Projet

Paramétrisation – Textures

Lumières – Bump-mapping

Version C#

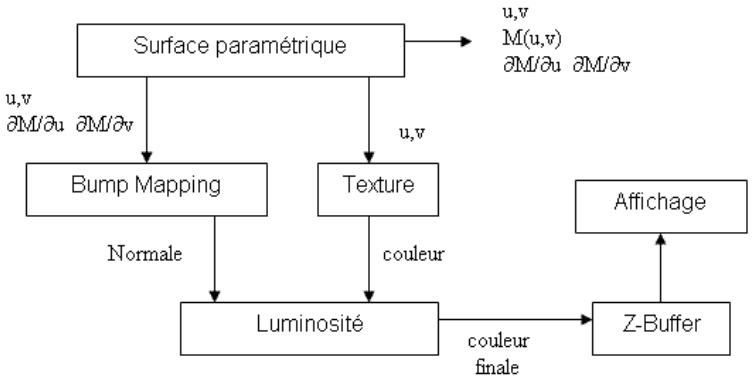
Ce projet vous propose de mettre en place les notions introductives à l'image de synthèse vues en cours. Un environnement a été mis en place afin de faciliter votre tâche.

Vous commencerez votre travail en intégrant votre code dans la fonction Go() présente dans le fichier ProjetEleve.cs. Cette fonction est appelée lors de l'appui du bouton présent sur la fenêtre de l'application. Lorsque vous allez créer des nouvelles classes et des nouvelles fonctions, créez les fichiers .cs correspondant et intégrez les dans votre projet. Pour ouvrir un projet, ne jamais ouvrir un fichier de code seul. Il faut ouvrir le fichier ***Projet_IMA_CS.sln*** qui contient toutes les informations relatives à l'ensemble des fichiers à compiler.

Lorsque vous voulez migrer votre projet d'un endroit à l'autre, procédez de la manière suivante : allez dans le répertoire de votre projet, supprimez les répertoires Debug ou Release et OBJ s'ils sont présents (ils contiennent des fichiers de compilation intermédiaire et des informations de débogage sans importance) et compressez le tout.

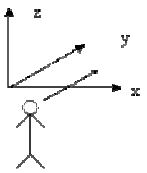
Schéma de la progression

Nous vous proposons de mettre en place successivement la paramétrisation des surfaces, le Z-Buffer et l'affichage à l'écran. Ensuite seront ajoutées les fonctions de luminosité, puis de textures… Il est fortement conseillé d'ajouter une IHM permettant d'ajouter des objets dynamiquement, de changer la position et la couleur de la lampe principale ou de charger un fichier scène…

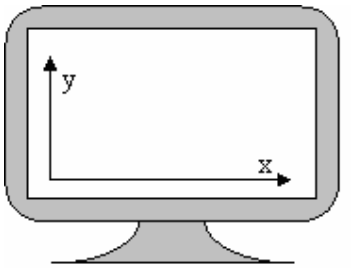


Conventions

Orientation de l'espace de travail, nous utiliserons le repère suivant :

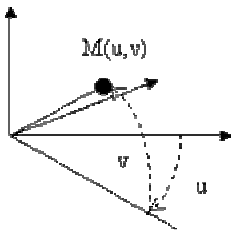


Rappel, le repère fourni par la classe BitmapEcran par l'intermédiaire de la fonction DrawPixel(x,y,coul) est le suivant



Nous utilisons comme repère pour les coordonnées sphériques les angles u et v suivants :

x(u,v) = cos(v) * cos(u)
y(u,v) = cos(v) * sin(u)
z(u,v) = sin(v)



Nous utilisons une simple projection orthogonale pour visualiser la scène à travers l'écran en mode Z-buffer. Ainsi, l'axe *x* de l'écran va correspondre à l'axe *x* du repère 3D et l'axe *y* de l'écran va correspondre à l'axe *z* de la scène 3D à une transformation prêt car ils ne sont pas orientés de la même manière.

Modèle d'illumination

Dans le projet est fournie la classe Couleur. Vous remarquez que de nombreux opérateurs ont été ajoutés, notamment l'opérateur + pour ajouter deux couleurs, l'opérateur * pour multiplier une couleur par un réel et l'opérateur * entre deux couleurs pour multiplier les composantes R,V et B deux à deux. Nous rappelons les formules données en cours pour chaque modèle d'illumination :

- Illumination ambiante (générée uniquement par la lampe ambiante)

$C_{amb} = C_{objet} * C_{ambiant}$

- Illumination diffuse (généré pour chaque lampe de la scène)

$C_{diff} = C_{objet} * C_{lampe} \times \cos(\theta)$

où θ représente l'angle entre le vecteur lumière (orienté du point courant vers la source) et le vecteur normal (orienté du point courant vers l'extérieur). Une coefficient multiplicatif supplémentaire peut être appliqué pour régler le niveau de l'effet diffus (entre 0 : rien et 1 : 100%)

- Illumination spéculaire (généré pour chaque lampe de la scène)

$C_{spec} = C_{lampe} \times \cos^{k}(S,O)$

Notons S, le vecteur idéal de réflexion de la lumière. Si L correspond au vecteur lumière (orienté du point courant vers la source) et si N correspond au vecteur normal au point courant, alors S est le symétrique de L par rapport à N. O représente le vecteur de l'observateur, il est orienté du point courant vers l'oeil de l'observateur. Notez qu'il faut par conséquent donner une position à cet oeil dans votre scène qui doit correspondre approximativement à votre position face à l'écran : c'est à dire devant la scène. Un coefficient multiplicatif supplémentaire peut être appliqué pour régler l'intensité de l'effet. k est l'exposant réglant la largeur du spéculaire sur l'objet, à choisir entre 30 et 100. On rappelle que :

$S = 2 N \times (L.N) – L$

Cette formule ne fonctionne bien sur que si les vecteurs sont normalisés

Séance de travail

● L'environnement de travail

Un squelette du projet (sous visual C# 2010) est fourni. Cette application est une application Windows Form. Sur la fenêtre de l'application vous trouverez un seul bouton déclenchant la fonction Go() présente dans le fichier ProjetEleve.cs. Les autres fichiers fournis sont :

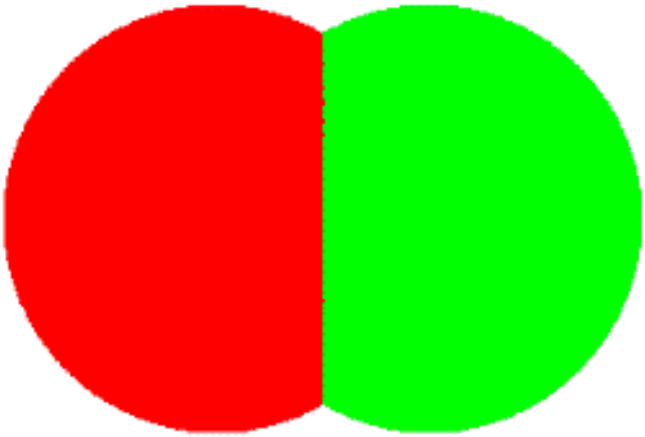
- V3.cs : classe des vecteurs 3D ainsi que les opérations classiques : + - * / …
- Couleur.cs : structure représentant les informations couleur RVB et des opérateurs permettant de les manipuler facilement
- Texture : permet de créer une texture à partir d'images. On peut ainsi accéder à l'information de couleur par la fonction LireCouleur(u,v). Pour être indépendant de la résolution de l'image, (0,0) et (1,1) correspondent aux coins de l'image.
- Inver_Coord_spherique : pour les plus courageux qui veulent calculer l'intersection entre un rayon et une sphère, cette fonction vous sera très utile

● Etape 1 : surfaces paramétriques et Z-Buffer

Un Z-Buffer est un tableau 2 dimensions. Chaque valeur dans le tableau est associée à un pixel de la zone d'affichage. Ainsi pour chaque pixel allumé à la position (x_ecran,y_ecran), on va stocker la composante en y (la

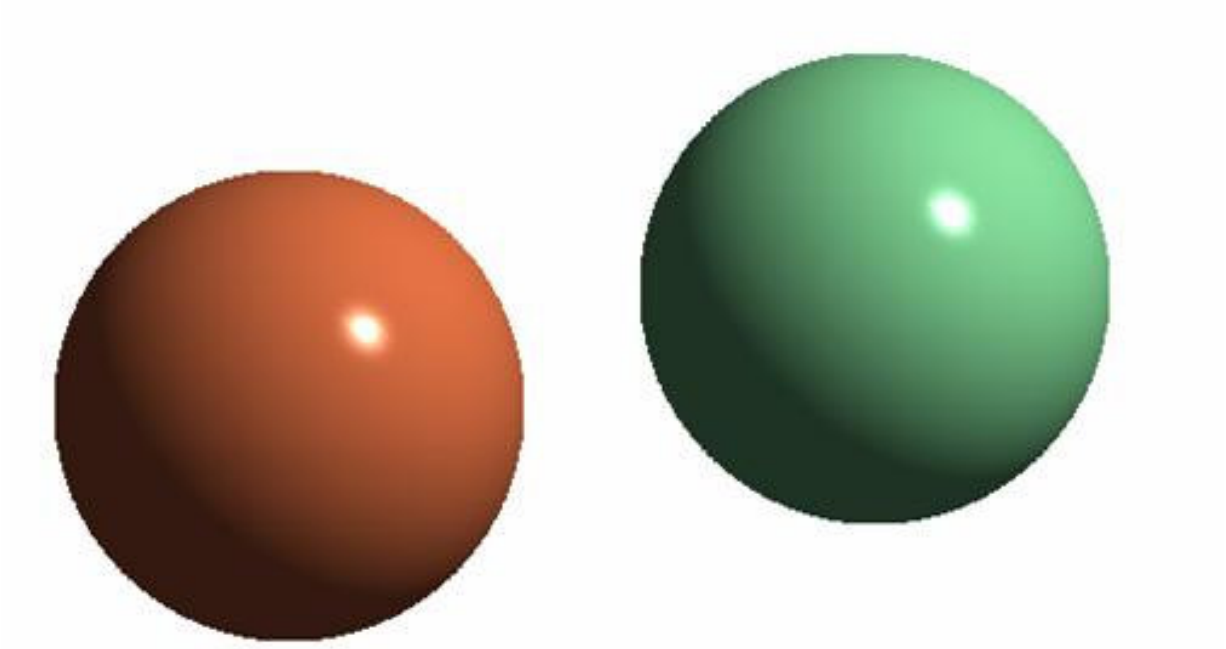
profondeur) dans la case ZBuffer[x_ecran][y_ecran]. Lorsque je trace un nouveau point dont les coordonnées écran sont identiques, je dois alors comparer leur profondeur. Si le nouveau point est devant, il est alors affiché et la profondeur correspondante dans le ZBuffer est mise à jour. Cette façon de procéder permet de gérer aisément l'occultation entre objets.

En utilisant une paramétrisation correcte à partir des coordonnées sphériques, dessinez deux sphères de rayon 100 et de coordonnées (200,0,200) et (350,0,200). Vous devriez obtenir le résultat suivant :



● Etape 2 : modèles d'illumination

Appliquez à ces deux sphères les modèles de lumière ambiante, diffuse et spéculaire vus en cours. Choisissez la source de lumière que vous souhaitez ainsi que son affaiblissement. Nous présentons le résultat obtenu pour une source directionnelle (1,-1,1) sans affaiblissement :



● Etape 3 : application des textures

Grâce à la paramétrisation bidimensionnelle (u,v) vous pouvez facilement appliquer une texture provenant d'une image extérieure. A partir des fichiers joints au projet, appliquez des textures sur vos sphères tout en laissant activer les modèles d'illumination précédents. Voici le résultat obtenu à partir des textures OR et METAL.

● Etape 4 : Bump-Mapping

Le Bump-Mapping est une technique permettant de simuler la variation des normales de l'objet relativement à une faible déformation de la surface qui produit cependant un effet visuel notable. Il s'agit par exemple des crénelures, des cabossages, des rainures… Nous allons supposer que le point M(u,v) à la surface de l'objet subi une variation de hauteur : h(u,v) dans la direction de la normale extérieure N(u,v). Nous posons :

$$M'(u,v) = M(u,v) + h(u,v) * N(u,v)$$

Par définition nous rappelons que la normale d'une surface paramétrique en u et v est égale à :

$$N(u,v) = \pm \partial M/\partial u \wedge \partial M/\partial v / ||\partial M/\partial u \wedge \partial M/\partial v||$$

Calculez l'expression de la nouvelle normale N'(u,v) à partir des égalités précédentes et des approximations devant être effectuées. Appliquez ces résultats à la paramétrisation sphérique de nos objets. Le résultat obtenu est dépendant de la grandeur $\partial h/\partial u$ et $\partial h/\partial v$. La carte de déformation h(u,v) (le bump-map) est stockée dans une image en niveau de gris. Pour estimer la

valeur ∂h(u,v)/ ∂u, utilisez une fonction fournie dans la classe Texture. Voici le résultat obtenu avec deux bump géométriques :

● Etape 5 : les objets

Le but de cette étape est de construire une classe représentant l'objet sphère. Ses attributs internes vont être par exemple : son rayon, son centre, sa couleur… Après cela, vous intégrerez un nouveau type d'objet : les rectangles. D'autres objets pourront être ajoutés. Dans tous les cas, l'ensemble de ces classes doit hériter d'une superclasse Objet3D commune à tous les futurs objets à afficher.

● Etape 6 : les lampes

Prévoyez une classe Lumière stockant tous les attributs propres à un modèle d'éclairage (type, position…). Vous pouvez commencer par le type : directional light. Comme pour la superclasse Objet3D, vous devrez créer une superclasse lampe qui sera une classe mère pour vos différents types de lampe. Essayez de ne pas mélanger la notion de lampe (cad une entité qui éclaire les objets de la scène) avec les modèles d'illumination (modèles qui décrivent la manière dont les objets renvoie la lumière à leur surface).

● Etape 7 : le ray-casting

Implémentez cette méthode afin de dessiner les boules anciennement tracées. N'oubliez pas de désactiver le Z-Buffer qui devient alors inutile. Reconstruisez le dessin obtenu à l'étape 4.

● Etape 8 : l'occultation

Pour chaque rayon touchant un objet, vérifiez que le trajet allant de chaque source lumineuse vers ce point ne traverse pas un objet de la scène.

● **Etape 9 : le raytracing**

Implémentez l'appel récursif dû à la propagation des rayons par transmission et réflexion.

● **Etape 10 : surprises...**

Galeries :