CFPT Ecole d'informatique - Technicien ES en informatique

Travail de semestre inter-degré 2016-2017

Cube à Led
Documentation technique

Elèves :
M. Kevin Amado
M. Alan Devaud
M. Grégory Mendez

Enseignants :
M. Denis Carbone
M. Nicolas Wanner

Version 1.0 du
20 mars 2017

1 Résumé

Table des matières

2 Introduction

Notre projet consiste à améliorer le projet Cube de M. Aubert Jonathan dans le cadre de l'atelier Technicien, qui regroupe les deuxièmes années avec les premières. Nous sommes trois à travailler ensemble. L'objectif est de reprendre le travail réalisé en ajoutant une vue 3D. Notre application doit pouvoir gérer la couleur des LED. La gestion d'animation est aussi demandée.

3 Cahier des charges

3.1 Sujet

Création d'une interface pour la gestion d'un cube à LED.

3.2 But

Créer une interface graphique permettant à l'utilisateur de gérer le cube à LED. Cette interface est un logiciel C# muni d'un cube 3D. L'utilisateur peut alors sélectionner les leds du cube et les modifier (allumer, éteindre, intensité). Le cube à LED - physique - se modifie en fonction des actions de l'utilisateur sur l'application.

3.3 Spécification

Le logiciel sera capable de :
* afficher un cube virtuel en 3D
* communiquer avec le cube
* sélectionner une face du cube virtuel
* modifier une LED du cube

3.4 Restriction

Le logiciel sera incapable de :
* générer un fichier ".cube"
* modifier la couleur des LEDS du cube (virtuelle et physique)
* gérer plusieurs cubes à LED en même temps
* modifier le programme du micro-contrôleur du cube à LED

3.5 Environnement

* Système d'exploitation : *Windows 7*
* Outil de développement : *Visual Studio*
* Langage de programmation : *C#*

3.6 Livrables

* Code source
* Documentation technique
* Journal de bord

3.7 Réddition

* 20 mars 2017 : code source
* 20 mars 2017 : documentation technique
* 20 mars 2017 : journal de bord
* 20 mars 2017 : présentation final

4 Analyse de l'existant

4.1 Représentation du Cube dans l'application

Le cube possède huit étages, chaque étage contient huit rangées qui, elles-mêmes contiennent huit LED. Le Cube à LED est représenté sous la forme d'un tableau à trois dimensions :

CubeLED[x][y][nbImage] = value

CubeLED : Le nom du tableau
x : Position d'une Led sur l'axe "x"
y : Position d'une Led sur l'axe "y"
nbImage : Numéro de l'image (animation)
value : Valeur stockée

4.1.1 Explication de la valeur stockée dans *"value"*

Ce sont les deux premiers paramètres qui sont utiles pour représenter la position d'une Led allumée, c'est donc grâce à la valeur stockée que nous indiquerons au micro-contrôleur quelle LED il va devoir allumer. On envoie donc une valeur entre 0 et 255, sous forme binaire. Dans cette chaîne binaire, chaque "1" représente une LED allumée et chaque "0" une LED éteinte.

Exemple 1 : CubeLED[0][0][0] = 128_{10} -> 10000000_2
-> La dernière Led de la ligne x = 0 / y = 0 est allumée.

Exemple 2 : CubeLED[6][4][0] = 170_{10} -> 10101010_2
-> Une Led sur deux est allumée sur la ligne x = 6 / y = 4.

Exemple 3 : CubeLED[2][7][0] = 0_{10} -> 00000000_2
-> Aucunes Leds sont allumées sur la ligne x = 2 / y = 7.

Exemple 4 : CubeLED[8][1][0] = 255_{10} -> 11111111_2
-> Toutes les Leds sont allumées sur la ligne x = 8 / y = 1.

Ce format est utilisé pour simplifier la communication avec le micro-contrôleur et la création de fichiers ".cube". Il est aussi important de prendre conscience de l'orientation du Cube en utilisant ce format.

Voici donc ci-dessous, un schéma représentant le tableau CubeLED. La face droite est l'équivalent des deux premiers index de notre tableau (*CubeLED[x][y]*) et les valeurs affichées sur cette dernière sont les éléments stockés à ces positions.

Le microcontrôleur analyse la valeur binaire convertie (voir axe Z) et allume une LED à chaque fois qu'elle rencontre un "1".

Note : Le tableau complet contient des animations, ce schéma représente une seule image qui pourrait être présente dans le tableau.

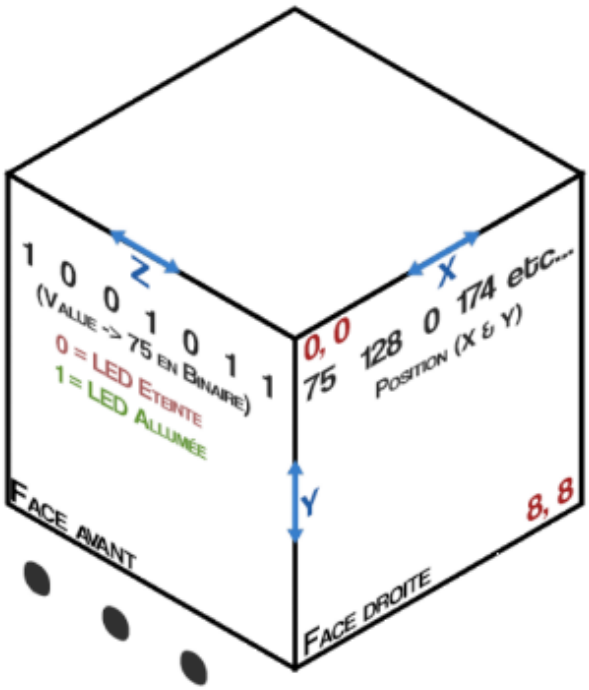

Figure 1 – Représentation 3D du tableau *CubeLED*

4.2 Format ".cube"

Le fichier cube est construit en trois catégories principales :

* En-tête du fichier (ASCII)
* Images du CubeLED
* Luminosité

Chaque partie du fichier est séparée par des séparateurs de début et fin. Pour indiquer le début d'une partie, on utilise "#", pour marquer la fin d'une partie on utilise "$" par convention.

Voici un schéma représentant la structure du fichier ".cube" :

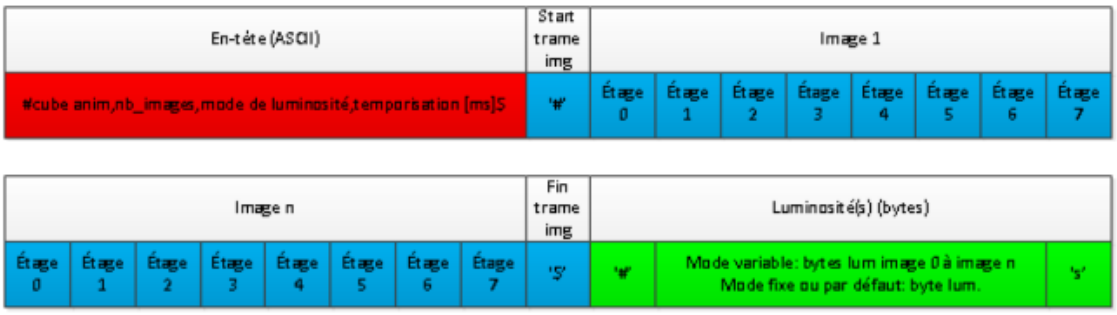

Figure 2 – Représentation d'un fichier du "*.cube*"

En-tête :	Contient les informations relatives à l'animation
Images :	Contient les informations sur la position des Leds allumées (par étages)
Luminosité :	Si le mode est variable, on indique la luminosité pour chaque image, sinon on indique la luminosité générale du cube.

Voici un schéma représentant un étage du cube, un étage contient huit lignes de leds :

FIGURE 3 – Représentation d'un étage du *CubeLed*

5 Analyse fonctionnelle

5.1 C#

Le logiciel sera développé en C#. Langage puissant et utilisé lors de la formation. Il va nous permettre de mettre en place une structure à notre programme en créant des classes et utilisant des bibliothèques.

5.2 Esquisse de l'interface

L'interface (Fig. 4 - page 8) contient les éléments essentiels pour "jouer" avec le cube. Au milieu, dans la partie verte claire, on retrouve le cube à LED matérialisé en 3D. L'utilisateur pourra sélectionner les leds du cube et les modifier grâce aux interactions que l'on trouve en dessous.

Dans le groupe boxe, *Options*, on trouve les informations de la led sélectionnée :
- *On* : allumer la led
- *Off* : étindre la led
- *Intensité* : l'intensité de la led
- *Changer couleur* : modifier la couleur de la led

Dans le second groupe boxe, *Général*, on retrouve les informations par apport à l'application :
- *Animation On* : indique qu'une animation sera gérée (plusieurs frame générée)
- *Animation Off* : indique que le cube sera une image fixe (une seule frame)
- *Play* : met en marche l'animation
- *Pause* : suspend l'animation
- *Stop* : arrête entièrement l'animation
- *Options - Nb image* : nombre d'image pour l'animation
- *Options - ms/image* : nombre d'image par milliseconde que le cube doit afficher

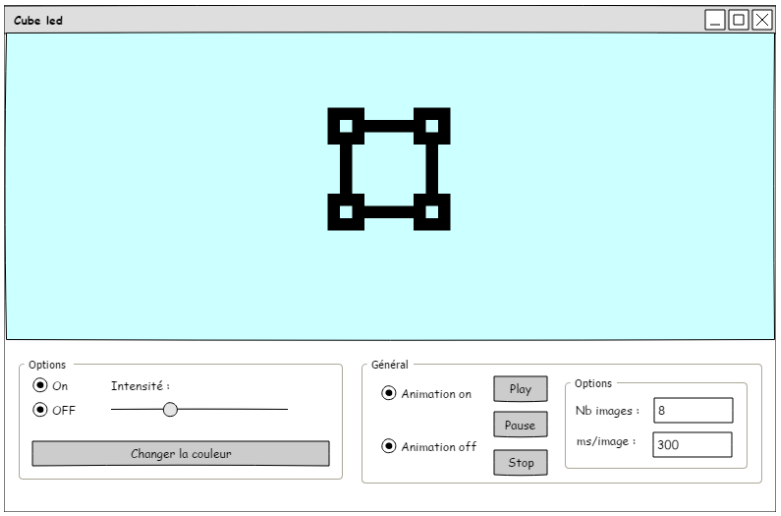

FIGURE 4 – Esquisse de l'interface du logiciel

5.3 *UsbLibraryCfptAdd*

UsbLibraryCfptAdd est une bibliothèque qui permet de communiquer sur une trame USB. Elle a été fournie avec le cube à LED pour que l'on puisse communiquer entre le cube à LED et le PC. Elle implémente certaines fonctionnalités telles que la détection d'un nouveau périphérique, la communication avec un périphérique donné, la déconnexion d'un appareil USB, etc.

Une bibliothèque créée par nous servira de sur-couche à celle-ci. La bibliothèque pourra alors transformer un format de données reçus de notre application, le transformer au format dont on a besoin. Puis, le format de sortie sera envoyé au cube par la connexion USB.

6 Analyse organique

6.1 Bibliothèque *CubeLedCommunicationLibrary*

Pour le processus de communication entre le cube à led et l'application, une bibliothèque C# a été créée. Cette bibliothèque est une sur-couche de *UsbLibraryCfptAdd*. Elle implémente les fonctionnalités qui permettent la communication spécifique entre le cube et l'application.

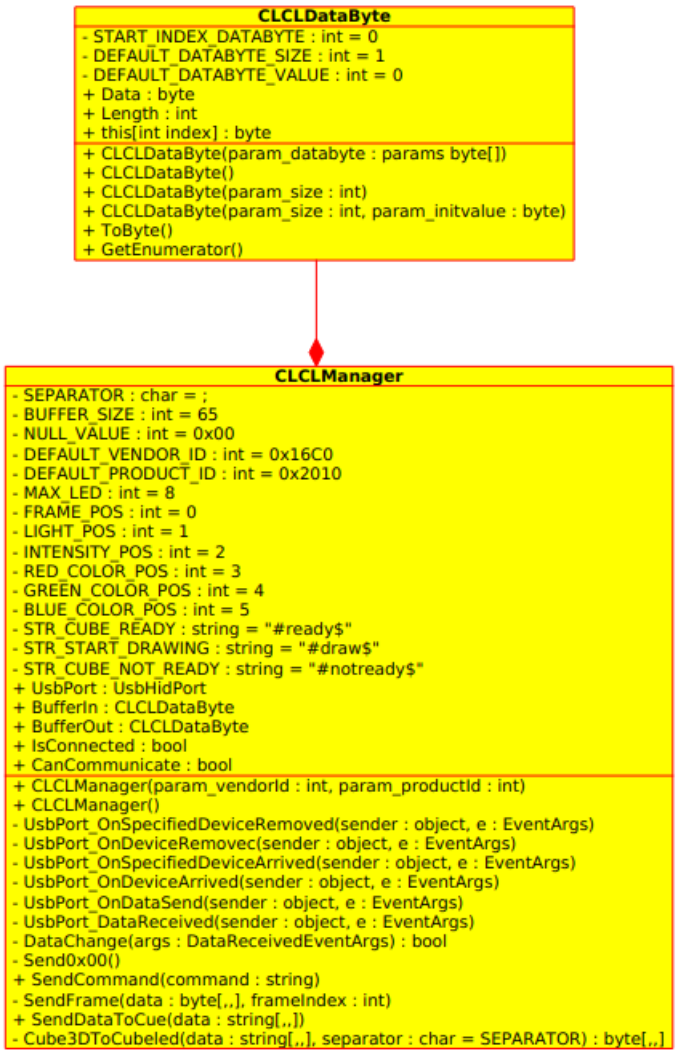

FIGURE 5 – Diagramme *UML* de la bibliothèque *CubeLedCommunicationLibrary*

6.2 Monogame

Monogame est une implémentation *Open Source* du framework Microsoft XNA 4. Le but de monogame dans notre projet est de faciliter la création et la gestion d'objet en 3D. Pour implémenter il suffit de télécharger sur le site "http ://www.monogame.net/downloads". Nous avons téléchargé la dernière version disponbile soit la version MonoGame 3.6.

Monogame n'était pas le seul choix possible, au contraire il existe par exemple unity qui permet la création et la gestion simple d'objet 3D (voir même plus simple que monogame), mais nous avons choisis monogame car, implémenter un projet monogame dans un projet windows form avait l'air pour nous beaucoup plus simple avec monogame (voir problèmes rencontrés).

6.3 Virtualisation du cube en 3D

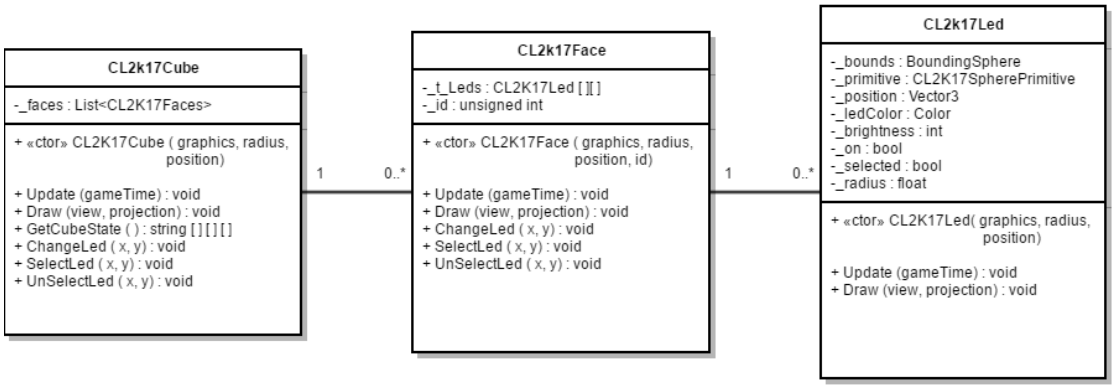

FIGURE 6 – Diagramme *UML* du cube 3D

1. **Classe CL2K17Led**

   Cette classe à deux fonctions enregistrer les données importantes des leds comme par exemple son état (on/off), sa luminosité et encore sa couleur. La deuxième fonction constitue à dessiner ces leds à l'aide d'une librairie télécharger sur "github.com", c'est de la que vient le type SpherePrimitive.

   Le booléan "*selected*" signifie que le focus de l'utilisateur est sur cette led ou non. Elle permet de changer la couleur ou non de la led en question.

   La variable de type BoundingSphere est utilisé pour l'intersection en picking, mais cette utilitée n'est pas fonctionnelle (voir problèmes rencontrés).

2. **Classe CL2K17Face**

   Une face est composée de 64 leds (classe CL2K17Led).

   Un "*id* est utilisé pour décaler le dessin de la face des autres faces.

   Une face peut changer l'état des leds, malheureusement la luminosité est une option qu'il manque, même si celle-ci est déjà implémentée sur la classe CL2K17led.

3. **Classe CL2K17Cube**

   Un cube contient 8 leds, et permet de renvoyer les données des leds dans un tableau de string 3D sous le format suivant : Frame ;State ;intensity ;Color ;

   — frame : numéro de l'image (exemple : 0)
   — State : Etat on/off de la led (exemple : true)

— intensity : la luminosité de la led en pourcentage (exemple : 55)
— Color : couleur de la led en RGB : (exemple : 65535)

Le tableau 3D contient un total de 512 variable string pour les 512 leds du cube.

6.4 Bibliothèque utilisée

Pour la création de sphère en 3D nous avons opté pour une bibliothèque *Open Source* disponible au lien suivant : https ://github.com/CartBlanche/MonoGame-Samples/tree/master/PerformanceMe

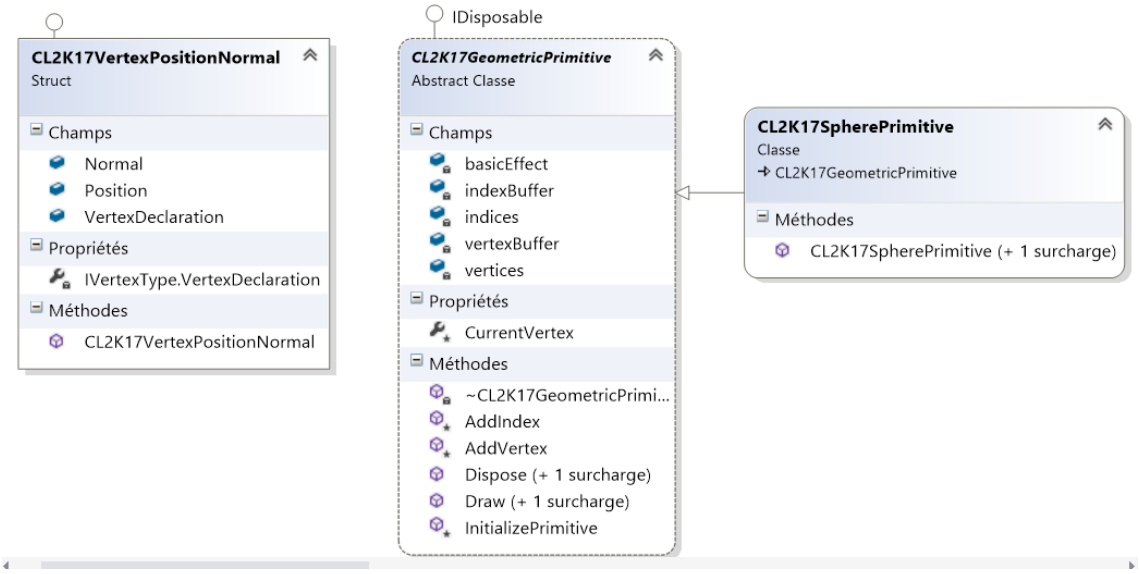

FIGURE 7 – Diagramme *UML* création sphere 3D

1. **structure CL2K17VertexPositionNormal**
   Pour créer une sphère en 3D, il faut des triangles, pour créer des triangles, il faut des points pour chaque sommet du triangle., c'est la que les vertex prennent tout leur sens. Cette structure aura pourra unique fonction de stocker les points de chaque triangles.
2. **Classe CL2K17GeometricPrimitive** Cette classe permet de dessiner n'importe quel model 3D.
3. **Classe CL2K17SpherePrimitive** Cette classe créer les triangles rejoignant chaque vertex pour créer une sphère.

7 Problèmes rencontrés

7.1 Implémentation du projet monogame en *Windows Form*

L'une des premières étapes de notre projet a été de définir, comment la creation des formes 3D allaient être créer. Plusieurs possibilitées se sont offertes comme créer la partie 3D sous Unity 3D ou sous monogame. Nous avons choisis d'utiliser monogame.

Le problème a été d'imbriquer le projet monogame dans un autre projet *Windows Form*. Mendez s'est occupé de cette tâche. Pendant un certain temps cela à fonctionné puis nous nous sommes aperçu que nous ne pouvions pas charger d'images 3D (.x , .fbx), ne pouvais pas être chargé avec le projet windows forme car *content* est nécessaire

7.2 Communication USB

Pour la communication avec le cube à led plusieurs problèmes ont été rencontrés.

7.2.1 compréhension code

Le plus gros des problèmes a été la compréhension du code source de l'application fournie avec le cube à led. En effet, son créateur était élèctronicien et certaine logique est différentes. L'interprètation se fut alors plus fastidieuse que pour d'autre code. Par exemple, pour l'enregistrement des leds allumées, il a utilisé un tableau tridimensionnel et stocke une valeur de 0 à 256. Cella lui permet de connaitre les leds qui doivent être étaintes et celle qui doivent être allumées (très astucieux).

7.2.2 Adaptation des données envoyées au MCU

Le second problème a été l'adaptation des données. La transformation ce fut difficile. Il a fallu ajuster la position des données par apport à la position réelle des leds et de celles du cube 3D. De plus, il fallait manipuler la modification des leds d'une ligne pour la transformer en une données entre 0 et 256.

7.3 Gestion de forme 3D

7.3.1 Compréhension de la 3D

7.3.2 Création d'une forme 3D

7.3.3 Gestion forme 3D

7.4 Picking

8 Conclusion

Pour conclure, ce projet nous a permis d'approfondire et de découvrir de nouvelle connaissance sur le développement d'application de bureau. Le sujet fut intéressant mais conscéquent, en effet avec seulement quatre périodes de cours par semaines nous avons trouvés les interstices rudes car d'une semaine à l'autre nous oublions facilement les connaissances acquérient lors du cours précédents.

8.1 Retour sur le développement

Nous avons pu découvrir qu'il était possible d'intégrer un objet *Monogame* dans une windows forme C#. Cette méthode n'est pas compliquée à mettre en place mais selon la solution la communication entre l'interface et le *monogame* est plus rude à mettre en place. Durant le développement, nous avons dû faire face à plusieurs problèmes tels que le *picking*. Cette solution, non terminée, permet à un utilisateur de de cliquer sur un objet 3D. Seulement des problèmes ont été rencontrés tels que le clique impossible.

8.2 La bibliothèque

Avec ce projet, nous avons découvert qu'il était possible de communiquer sur une trame usb grâce à des bibliothèques. Elles permettent de rendre le processus de communication plus facile en cachant au développeur de nombreuse données fastidieuses à manipuler.

9 Source

— Documentation de l'équipe Cube à led de 2015-2016 (existant)

Table des figures