Intégration d'une méthode de mélange de couleurs physiquement réaliste dans un logiciel de simulation de peinture

Nicolas Barroso Valentin Chevrier Hugo Lévêque Ilyes Naidji Moearii Teraitetia

Client: David Vanderhaeghe
Université Paul Sabatier

Février 2017

Sommaire

Première partie

Recette

1 Introduction

Ce quatrième rapport rentre dans le cadre du Chef d'Oeuvre proposé au cours du Master Informatique Graphique et Analyse d'Images. Notre projet consiste à implémenter une solution au mélange de pigments dans un moteur de simulation de peinture réaliste.

2 Fonctionnement du programme

2.1 Version actuelle

2.1.1 Exigences fonctionnelles

- Priorité de niveau 1 :
  - EF1 : L'application doit utiliser la méthode de mélange de pigments décrite dans l'article de Baxter et col.[1].
  - EF2 : Le mélange de couleurs doit être contrôlé par les coefficients d'absorption et de diffusion des pigments à mélanger.
- Priorité de niveau 2 :
  - EF3 : L'application doit permettre à l'utilisateur de définir les coefficients d'absorption et de diffusion à l'aide d'une interface graphique.
- Priorité de niveau 3 :
  - EF4 : L'application doit permettre la visualisation du mélange de couleurs en temps réel.
  - EF5 : L'application doit proposer à l'utilisateur une gamme de pigments avec des coefficients d'absorption et de diffusion définis pour le mélange.
- Fonctionnalités supplémentaires :
  - EF6 : L'application permet à l'utilisateur d'enregistrer le pigment obtenu après le mélange de deux pigments sous la forme d'un fichier .xml qu'il pourra réutiliser par la suite.

n°	Description
EF1	Cette exigence est respectée.
EF2	Cette exigence est respectée.
EF3	L'interface graphique permet à l'utilisateur de choisir les pigments qu'il souhaite mélanger. Cependant il ne peut pas définir lui-même les coefficients d'absorption et de diffusion de ces pigments.
EF4	Cette exigence est respectée.
EF5	Cette exigence est respectée.

2.1.2 Exigences logicielles

- Priorité de niveau 1 :
  - EL1 : L'application doit être codée en C++.
  - EL2 : L'application doit utiliser l'A.P.I OpenGL.
  - EL3 : Tous les calculs concernant le mélange de couleurs doivent pouvoir s'effectuer sur CPU et GPU.
- Priorité de niveau 2 :
  - EL4 : L'interface graphique doit être codée en utilisant QT.
- Priorité de niveau 3 :
  - EL5 : L'application doit être intégrée dans le moteur de simulation de peinture du client.

n°	Description
EL1	Cette exigence est respectée.
EL2	Cette exigence est respectée.
EL3	Actuellement, le calcul du mélange de couleurs s'effectue uniquement sur le CPU. Par manque de temps, nous n'avons pas implémenté le calcul du mélange de couleurs sur le GPU.
EL4	Cette exigence est respectée.
EL5	Cette exigence n'est pas respectée.

3 Tests

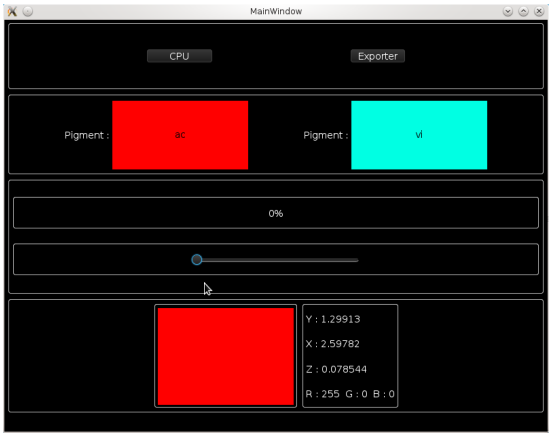
3.1 Tests unitaires

Lors de la phase de conception du projet, nous avions prévu une série de tests permettant de vérifier le bon fonctionnement de notre application avec son interface.

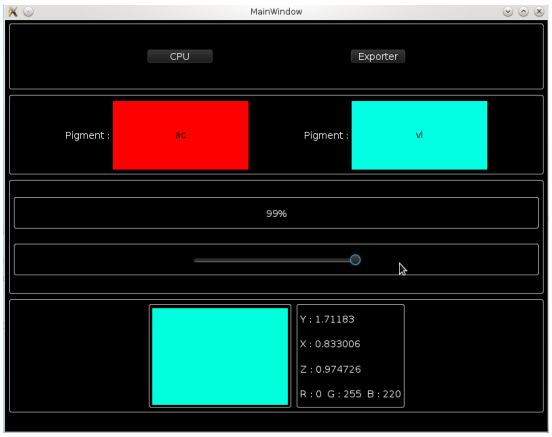
3.1.1 Tests validés

Chargement des données d'entrée
À la sélection d'un pigment via l'interface, les valeurs que l'application renvoie correspondent bien à celles contenues dans le fichier XML correspondant.

Tests de concentration
Lorsque nous mélangeons un pigment1 de concentration 100 et un pigment2 de concentration 0, le mélange obtenu est bien composé uniquement du pigment1 (et réciproquement).



(a) Pigment1 = 100 et Pigment2 = 0



(b) Pigment1 = 0 et Pigment2 = 100

3.1.2 Tests non validés

Enregistrement des données de sortie
Actuellement, l'application ne permet pas d'exporter le pigment mélange. Une fois que cette fonctionnalité sera ajoutée à l'interface, nous vérifierons que le fichier résultant est bien enregistré dans le format que nous avons défini.

3.2 Tests de validation

Lors de la phase de spécification du projet, nous avions prévu une série de tests de validation permettant de vérifier l'adéquation de notre application aux besoins de notre client.

Méthode de Kubelka-Munk
Dans un premier temps nous effectuerons différents mélanges de peinture réelle. Chaque mélange obtenu sera numérisé avec un capteur numérique. Ensuite nous comparerons les mélanges numérisés avec les mélanges obtenus via notre application. Il est primordial que ces comparaisons s'effectuent sur le même moniteur d'affichage afin que les deux mélanges à comparer subissent la même déformation à l'affichage. On pourra par exemple prendre en référence les mélanges réalisés dans l'article de Baxter et col.[1].

Nous n'avons pas eu le temps de réaliser ce test de validation.

Contrôle des coefficients K et S
Afin de tester si le calcul du mélange est contrôlé par les coefficients d'absorption et de diffusion de ces pigments, nous ferons varier ces coefficients. Si le mélange résultat diffère à chaque variation des coefficients, alors le test sera validé.

Ce test est validé par le slider de concentration de l'interface car la concentration influe directement sur les coefficients d'absorption et de diffusion de chaque pigment.

Saisie des coefficients K et S
Pour valider cette fonctionnalité, il suffit de demander à l'utilisateur de notre application de lancer le calcul du mélange après avoir sélectionné les pigments à mélanger. En sélectionnant des pigments pré-définis, l'utilisateur sélectionnera les coefficients d'absorption et de diffusion du pigment associé indirectement.

Fonctionnalité abandonnéee après réunion avec notre client car nous avons pu obtenir des données fiables sur les coefficients d'absorptions et de diffusions de nos pigments grâce Mr Jeremy Wendt (co-auteur de [1]).

Temps réel
Lorsque l'utilisateur lance le calcul du mélange de couleur via l'interface graphique, il ne doit pas percevoir de latence entre le moment où le calcul est lancé et le moment où le résultat du mélange s'affiche.

Le mélange de couleurs s'effectue bien en temps réel. Lorsque nous manipulons l'interface, nous ne percevons pas de temps de latence.

Gamme de pigments
On considère que le test est réussi, si l'utilisateur peut choisir 5 pigments ou plus pour le mélange de couleurs à partir de l'interface graphique.

Test validé car l'utilisateur peut choisir entre 11 pigments.

4 Programmes annexes

Au départ, les données que nous avions sur les pigments (les spectres d'absorption et de diffusion) et les lumières étaient stockées sous la forme de fichiers .cpp. Afin de pouvoir charger les données plus facilement dans notre application, nous avons créé un petit programme en C++ qui associé un script bash transforme ces données dans des fichiers .xml, le format que nous avons défini.

Voici les lignes de commande du programme C++ (createxml.cpp) :
./createxml pigment_absorption.cpp pigment_diffusion.cpp pigment.xml
./createxml lumiere.cpp lumiere.xml

Le script c++ n'est pas vraiment robuste car c'est un programme de lecture de fichier. Par conséquent il faudra faire en sorte que les fichiers d'entrées soient similaires aux fichiers c++ fournis.

5 Conclusion

Pour aborder cette conclusion, nous allons parler de ce qui est améliorable et de ce qui reste à faire.

Ce qui saute aux yeux c'est la faible quantité de tests. Il faudrait donc poursuivre les tests en profondeur sur la validité des données, bien qu'elles soient cohérentes et qu'elles ont été vérifiées logiquement après correction de bug. On peut également imaginer une comparaison entre nos valeurs obtenues et celles d'un autre groupe de recherche sur ce sujet.

L'implémentation GPU n'a pas pu être implémentée par manque de temps. C'est celle qui utilise 8 échantillons pour le calcul du mélange. Nous avons abandonné l'utilité du calcul sur GPU avec les 101 échantillons après réunion avec notre client. Dans l'avenir il faudra reprendre les travaux sur la quadrature gaussienne afin d'extraire les 8 meilleurs échantillons des spectres (parmi les 101) en fonction de chaque lumière incidente pour poursuivre sur l'implémentation GPU.

Nous pensons que notre point fort sur le projet est la réutilisabilité de notre code. Nous avons fait en sorte d'avoir un bon découpage et des interfaces intésressante pour de futures réutilisations. Nous avons aussi bien détaillé le rôle de chaque composant et fonction dans le manuel d'utilisation et avons utilisé au maximum des noms de fonction et de variable en fonction de leur utilité.

Certaines des fonctionnalités EF3 prévues au départ n'ont pas pu être respectées car elles supposaient que l'on n'ait pas de données spectrales sur les pigments. Elles ont donc été supprimées de nos priorités car peu intéressantes pour le client et la validité des données.

Pour conclure, ce projet est en partie une réussite pour nous car nous avons réussi à implémenter la plupart des fonctionnalités et à obtenir des résultats de mélange cohérents tout en ayant compensé le travail des personnes qui ont quitté le projet.

Deuxième partie

Manuel d'utilisation

6 Architecture du projet

6.1 Répertoire & fichiers

Voici ce que possède le répertoire du projet :

— build : C'est le répertoire de compilation du projet (à clean pour tout recompiler).

— createxml : Dans ce répertoire vous trouverez le programme c++ et le script bash que nous avons utilisé pour la génération des fichiers XML des pigments et des lumières.

— include : C'est le répertoire des headers (.h).

— lib : Répertoire des librairies externes que l'on utilise (dans notre cas glm).

— lumieres : Répertoire des fichiers lumières au format XML.

— pigments : Répertoire des fichiers pigments au format XML.

— shader : Répertoire des shaders utilisables par l'application (actuellement vide car l'implémentation GPU n'a pas été réalisée).

— src : Liste des fichiers sources (.cpp).

— CMakeLists.txt : Fichier de configuration du projet.

6.2 Cmake

Pour que l'application puisse être compilée correctement, il est nécessaire d'avoir une version cmake d'au minimum 2.8 ainsi qu'une version Qt 4 et OpenGL 4.3.

7 Prise en main de l'exécutable

Nous avons essayé de concevoir l'interface pour qu'elle soit la plus intuitive possible. La couleur affichée par défaut de l'application est celle du pigment : "Alizarin Crimson" (rouge). Les calculs se font par défaut sur cpu (bien qu'on ne puisse pas encore changer sur gpu car cette partie n'est pas implémentée).

Pour démarrer sur un mélange de pigment, il faut :

- 1 : Cliquer sur le bouton pigment 1 jusqu'à trouver le pigment que l'on souhaite.
- 2 : Cliquer sur le bouton pigment 2 jusqu'à trouver le pigment que l'on souhaite.
- 3 : Ajuster la concentration en utilisant le slider à disposition jusqu'à avoir la concentration désirée.
- 4 : Contempler le résultat produit à chaque mouvement du slider en bas de l'interface.

L'utilisateur peut à tout moment exporter le pigment résultant du mélange au format XML grâce au bouton "exporter". À l'heure où ces lignes sont écrites, cette fonctionnalité n'est pas implémentée mais elle le sera lors de la soutenance du projet.

8 Descriptif des informations de l'exécutable

En rouge les numéros des différentes zones possédant de l'information :

- 1 : Nom du pigment 1 + couleur.
- 2 : Nom du pigment 2 + couleur.
- 3 : Taux de concentration du pigment 2 inversement proportionnel au pigment 1.
- 4 : Dernière couleur calculée.
- 5 : Valeurs XYZ et RGB de la dernière couleur calculée.

9 Descriptif du code

Dans cette partie nous allons faire une description des différentes classes de notre projet et les fonctions associées à chacune d'elles.

9.1 Rappel diagramme de classe de la conception

Dans l'annxe A, vous trouverez le diagramme de classe initialement prévu dans le rapport de conception. Au final nous avons suivi à la lettre ce diagramme lors de notre implémentation du projet.

9.2 Main

La classe main a pour fonction de créer la MainFrame qt du projet. Dans notre cas, la MainFrame qt de notre projet est représentée par notre classe Mixeur. La classe main instancie donc la classe Mixeur.

9.3 Spectre

La classe Spectre fait partie du groupe des classes qui servent à la structuration des données du projet. Elle a pour mission de stocker en mémoire et d'interfacer les différentes données liées à un spectre. Dans notre cas les données sont :

- une liste de longueurs d'onde
- une liste de valeurs d'amplitude

Les valeurs d'amplitude sont à correspondre avec les longueurs d'ondes en utilisant la même indexation. Au niveau de l'interface, la classe propose 3 constructeurs (défaut, copie et paramétré). Elle propose aussi des accès en lecture & écriture des données et possède une fonction d'affichage.

9.4 Pigment

La classe Pigment fait elle aussi partie du groupe des classes qui servent à la structuration des données du projet. Elle a pour mission de stocker en mémoire et d'interfacer les différentes données liées à un pigment. Dans notre cas les données sont :

- Un label (nom du pigment)
- Un spectre d'absorption
- Un spectre de diffusion

La classe propose deux constructeurs (défaut et paramétré). Elle permet l'accès en lecture & écriture des données. Les spectres doivent posséder 101 valeurs de longueurs d'onde et d'amplitudes. Elle permet aussi l'affichage des données.

9.5 Light

La classe Light fait elle aussi partie du groupe des classes qui servent à la structuration des données du projet. Elle a pour mission de stocker en mémoire et d'interfacer les différentes données liées à une lumière. Dans notre cas les données sont :

- Un label (nom de la lumière)
- Un spectre d'amplitude
- Un coefficient de normalisation (k)

Le coefficient de normalisation k est calculé de façon à normaliser les composantes trichromatiques par rapport aux fonctions de color-matching et par rapport à la lumière réfléchie. La classe propose deux constructeurs(défaut, paramétré). Elle permet l'accès en lecture & écriture de ses données. Elle permet aussi l'affichage des données.

9.6 RenderingWidget

Cette classe hérite de QGLWidget, elle est donc présente dans l'interface et instanciée une seule et unique fois. Elle s'occupe principalement d'interfacer tout le code utile et spécialisé dans le calcul du mélange de couleurs selon la théorie de Kubelka-Munk.

Les données que la classe stocke sont directement liées aux fonctions qu'elle possède. Voici donc les différentes fonctions qu'elle propose :

- initializeGL() : Cette fonction doit impérativement s'être exécutée avant toute utilisation de l'application. Elle crée les différents pipeline (cpu & gpu) permettant de rediriger le calcul. Elle instancie et utilise un parseur xml qui lui permet de récupérer (à partir des fichiers xml du projet) et de stocker une liste de pigments et de lumières (à disposition).
- paintGL() : Cette fonction permet de calculer la couleur résultante du mélange en envoyant toutes les données nécessaires sur le pipeline (cpu ou gpu). Elle affiche ensuite cette couleur sur le widget QT.
- getResultat() : Cette fonction permet de récupérer la couleur résultante du mélange sous un format QColor. Ce format est utilisé pour l'affichage d'une couleur sur un composant QT. Attention ce format tronque les données float en int.
- getResultatFloatPrecision() : Cette fonction permet de récupérer la couleur résultante du mélange sous un vecteur x,y,z de float.
- getPigmentsLabels() : Cette fonction permet de récupérer la liste des labels des pigments chargés en mémoire.
- getXYZ() : Cette fonction permet de récupérer les valeurs de la couleur résultante du mélange dans l'espace colorimétrique XYZ.
- setLabel_pigment1() : Cette fonction permet de mettre à jour le pigment 1 servant dans le mélange.
- setLabel_pigment2() : Cette fonction permet de mettre à jour le pigment 2 servant dans le mélange.
- setSlider_concentration() : Cette fonction permet de mettre à jour le rapport de concentration du mélange. Les valeurs doivent être comprises en 0 et 1. Cette concentration est exprimée en pourcentage du pigment 1.

La classe possède aussi 2 fonctions privées permettant de factoriser un peu le code. Ces fonctions récupèrent l'adresse d'un pigment ou d'une lumière à partir d'un label.

9.7 ConfigXML

La classe configXML sert de parseur. Elle permet de lire et de stocker en mémoire les données au format XML du projet (définies dans le rapport de conception). Pour utiliser les données XML dans notre programme Qt, nous adaptons à nos besoins les fonctions définies dans le module QtXml. Voici les différentes fonctions que notre parseur propose :

- loadFilesFromDirectory() : Cette fonction permet de charger tous les fichiers XML d'un répertoire. Elle est appelée une fois pour charger tous les fichiers pigments du répertoire pigments et une deuxième fois pour charger tous les fichiers lumières du répertoire. lumieres.
- loadDataFromFile() : Cette fonction permet de charger les données d'un seul fichier XML.
- parsePigment() : Cette fonction permet de récupérer les données d'un fichier pigment (label, liste de longueurs d'onde, liste d'amplitudes d'absorption et de diffusion) au format XML.

- parseLight() : Cette fonction permet de récupérer les données d'un fichier lumière (label, coefficient k, liste de longueurs d'onde, liste d'amplitudes) au format XML.
- parseNodeLabel() : Cette fonction stocke la valeur comprise entre des balises <label> dans un QString.
- parseNodeAmplitudePigment() : Cette fonction récupère les valeurs d'un fichier .xml dont la racine est <pigment>. À partir des données récupérées, la fonction instancie un nouveau pigment et l'ajoute dans un vector qui contiendra tous les pigments dont nous disposons.
- parseNodeAmplitudeLight() : Cette fonction récupère les valeurs d'un fichier XML dont la racine est <lumiere>. À partir des données récupérées, la fonction instancie une nouvelle lumière et l'ajoute dans un vector qui contiendra toutes les lumières.
- parseNodeCoefK() : Cette fonction stocke la valeur comprise entre des balises <k> dans un float.

Pour que l'application fonctionne, il faut impérativement que les fichiers pigments et lumières respectent les formats suivants :

```
<pigment>
<label></label>
<amplitude wavelength=" ">
      <absorption></absorption>
      <scattering></scattering>
</amplitude>
</pigment>
```

(c) Format d'un fichier pigment

```
<lumiere>
       <label></label>
       <k></k>
       <amplitude wavelength=" "></amplitude>
</lumiere>
```

(d) Format d'un fichier lumière

Si les fichiers XML pigments et lumières ne respectent pas ce format, il faudra modifier et ou ajouter des fonctions dans la classe ConfigXML.

9.8 Pipeline

Cette classe est une classe abstraite. Elle sert principalement d'interface. Elle possède deux fonctions qui doivent être implémentées dans toutes les classes héritant (ou implémentant) de cette interface. Ces deux fonctions sont :

- run_8_samples() : Cette fonction est appellée dès lors que l'on souhaite prendre en compte seulement 8 échantillons des pigments pour le calcul du mélange de couleur.
- run_full_samples() : Cette fonction est appellée dès lors que l'on souhaite prendre en compte tous les échantillons des pigments pour le calcul du mélange de couleur.

Les classes héritant de cette interface mettront donc en place différentes opérations pour aboutir au calcul de la couleur mélange des deux pigments sélectionnés.

9.9 Pipeline_cpu

Cette classe hérite de la classe Pipeline. Elle implémente donc les fonctions de l'interface. La classe Pipeline_cpu utilisera la classe Utils_Kubelka pour déterminer la couleur du mélange. Elle calculera donc :

- le spectre de mélange des pigments
- le spectre de réflectance
- le spectre de la lumière réfléchie
- la couleur mélange dans l'espace XYZ
- la couleur au format RGB après clipping et la correction gamma

9.10 Pipeline_gpu

Cette classe hérite de la classe Pipeline. Elle implémente donc les fonctions de l'interface qui permettent de mettre en place les différentes combinaisons d'opérations pour le calcul de la couleur du mélange. Cette classe ne possède pas encore d'implémentation.

9.11 Utils_Kubelka

La classe Utils_Kubelka est une classe statique. Elle n'a pas d'instanciation. Elle propose diverses opérations de calcul rentrant en compte dans le mélange de couleur selon la théorie de Kubelka-Munk. Voici les opérations qu'elle propose :

- compute_melange() : Cette fonction calcule le spectre mélange à partir des spectres d'absorption et de diffusion des pigments à mélanger.
- compute_reflectance_melange() : Cette fonction calcule le spectre de réflectance à partir du spectre de mélange.
- compute_lumiere_reflechie() : Cette fonction calcule le spectre de la lumière réfléchie à partir du spectre de réflectance et du spectre de la lumière incidente.
- convertXYZ() : Cette fonction calcule la couleur résultante du mélange dans l'espace colorimétrique XYZ à partir du spectre de la lumière réfléchie, de la lumière incidente et des fonctions de color-matching définient par le CIE 1964.
- convertRGB() : Cette fonction calcule la couleur résultante du mélange dans l'espace colorimétrique RGB (espace linéaire) à partir de sa valeur dans l'espace XYZ.
- convert_sRGB() : Cette fonction calcule la couleur résultante du mélange dans l'espace colorimétrique sRGB (espace non linéaire) à partir de sa valeur dans l'espace RGB.
- correctionGamma() : Cette fonction effectue une correction gamma sur la couleur RGB passée en paramètre.
- clipping() : Cette fonction permet de clamper les valeurs RGB de la couleur résultante du mélange entre 0 et 1 pour l'affichage.

Cette classe possède en dur les fonctions de color-matching définient par le CIE 1964 ainsi que la matrice de conversion XYZ vers RGB.

9.12 Shader

Cette classe est utilisée dans le cadre de l'implémentation GPU. Elle est censée être utilisée par la classe Pipeline_gpu. Cette classe utilise OpenGL. Elle s'occupe de créer le vertex shader et le fragment shader de la pipeline OpenGL de rendu, de les compiler, de les linker et de les utiliser. La classe propose seulement un constructeur par défaut. Pour créer un couple vertex/fragment shader on utilise la fonction init() qui va lire le contenu des fichiers correspondant, créer, compiler et linker les shaders à partir des fichiers (path) passés en paramètre. Pour utiliser ces shaders, on appellera la fonction use().

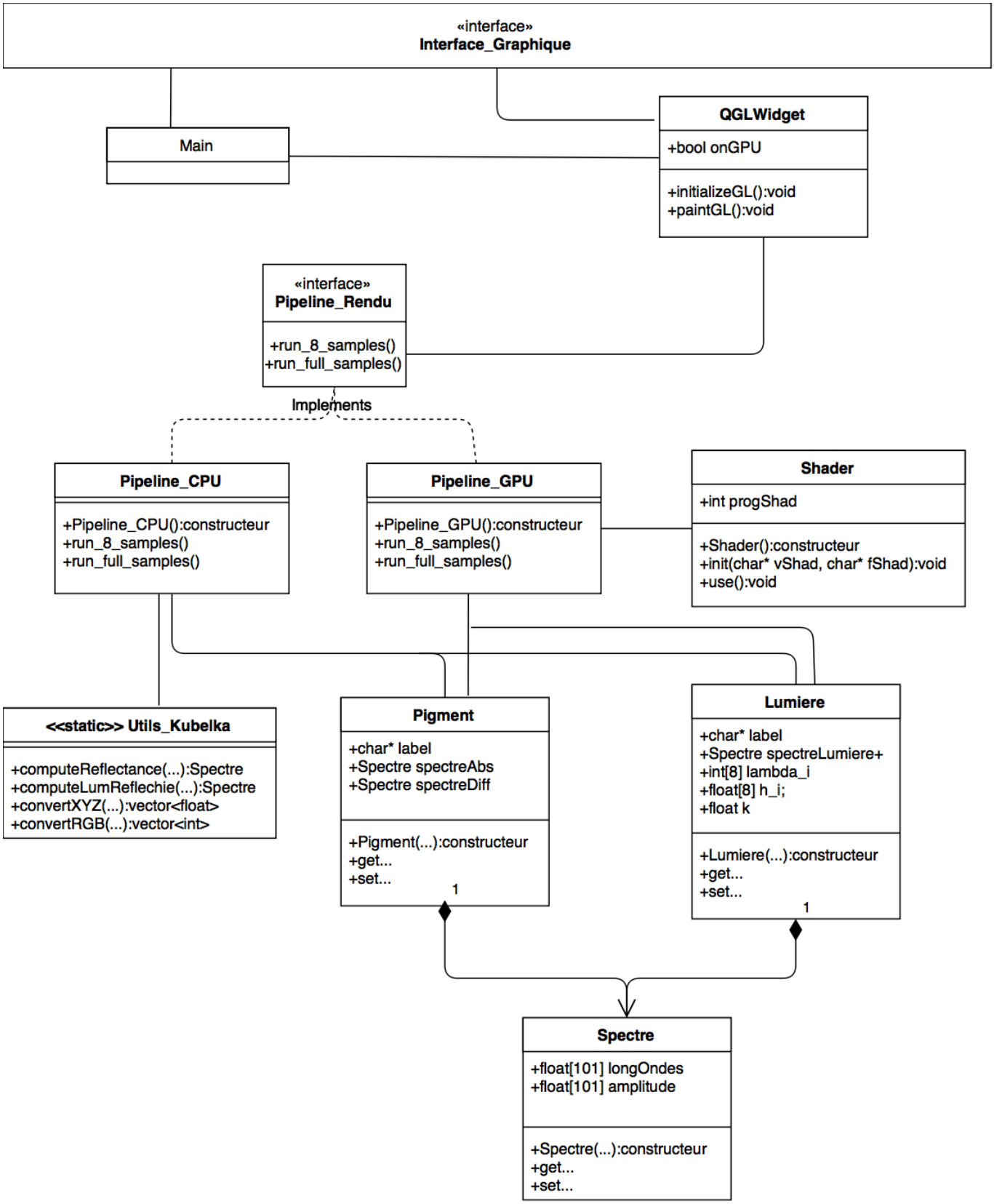
9.13 Mixeur

Cette classe représente la MainFrame qt de notre application. Elle crée l'interface (conçue avec QtDesigner) et gère les slots associés. Elle possède deux fonctions privées permettant la factorisation de code redondant. Voici les slots (auto-connectés) que cette classe propose :

- on_pigment_1_clicked() : Cette fonction va configurer la classe RenderingWidget pour que celle-ci calcule la couleur du pigment 1 pour ensuite l'afficher à l'aide d'une QPalette sur le bouton pigment 1. Pour le calcul, elle force la classe RenderingWidget à se repeindre.
- on_pigment_2_clicked() : Cette fonction va configurer la classe RenderingWidget pour que celle-ci calcule la couleur du pigment 2 pour ensuite l'afficher à l'aide d'une QPalette sur le bouton pigment 2. Pour le calcul, elle force la classe RenderingWidget à se repeindre.
- on_cpu_gpu_clicked() : Cette fonction n'est pas encore implémentée. Elle permettra de configurer la classe RenderingWidget pour que le calcul se déporte soit sur gpu soit sur cpu.
- on_slider_valueChanged() : Cette fonction va configurer l'unique instance de la classe RenderWidgntet pour le calcul du mélange de couleur en renseignant les pigments courants à mélanger, la concentration courante. Elle s'occupe également de mettre à jour les différents labels permettant l'affichage des valeurs RGB et XYZ de la couleur résultante.

Au niveau de ces attributs, la classe possède la liste des labels des différents pigments disponibles pour le mélange ainsi que l'adresse de l'interface "ui".

A Annexe


«interface»
Interface_Graphique
Main
QGLWidget
+bool onGPU
+initializeGL():void
+paintGL():void
«interface»
Pipeline_Rendu
+run_8_samples()
+run_full_samples()
Implements
Pipeline_CPU
+Pipeline_CPU():constructeur
+run_8_samples()
+run_full_samples()
Pipeline_GPU
+Pipeline_GPU():constructeur
+run_8_samples()
+run_full_samples()
Shader
+int progShad
+Shader():constructeur
+init(char* vShad, char* fShad):void
+use():void
<<static>> Utils_Kubelka
+computeReflectance(...):Spectre
+computeLumReflechie(...):Spectre
+convertXYZ(...):vector<float>
+convertRGB(...):vector<int>
Pigment
+char* label
+Spectre spectreAbs
+Spectre spectreDiff
+Pigment(...):constructeur
+get...
+set...
1
Lumiere
+char* label
+Spectre spectreLumiere+
+int[8] lambda_i
+float[8] h_i;
+float k
+Lumiere(...):constructeur
+get...
+set...
1
Spectre
+float[101] longOndes
+float[101] amplitude
+Spectre(...):constructeur
+get...
+set...

Définition des impacts des risques sur le projet

Impact	Description
Critique	Le risque peut compromettre une ou plusieurs fonctionnalités de notre application.
Important	Le risque peut engendrer un retard important lors de la livraison de nos livrables.
Faible	Le risque peut nécéssiter une réorganisation des moyens accordés aux différentes tâches du projet.
Nul	Le risque n'a pas d'impact sur le projet.

Définition des probabilités des risques sur le projet

Probabilité	Description
Très forte	Est quasi certain
Forte	A de fortes chances de se produire
Moyenne	Peut apparaître
Faible	Improbable

Définition des niveaux de priorité des exigences

Niveau	Description
1	Cette exigence ne peut pas être ignorée sans quoi la recette ne pourra pas être validée.
2	Cette exigence aura un impact important lors de la validation de la recette, elle doit être traitée avant les exigences de priorité inférieure.
3	Cette exigence aura un impact limité sur la validation de la recette. Elle est demandée mais reste secondaire par rapport aux exigences de niveau supérieur.

Définition des codes couleur pour les exigences

Couleur	Description
	L'exigence n'est pas respectée. Le non-respect de cette exigence a un impact critique sur la validation de la recette.
	L'exigence n'est pas respectée. Le non-respect de cette exigence a un impact important sur la validation de la recette.
	L'exigence n'est pas respectée. Le non-respect de cette exigence a un impact limité sur la validation de la recette car l'exigence est secondaire.
	L'exigence est respectée.

Références

[1] William Baxter, Jeremy Wendt, and Ming C Lin. Impasto : a realistic, interactive model for paint. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 45–148. ACM, 2004.