| Responsable pédagogique | AF | **AM** | OP | PM |
|---|---|---|---|---|
| Période | Sem1 | Sem2 | **Sem3** | Sem4 |
| Volume horaire | Cours/TD | | TP | |
| | | | **4** | |

# [CPP]
# Matériaux OpenGL

Avertissement : Toute documentation autorisée, lire l'intégralité du sujet avant de commencer…

Documents à rendre : Les 5 fichiers développés (**`crvba.h`**, **`crvba.cpp`**, **`cmateriau.h`**, **`cmateriau.cpp`**, **`main.cpp`**) doivent impérativement comporter en en-tête vos nom et prénom.

Ajoutez en tête du fichier **`main.cpp`** un commentaire relatant l'état d'avancement des travaux (liste des questions auxquelles vous pensez avoir répondu).

Barème indicatif : Q1..Q14 + Q21 sur 10 points, Q15..Q20 + Q22..Q25 sur 10 points

## Cahier des Charges

On se propose de créer deux petites classes utilitaires en vue de définir et manipuler les matériaux tels qu'ils sont reconnus par la bibliothèque OpenGL.
Définition simplifiée : Un matériau est défini par un jeu de valeurs qui déterminent la réflexion de la lumière ambiante, diffuse et spéculaire pour chacune des composantes RVB (couleurs de base Rouge, Vert et Bleu). On ajoute à cela, pour chacune des trois réflexions, une indication de transparence nommée en général « canal alpha ». Ces 12 valeurs réelles doivent appartenir à l'intervalle 0..1. La définition du matériau est complétée par une valeur de brillance (nombre réel compris entre 0 et 128) et un nom (simple chaîne de caractères).
En pratique, une couleur RVBA est souvent encodée sous forme d'un mot de 32 bits : Les 24 premiers bits représentent chacune des composantes RVB codées sur 8 bits, et les 8 bits restant sont réservés pour le canal alpha (lui aussi codé sur 8 bits). La majorité des IHM de manipulation de couleurs proposent donc des possibilités de réglage dans l'intervalle entier 0..255.

Q1. Création de la classe **`CRvba`** (**`crvba.h`** + **`crvba.cpp`**) : Son rôle est de maintenir une couleur codée sur 32 bits, par exemple sous la forme d'un unique mot de 32 bits, de 4 entiers ou d'un tableau de 4 octets…

Attributs privés (minimum exigé) :
Q2. - de quoi stocker les 4 composantes RVBA…

Méthodes publiques à développer (minimum exigé) :
Q3. - un constructeur par défaut fixant la couleur noire (0,0,0) sans transparence (255),
Q4. - un constructeur acceptant 4 entiers dont le dernier est optionnel (255 par défaut),
Q5. - un sélecteur *inline* pour chacune des 4 composantes (méthodes **`const`**),
Q6. - un modificateur implémenté en externe pour chacune des 4 composantes.

Des méthodes privées ou des macros peuvent éventuellement être ajoutées, par exemple pour tester la validité et corriger si nécessaire les valeurs fournies par le client…

Q7. Ajout d'un programme de validation (**`main.cpp`**) :

```
#include "crvba.h"
#include <iostream>
using namespace std ;

int main(int argc, char** argv )
{
    CRvba   couleur1 ;
    CRvba   couleur2(115,157,172) ;
Q8.     // TODO : fixer couleur1 à cyan (rouge à 0, vert et bleu à 100%)
Q9.     // TODO : affichage des 4 valeurs de couleur1
Q10.    // TODO : affichage des 4 valeurs de couleur2
    return 0 ;
}
```

Q11. Création de la classe **`CMateriau`** (**`cmateriau.h + cmateriau.cpp`**) : elle maintient le nom, les réflexions ambiante, diffuse et spéculaire et la brillance d'un matériau. Le matériau par défaut doit correspondre aux données suivantes :

- nom = "défaut"
- réflexion ambiante = { 0.2, 0.2, 0.2, 1.0 }
- réflexion diffuse = { 0.8, 0.8, 0.8, 1.0 }
- réflexion spéculaire = { 0.0, 0.0, 0.0, 1.0 }
- brillance = 0.0

Attributs privés (minimum exigé) :

Q12. - de quoi stocker les éléments ci-dessus (le nom peut être du type STL **`string`**),

Méthodes publiques (minimum exigé) :

Q13. - un constructeur par défaut,

Q14. - un sélecteur pour chacune des 5 informations caractérisant le matériau (les réflexions doivent être retournées sous forme d'objets **`CRvba`**...).

Q15. - un constructeur susceptible de recevoir de quoi initialiser complètement le matériau sous forme d'un nom, de trois objets de classe **`CRvba`** et d'une valeur réelle optionnelle pour la brillance,

Q16. - un modificateur permettant de modifier la transparence du matériau à partir d'une valeur fournie dans l'intervalle 0..1. Le canal alpha des 3 réflexions doit être affecté.

Q17. - un constructeur de copie (si cela s'avère nécessaire),

Q18. - une surcharge de l'opérateur d'affectation (si cela s'avère nécessaire).

Méthodes privées (minimum exigé) :

Q19. - une méthode de conversion d'une valeur entière 0..255 vers une valeur réelle proportionnelle dans l'intervalle 0..1,

Q20. - une méthode assurant la conversion inverse de la précédente.

Programme de validation (**`main.cpp`** à compléter) :

```
CMateriau   mat1 ;
CMateriau   mat2( "turquoise",
                  couleur2,
                  CRvba( 0.396, 0.741, 0.691 ),
                  CRvba( 0.297, 0.308, 0.306 ) ) ;
```

Q21. // TODO : afficher les caractéristiques du matériau **`mat1`**

Q22. // TODO : copier **`mat2`** dans **`mat1`**

Q23. // TODO : fixer une transparence de 50% pour le matériau **`mat1`**

Q24. // TODO : afficher les caractéristiques du matériau **`mat1`**

Q25. Finaliser l'ergonomie de la classe **`CMateriau`** en proposant des méthodes compatibles avec la primitive **`glMaterialfv()`** d'OpenGL qui ne peut accepter en argument qu'un pointeur sur un tableau de **`float`** (suffixe fv = float vector). On souhaite par exemple pouvoir écrire :

```
glMaterialfv( GL_FRONT, GL_AMBIENT,   mat2.rAmbiante() ) ;
glMaterialfv( GL_FRONT, GL_DIFFUSE,   mat2.rDiffuse() ) ;
glMaterialfv( GL_FRONT, GL_SPECULAR,  mat2.rSpeculaire() ) ;
glMaterialfv( GL_FRONT, GL_SHININESS, mat2.brillance() ) ;
```