

**UNIVERSITÉ DE LIÈGE**  
**Faculté des Sciences Appliquées**  
**Département d'Electricité, Electronique et Informatique**



## **INFOGRAPHIE - Sujet 1 : Application de textures**

Frédéric Gendebien et Aubry Mockel

Année académique 2009 - 2010

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Les modifications apportées</b>	<b>4</b>
2.1	<i>materials.h</i> . . . . .	4
2.2	<i>models.h</i> et <i>models.cc</i> . . . . .	4
2.3	<i>scene.cc</i> . . . . .	5
2.4	Fichiers ajoutés . . . . .	5
<b>3</b>	<b>Résultats</b>	<b>6</b>
3.1	Générateur d'images . . . . .	6
<b>4</b>	<b>Etat des lieux, conclusion</b>	<b>7</b>
4.1	Les objets . . . . .	7
4.2	L'application de texture . . . . .	8

**Table des figures**

1	Différentes textures générées par notre programme . . . . .	7
2	Résultats obtenus . . . . .	8

## 1 Introduction

Ce travail consistait à appliquer des textures à deux dimensions sur des objets en trois dimensions dans le cadre du projet de *Ray-Tracing*. À la base, rien n'était prévu pour de telles fonctionnalités, les objets possédaient une couleur unie, et rien ne permettait de connaître la couleur de l'objet en un point donné.

Notre premier travail a donc été de trouver et d'implémenter correctement pour chaque objet dont nous voulions changer la texture son équation paramétrique. En effet, c'est elle qui va nous permettre d'associer à un point de l'objet (dans l'espace à trois dimensions), un point de la texture (dans le plan à deux dimensions).

Nous avons naturellement commencé par les sphères, qui étaient au départ les seuls objets implémentés et fonctionnels du projet. En l'état actuel du projet, l'application est définie dans le programme et ne peut pas être changée. Une amélioration ultérieure pourra être de permettre à l'utilisateur de spécifier une fonction de transformation des coordonnées  $u$  et  $v$ , après notre mapping. Cela permettra de choisir le type d'application effective de la texture.

Nous nous sommes ensuite intéressés au cylindre. Ici encore, nous pouvons utiliser la paramétrisation naturelle de l'objet pour appliquer notre texture. Il faut noter que pour d'autres objets ne disposant pas de ce type de paramétrisation, il faut alors changer radicalement de méthode. En effet, ces objets seront implémentés sous forme d'assemblage de triangles, il faudra alors utiliser la paramétrisation de cette multitude de triangle, un par un, chaque fois qu'un rayon intersecte l'un d'entre eux.

## 2 Les modifications apportées

### 2.1 *materials.h*

De base, le programme permettait uniquement d'attribuer une couleur unique aux objets, et le `GMaterial` défini se contentait de renvoyer la couleur à chaque appel de la fonction `getColor()`. Une texture ne pouvant pas se résumer à une seule couleur, nous avons dû apporter des modifications à ce niveau. Tout d'abord, nous avons ajouté un constructeur permettant de spécifier une texture au matériau, sous la forme d'un tableau en deux dimensions de couleurs. Le type de ces dernières est celui défini de base dans le projet. Deux arguments supplémentaires sont requis, il s'agit des dimensions du tableau. Au cas où le constructeur de base, avec une couleur unie, est utilisé, ces dimensions seront mises à -1.

Ensuite, nous avons ajouté une fonction `getColor` qui prend deux coordonnées en argument,  $u$  et  $v$ , et qui renvoie la couleur correspondante dans le tableau représentant la texture. En effet, le matériau est totalement indépendant de l'objet qui l'utilise, ce n'est donc pas à lui d'effectuer une éventuelle transformation de coordonnées. Ces transformations seront effectuées dans les objets, puisque chacun d'entre eux devra utiliser des équations différentes, et devra appliquer un mapping différent. Nous avons laissé la fonction `getColor()` de base, qui renvoie la couleur unie, pour laisser cette possibilité à l'utilisateur.

Ces modifications n'ont donc pas d'influence sur le code actuel du projet, et permettront pour la suite du développement une modularité et une souplesse certaine au niveau de l'implémentation des textures.

### 2.2 *models.h* et *models.cc*

Ces fichiers contiennent les déclarations, définitions, et implémentations des différents objets disponibles dans le projet. C'est ici que nous allons devoir transformer les coordonnées des points de l'objet (en 3D) en coordonnées 2D à aller chercher dans la texture. Ainsi, en implémentant cette association ici, nous avons directement à notre disposition les informations nécessaires sur les objets, et nous n'avons pas besoin de créer de multiples fonctions d'accès.

Nous retrouvons ici le même principe que dans *materials.h*, à savoir la fonction `getColor()`. Nous avons modifié la fonction qui va demander en argument les coordonnées du point en trois dimensions. Cela ne changera pas le comportement de la fonction de base. En effet, nous commençons par tester les dimensions du tableau. Si les dimensions sont égales à -1, il s'agit d'un objet à couleur unie, et nous appelons la fonction `getColor` du matériau sans argument. Dans le cas contraire, nous avons une texture, et nous devons trouver le point de cette dernière qui doit être appliqué au point spécifié.

A ce niveau, chaque fonction *getColor* des différents objets sera différente. En effet, c'est ici que va se trouver la transformation du point à trois coordonnées en point à deux coordonnées, que nous utiliserons pour récupérer la couleur du matériau.

### 2.3 *scene.cc*

La modification effectuée dans ce fichier est triviale, et se situe dans la fonction *shade*. La fonction doit récupérer la couleur de l'objet. Nous avons donc remplacé l'appel à *getColor()* par un appel à notre nouveau *getColor*, en passant en argument les coordonnées du point d'intersection avec le rayon. Les calculs seront donc effectués avec comme base les couleurs de la texture.

### 2.4 Fichiers ajoutés

Pour représenter la texture à appliquer, nous utilisons des images au format PPM3 très simple à lire. En effet, ce format contient un header de quatre éléments : *le type de format, la largeur de l'image, la hauteur de l'image et la plus haute valeur de couleur possible*. Nous avons pour cela créé une nouvelle structure de classes :

1. *ImageParser* est une classe abstraite représentant tout ce qu'il est nécessaire de disposer après la lecture d'un format quelconque. C'est-à-dire la table des couleurs relatives à chaque pixel et les dimensions de la texture. Ainsi, la classe est prête à être héritée pour n'importe quel parser qui permettrait de générer des images dans le format choisi par l'utilisateur.
2. *PPMParser* est la classe qui hérite de la class *ImageParser* chargée de lire un fichier au format PPM3 et d'en extraire l'information utile.

Nous avons également créé un générateur de fichier PPM3 en *Java* pour tester différentes textures. Ce programme génère des damiers de dimensions données (largeur et hauteur de l'image et largeur et hauteur des rectangles) avec un nombre de couleurs donné.

## 3 Résultats

### 3.1 Générateur d'images

Notre générateur d'images permet de créer toutes sortes de motifs sur base de rectangles de largeur et de hauteur données, voir Figure 1. Son fonctionnement est très simple, il suffit d'exécuter le programme avec le paramètre "-h" pour obtenir le menu d'aide. Par exemple, pour créer le drapeau belge de 210 pixels de haut pour 300 de large, la commande est la suivante :

```
java PPMCreator -f ../image.ppm -WHxy 300 210 100 210 -c 3
0 0 0 255 255 0 255 0 0
```

Ou pour créer un damier en noir et blanc :

```
java PPMCreator -f ../image.ppm -WHxy 256 256 32 32 -c 2
0 0 0 255 255 255
```

Et encore pour créer un motif spécial :

```
java PPMCreator -f ../image9.ppm -WHxy 256 256 32 32 -c 8
0 0 0 0 0 0 255 0 0 0 0 0
0 0 0 255 255 0 0 0 0 0 0 0
-s 5
```

En ce qui concerne l'application de textures, le résultat est assez satisfaisant malgré les déformations. Ce problème peut être facilement régler par une fonction modifiant le mapping entre point 3D et point 2D. Nous pouvons voir quelques résultats avec des textures générées par notre créateur d'images Figure 2.



FIGURE 1 – Différentes textures générées par notre programme

## 4 Etat des lieux, conclusion

Nous allons terminer ce rapport par un état des lieux du projet. Nous espérons que cela permettra aux futurs contributeurs d'avoir une vision plus claire de ce qu'il reste à faire et à corriger pour améliorer le programme.

Tout d'abord, de manière générale, la structure du programme n'a pas été conçue pour permettre facilement à plusieurs personnes et/ou plusieurs groupes de travailler sur le projet sans devoir s'arracher les cheveux sur des détails d'implémentation relatifs à une partie que l'on n'a pas codé soi-même. En l'état, les différentes parties fonctionnent plutôt de manière isolée sans jamais penser qu'une autre partie pourrait avoir besoin de quelque chose. Nous avons tenté d'apporter quelques améliorations, mais repenser la structure du code de base pourrait être très profitable pour la lisibilité du programme.

Plus particulièrement, au niveau des textures, on peut séparer les améliorations à apporter en deux parties : l'interaction avec les objets, et l'application des textures en elle-même.

### 4.1 Les objets

Pour l'instant, seules les sphères sont pleinement fonctionnelles, s'affichent, et supportent l'application d'une texture. Nous manquons cruellement d'informations



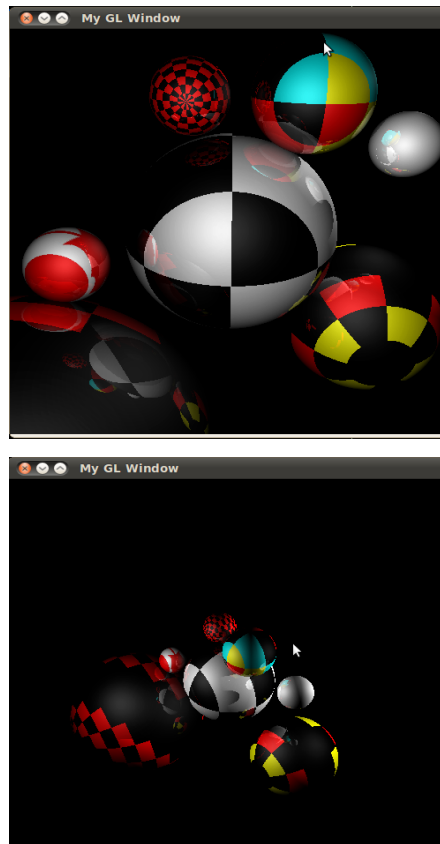


FIGURE 2 – Résultats obtenus

pour les autres objets, en terme de paramètres principalement. Dans l'idéal, un objet devrait être capable de mettre à disposition son équation paramétrique s'il en a une, sinon il devrait pouvoir donner les coordonnées du triangle qui contient exactement un point donné. Cela permettrait d'implémenter une application de texture pratiquement indépendante des objets.

## 4.2 L'application de texture

Plusieurs améliorations sont envisageables concernant l'application de texture, nous en avons déjà évoqué dans les sections précédentes. Une première modification pourrait être de pouvoir appliquer une texture sur une surface composée de triangles. Actuellement, nous appliquons une texture suivant une paramétrisation naturelle de l'objet, mais ce n'est pas possible pour tous les objets. Cette modification permettrait d'implémenter l'application de texture pour n'importe quel objet, quelle que soit sa forme, sa taille, ou même ses éventuelles équations paramétriques, dont nous n'aurions plus besoin.

Lors d'une application de texture sur un objet, la texture subit très souvent des déformations dues à la forme de l'objet. Un moyen de contrer ces déformations

est de permettre à l'utilisateur de spécifier une fonction de transformation qui agirait sur les coordonnées  $u$  et  $v$ , obtenues à partir des coordonnées d'un point de l'objet. L'exemple typique est l'application d'une carte du monde sur un sphère. Le résultat dépendra directement de la projection utilisée pour dessiner la carte.