Synthèse d'Image - TD06

TRANSPARENCE ET OPACITÉ

L'objectif de ce TD est de comprendre comment fonctionne le Blending en OpenGL pour rendre des objets transparents.

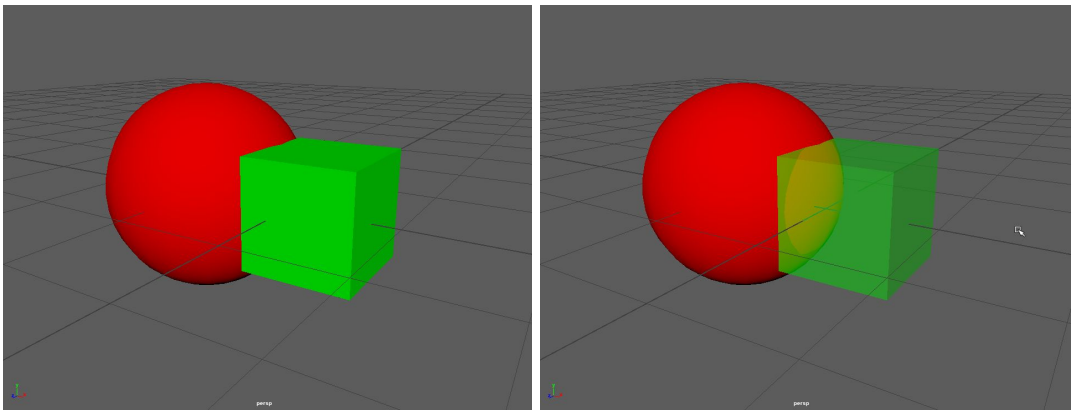
Vocabulaire

En synthèse d'image, on utilise souvent les mots "opacité", "transparence" et "alpha". Voici leurs définitions :

- L'**opacité** d'un objet correspond à "à quel point cet objet est opaque". On la représente par une valeur comprise entre 0.0 et 1.0. Quand l'opacité vaut 0.0, l'objet n'est pas opaque du tout, donc transparent. Quand l'opacité vaut 1.0, l'objet est complètement opaque et ne laisse passer aucune autre couleur.
- La **transparence** d'un objet correspond à "à quel point cet objet est transparent". C'est donc l'inverse de l'opacité. On la représente par une valeur comprise entre 0.0 et 1.0. Quand la transparence vaut 1.0, l'objet est totalement transparent. Quand la transparence vaut 0.0, l'objet n'est pas transparent du tout.
- L'**alpha** est un synonyme d'**opacité**.

La transparence et l'opacité sont donc liées par la formule :

opacité = alpha = 1.0 - transparence



à gauche : la sphère et le cube ont une opacité de 1.0 et une transparence de 0.0
à droite: le cube a une opacité de 0.5 et la sphère une transparence de 0.0

Introduction au Blending

En OpenGL, il n'existe pas de fonctionnalité de base pour recréer l'opacité des objets. Cette fonctionnalité est encapsulée dans une feature plus générale appelée **Blending**.

Le Blending, c'est l'action de **mélanger deux couleurs en fonction de certains paramètres**. Dans notre cas, nous allons effectuer un blending prenant en paramètre l'opacité des objets pour créer des effets de transparence.

Pour activer l'option de blending en OpenGL, il faut appeler la fonction :

```
glEnable(GL_BLEND);
```

Une fois le blending activé, il va falloir préciser à OpenGL quels sont les paramètres à utiliser dans notre fonction de blending. Mais pour savoir lesquels utiliser, il faut dans un premier temps comprendre comment fonctionne la fonction de blending OpenGL.

En OpenGL, lorsque vous dessinez un objet à l'écran et que le blending est activé, OpenGL applique la formule suivante pour déterminer la couleur de chaque pixel :

Cf = Cs * S + Cd * D

avec :

- Cf = la **couleur finale** à afficher
- Cs = la **couleur de l'objet** à dessiner (source)
- S = le facteur multiplicateur de la **source**
- Cd = la **couleur de l'objet** déjà présent (destination)
- D = le facteur multiplicateur de **l'objet déjà présent**

OpenGL effectue donc un mélange entre la couleur déjà présente dans le buffer (la destination) et la couleur que vous voulez dessiner (la source). Ce mélange est pondéré par les deux facteurs S et D qu'il faut donc définir via la fonction :

```
glBlendFunc(S, D);
```

Pour les valeurs S et D (appelées aussi *sfactor* et *dfactor*), vous avez le choix entre les valeurs suivantes : `GL_ZERO`, `GL_ONE`, `GL_SRC_COLOR`, `GL_ONE_MINUS_SRC_COLOR`, `GL_DST_COLOR`, `GL_ONE_MINUS_DST_COLOR`, `GL_SRC_ALPHA`, `GL_ONE_MINUS_SRC_ALPHA`, `GL_DST_ALPHA`, `GL_ONE_MINUS_DST_ALPHA`, `GL_CONSTANT_COLOR`,`GL_ONE_MINUS_CONSTANT_COLOR`,`GL_CONSTANT_ALPHA`, `GL_ONE_MINUS_CONSTANT_ALPHA`, `GL_SRC_ALPHA_SATURATE`, `GL_SRC1_COLOR`, `GL_ONE_MINUS_SRC1_COLOR`, `GL_SRC1_ALPHA`, et `GL_ONE_MINUS_SRC1_ALPHA`.

Les possibilités de blending sont donc très nombreuses. Ne paniquez pas ! Vous n'avez pas à apprendre par coeur ces paramètres. Ce que vous pouvez retenir, ce sont les associations les plus courantes pour arriver à rendre les effets les plus connus. Par exemple, les paramètres à choisir pour faire de la transparence sont les suivants :

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Ici, on dit donc que la couleur finale du pixel sera :

Couleur Source * Opacité Source + Couleur Présente * Transparence Source

Exercice 01 : Cube opaque et cube transparent

Pour ce TD vous pouvez repartir du fichier minimal.c fourni.

1. **Dans votre programme, avant la boucle de rendu, activez l'option de blending OpenGL, et spécifiez les paramètres de la fonction de blending pour pouvoir rendre des objets transparents.**
2. **Dans la boucle de rendu, dessinez un cube de couleur en spécifiant une valeur d'opacité de 0.5 pour la couleur. Pour cela, vous pouvez utiliser au choix la fonction `glColor4f` ou `glColor4ub`. Faites varier la valeur de l'opacité et constatez le résultat.**

Nous allons maintenant rendre plusieurs objets et constater que **l'ordre de dessin est important**.

3. **Dans votre programme, dessinez un deuxième cube de couleur différente qui chevauche le premier cube. Donnez une valeur d'opacité de 0.5 à ce nouveau cube.**
4. **Changez l'ordre de dessin des deux cubes et comparez les résultats.**
5. **Améliorez votre programme en ajoutant les fonctionnalités suivantes :**
    a. **Lorsque l'utilisateur appuie sur une touche du clavier numérique, le programme sélectionne un des cubes dessinés.**
    b. **Lorsque l'utilisateur appuie sur les flèches du clavier, le cube sélectionné se déplace dans l'espace.**
    c. **Lorsque l'utilisateur appuie sur les touche + et - du clavier, l'opacité du cube sélectionné change.**
    d. **Lorsque l'utilisateur appuie sur la touche Espace, le cube sélectionné devient le dernier cube dessiné.**

Exercice 02 : Matrix Rain Code

Nous allons mettre en place un programme représentant la fameuse pluie de nombre de la série Matrix :

Vous trouverez ci après quelques explications sur différents aspects du programme qu'il vous faudra utiliser pour parvenir au résultat attendu.

Chargement des textures

Pour les symboles, vous utiliserez une unique image fournie : **kana.png**.

Cette texture contient plusieurs symboles issus de l'écriture japonaise. Tous les symboles sont contenus dans la même image. Pour pouvoir afficher 1 seul symbole, vous devrez donc utiliser les bons paramètres de texture. Pour vous aider, voici les informations de proportion :

Chargement des textures avec canal alpha

Pour pouvoir rendre une image qui possède un canal alpha (comme c'est le cas du format PNG), vous devez modifier l'appel à la fonction glTexImage2D comme suit :

```
glTexImage2D(
  GL_TEXTURE_2D,
  0,
  GL_RGBA, // Pour dire à OpenGL de ranger l'image en 4 canaux
  image->w,
  image->h,
  0,
  GL_RGBA, // Pour dire à OpenGL que les données envoyées comprennent 4 canaux
  GL_UNSIGNED_BYTE,
  image->pixels);
```

Appliquer un masque de couleur sur un objet ou une texture

Voici une technique alliant 2 fonctions de blending différentes pour appliquer un masque de couleur sur une image en noir & blanc :

```
// 1 : dessiner le cube texturé

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glEnable(GL_TEXTURE_2D);
glBindtexture(GL_TEXTURE_2D, textureID)

glBegin(GL_QUADS);
    glTexCoord2f(0, 0);
    glVertex2f(-0.5, 0.5);
    ...
glEnd();

glDisable(GL_TEXTURE_2D);
glBindtexture(GL_TEXTURE_2D, 0)

// 2 : Appliquer un masque de couleur
glBlendFunc(GL_DST_COLOR, GL_ZERO);
// La couleur du masque de couleur (peut contenir un alpha < 1.0)
glColor4f(0.0, 1.0, 0.0, 1.0);
glBegin(GL_QUADS);
    glVertex2f(-0.5, 0.5);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.5, -0.5);
    glVertex2f(-0.5, -0.5);
glEnd();
```

Exercices

Dans votre programme :

1. **Changez la couleur de clear pour un noir complet (0, 0, 0, 1);**
2. **Dessinez un cube sur lequel vous afficherez la texture kana.png en prenant en compte sa transparence.**
3. **Faites en sorte de pouvoir dessiner 1 seul symbole (que vous pourrez changer facilement) au lieu de dessiner toute la grille.**
4. **Faites en sorte d'appliquer un masque de couleur vert à votre symbole affiché.**
5. **Faites varier la couleur du masque en fonction du temps, allant de la couleur vert à la couleur blanche.**
6. **Faites varier l'opacité du masque de couleur en fonction du temps, allant de 1.0 à 0.0 au bout de N secondes (variable que vous stockerez en amont de la boucle).**

Il est temps désormais :

7. **Mettez vos connaissances fraîchement acquises au service de la réalisation d'un petit programme qui reproduit le "Matrix Rain Code".**