une fonction `projette(Polyedre p, Buffer& buf)` qui dessine la projection de `p` dans `buf`. Celle-ci appelle simplement une fonction
    `projette(TriangleFace tr, Point* sommets, Buffer& buf)`
    sur chacun des triangles et que nous allons implémenter.
17. Cette méthode fonctionne en trois temps :
    - Elle cherche le rectangle (de coordonnées entières) englobant le triangle.
    - Pour chaque pixel de ce rectangle (attention s'il déborde de l'image!) il calcule ses coordonnées barycentriques (voir équation 1 en annexe) et vérifie s'il est bien à l'intérieur du triangle (coordonnées toutes positives). Si c'est le cas, son z est la moyenne pondérée par les coordonnées barycentriques des z des sommets du triangle.
    - Si le z est au-dessus de la valeur enregistrée dans le z-buffer, on met à jour le `Buffer` comme indiqué dans l'introduction.
18. Ecrivez ces fonctions `rectangle` et `barycentre` et servez-vous en pour `projette`.

1.2.3 Animation

19. A ce stade, vous pouvez déjà afficher un tétraèdre (dont on ne peut jamais voir plus de 3 faces à la fois).
20. Pour varier le point de vue, nous allons faire tourner le polyèdre autour d'un axe arbitraire :
    (a) Ecrivez une fonction `randomVect()` renvoyant un vecteur 3D aléatoire de la sphère :
    $$\begin{pmatrix}\cos\alpha\cos\beta & \sin\alpha\cos\beta & \sin\beta\end{pmatrix},$$
    avec $\alpha \in [0, 2\pi]$ et $\beta \in [0, \pi]$, représentant la direction d'un axe de rotation.
    (b) Pour effectuer la rotation du polyèdre, il suffit de :
    - translater le polyhèdre pour le centrer à l'origine (fonction `translate_inv` à écrire.
    - appliquer la rotation à chacun des sommets (voir les équations 2 et 3 en annexe)
    - utiliser `translate` pour ramener le centre à sa position initiale.

    (c) Avant d'afficher votre tétraèdre, faites-le tourner de façon arbitraire suivant un vecteur donné par `randomVect()`.
21. Animer le tétraèdre en rotation, en le faisant tourner autour d'un axe arbitraire mais fixé une fois pour toutes. Multiplier le résultat de `randomVect` par 0.1 pour avoir un angle pas trop grand (environ 6°).
22. Faites tourner ainsi une bonne dizaine de tétraèdres, chacun ayant son propre axe de rotation.
23. **Question bonus :** ajouter une fonction `octaedre` sur le modèle de `tetraedre` dont les 6 sommets sont les points
    $$\begin{pmatrix}0 & 0 & \pm1\end{pmatrix} \text{ (points 0 et 5)}$$
    $$\begin{pmatrix}\pm1 & 0 & 0\end{pmatrix} \text{ (points 1 et 3)}$$
    $$\begin{pmatrix}0 & \pm1 & 0\end{pmatrix} \text{ (points 2 et 4)}$$
    et les 8 faces données par les points d'indices
    $$\begin{pmatrix}0 & 1 & 2\end{pmatrix}\ \begin{pmatrix}0 & 2 & 3\end{pmatrix}\ \begin{pmatrix}0 & 3 & 4\end{pmatrix}\ \begin{pmatrix}0 & 4 & 1\end{pmatrix}$$
    $$\begin{pmatrix}5 & 1 & 2\end{pmatrix}\ \begin{pmatrix}5 & 2 & 3\end{pmatrix}\ \begin{pmatrix}5 & 3 & 4\end{pmatrix}\ \begin{pmatrix}5 & 4 & 1\end{pmatrix}$$
    Ajoutez dans votre scène des octaèdres en rotation.

Important : Quand vous avez terminé, *nettoyez la solution* et créez une archive du projet *à votre nom*. Ne vous déconnectez pas avant que le surveillant ne soit passé vous voir pour copier cette archive sur clé USB.

1.3 Annexe : les formules

Coordonnées barycentriques

Le point $\begin{pmatrix} x & y \end{pmatrix}$ s'écrit $\alpha A + \beta B + \gamma C$ avec

$$\begin{aligned} d &= (x_B - x_A)(y_C - y_A) - (y_B - y_A)(x_C - x_A) \\ \alpha &= ((x_B - x)(y_C - y) - (y_B - y)(x_C - x))/d \\ \beta &= ((x_C - x)(y_A - y) - (y_C - y)(x_A - x))/d \\ \gamma &= ((x_A - x)(y_B - y) - (y_A - y)(x_B - x))/d \end{aligned} \tag{1}$$

d est l'aire (éventuellement négative) du parallèlogramme dont 3 sommets sont A, B et C. Si cette aire est trop petite $|d| < 1$, renvoyer des coordonnées négatives de manière à dire que le triangle est vide et donc que tout point est en-dehors.

Justification (pour les curieux) : on peut imposer $\alpha + \beta + \gamma = 1$ et on se retrouve alors avec le système :

$$\begin{pmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Par les formules de Cramer, on obtient

$$\alpha = \frac{\begin{vmatrix} x & x_B & x_C \\ y & y_B & y_C \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{vmatrix}} \quad \beta = \frac{\begin{vmatrix} x_A & x & x_C \\ y_A & y & y_C \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{vmatrix}} \quad \gamma = \frac{\begin{vmatrix} x_A & x_B & x \\ y_A & y_B & y \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{vmatrix}}.$$

Pour calculer de tels déterminants, on soustrait une colonne à chacune des deux autres et on développe suivant la troisième ligne. Par exemple :

$$\begin{vmatrix} x & x_B & x_C \\ y & y_B & y_C \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} x & x_B - x & x_C - x \\ y & y_B - y & y_C - y \\ 1 & 0 & 0 \end{vmatrix} = \begin{vmatrix} x_B - x & x_C - x \\ y_B - y & y_C - y \end{vmatrix} = (x_B - x)(y_C - y) - (y_B - y)(x_C - x).$$

Rotation 3D autour d'un axe

On code la rotation par un vecteur v dont la norme représente l'angle de rotation et la direction l'axe de rotation. On extrait d'abord ceci de v :

$$\begin{aligned} \alpha &= \sqrt{x_v^2 + y_v^2 + z_v^2} \\ v &\leftarrow v/\alpha \\ c &= \cos\alpha \\ s &= \sin\alpha \end{aligned} \tag{2}$$

(remarquez qu'on renormalise v pour qu'il soit de norme 1, si α est trop petit, on se contente de ne rien faire, car c'est une rotation faible) puis

$$\begin{aligned} d &= x_v x + y_v y + z_v z \\ x' &= d(1-c)x_v + cx - sz_v y + sy_v z \\ y' &= d(1-c)y_v + cy + sz_v x - sx_v z \\ z' &= d(1-c)z_v + cz - sy_v x + sx_v y \end{aligned} \tag{3}$$