Cubicle

# Firmware sur STM32 F4 pour cube à LEDs

Sujet de projet de spécialité 2A Ensimag

2014-2015

**Encadrant :** Charles Eynard

**Contact :** pierre.schefler@cubicle-3d.com

# Préambule

Le projet *Cubicle* vise à concevoir un afficheur 3D pour représenter des motifs cristallographiques (voir ci-dessous).


C'est un réseau cubique tridimensionnel de 9 * 9 * 9 = 729 LEDs allumables individuellement. Une carte SD à l'arrière contient des motifs qui sont chargés au démarrage du dispositif. L'écran à l'avant affiche le nom du motif représenté et les boutons permettent de le manipuler.

Pour faciliter la navigation, les motifs sont organisés par groupe. L'écran LCD fait 2 * 20 caractères. La ligne du haut affiche le nom du groupe et celle du bas le nom du motif affiché (qui fait donc partie du groupe). Les 6 boutons de gauche servent à déplacer les structures dans l'espace. Parmi les 4 de droite, les 2 du haut servent à naviguer parmi les groupes et les 2 du bas servent à naviguer parmi les motifs, à l'intérieur du groupe.

A l'heure actuelle, le cube utilise deux Arduino. Pour des raisons de fiabilité, de performances et de coût, il est prévu de passer sur un microcontrôleur intégré directement à la carte électronique, en l'occurrence, un stm32 F4.

# Objectif

Le but de ce projet est de programmer le nouveau firmware. Les fonctionnalités ci-dessous sont requises.

## Contrôle des LEDs

La première chose à faire est de savoir contrôler les LEDs convenablement. La carte électronique à l'intérieur du cube sera adaptée pour tester le firmware depuis le stm32. Vous trouverez une explication détaillée du fonctionnement de la carte en annexe 1 qui vous servira à écrire la première brique permettant de contrôler l'allumage des LEDs.

Cette étape vous amènera à contrôler les bus SPI du microcontrôleur et à réfléchir à une structure de données avantageuse pour décrire la matrice de LEDs. Cette structure doit être la plus petite possible sans pour autant demander trop de ressources lors du traitement. En ce sens, une bonne solution est d'allouer de la mémoire pour 81 blocs de 2 octets. Chaque bloc représente une ligne de 9 LEDs. On gaspille un peu de mémoire mais la ram du F4 est suffisamment grande pour que ce soit négligeable. On pourrait compresser les données mais le nombre d'opérations requises à chaque traitement de la structure ne serait pas forcément avantageux. Vous êtes toutefois libres de choisir ce que vous voulez, voire de faire des tests de performances…

## Manipulation des motifs

Une fois cette première couche implémentée, vous pourrez passer à un niveau d'abstraction supérieur en manipulant directement les motifs présents sur la carte SD. A l'heure actuelle, les motifs sur la carte SD sont organisés d'une façon particulière décrite en annexe 2.

Pour réaliser cela, vous aurez à créer des communications avec les interrupteurs, l'écran LCD et la carte SD. Utiliser un des 4 boutons de droite implique l'accès à la carte SD pour charger un autre motif (ce qui met à jour l'affichage sur l'écran LCD, cf annexe 3) et utiliser un des 6 boutons de gauche produit une translation sur le motif affiché. Cela nécessitera quelques fonctions haut niveau de manipulation géométrique de la structure de données.

En outre, le cube à l'heure actuelle contient deux Arduino car il est important que l'un puisse afficher en permanence pendant que l'autre réalise les calculs (translation, etc). De si simple calculs peuvent malgré tout (sur Arduino) demander un certain temps de calcul qui crée des flash noirs lors de l'affichage, à cause de l'attente. D'où la nécessité d'avoir les tâches de calcul et d'affichage séparées. Avec le F4, plus besoin d'avoir deux processeurs : un OS très basique suffit. Il faudra donc implémenter FreeRTOS (gratuit et bien supporté par ST) afin de créer les différentes tâches qui s'exécuteront en parallèle.

# Contraintes

## Organisation du code

Le firmware doit être écrit en C uniquement. Veillez à uniformiser vos règles de codage et à bien commenter.

Vous aurez à implémenter des drivers pour contrôler les interrupteurs, la carte SD, le SPI, et l'écran LCD. Vous pouvez utiliser les HAL (des drivers fournis par ST) ou les faire vous-mêmes. Notez que pour la carte SD, vous avez tout intérêt à utiliser la libraire FatFS. Quant au LCD, il utilise le driver standard Hitachi HD44780. Des librairies doivent donc pouvoir se trouver sans problème. En fait, vous avez sûrement tout intérêt à écrire vous-même les codes d'initialisation et d'utilisation des interrupteurs et du SPI et à savoir en détails ce que vous faites plutôt que d'inclure une partie des drivers écrits par ST.

Veillez à bien séparer les différentes couches logicielles dans le firmware. La couche applicative doit être indépendante des drivers afin de pouvoir adapter le firmware sans trop de travail si le microcontrôleur ou un autre composant venait à changer.

## Performances

Il faut atteindre la fréquence de rafraîchissement la plus haute possible. Mais cela ne doit pas se faire au détriment de la lisibilité du code. Il doit être simple et facilement paramétrable. http://en.wikipedia.org/wiki/Don%27t_repeat_yourself

# Bonus

Les fonctionnalités suivantes ne sont pas essentielles à votre travail mais pourraient être intéressantes pour vous et pour le firmware.

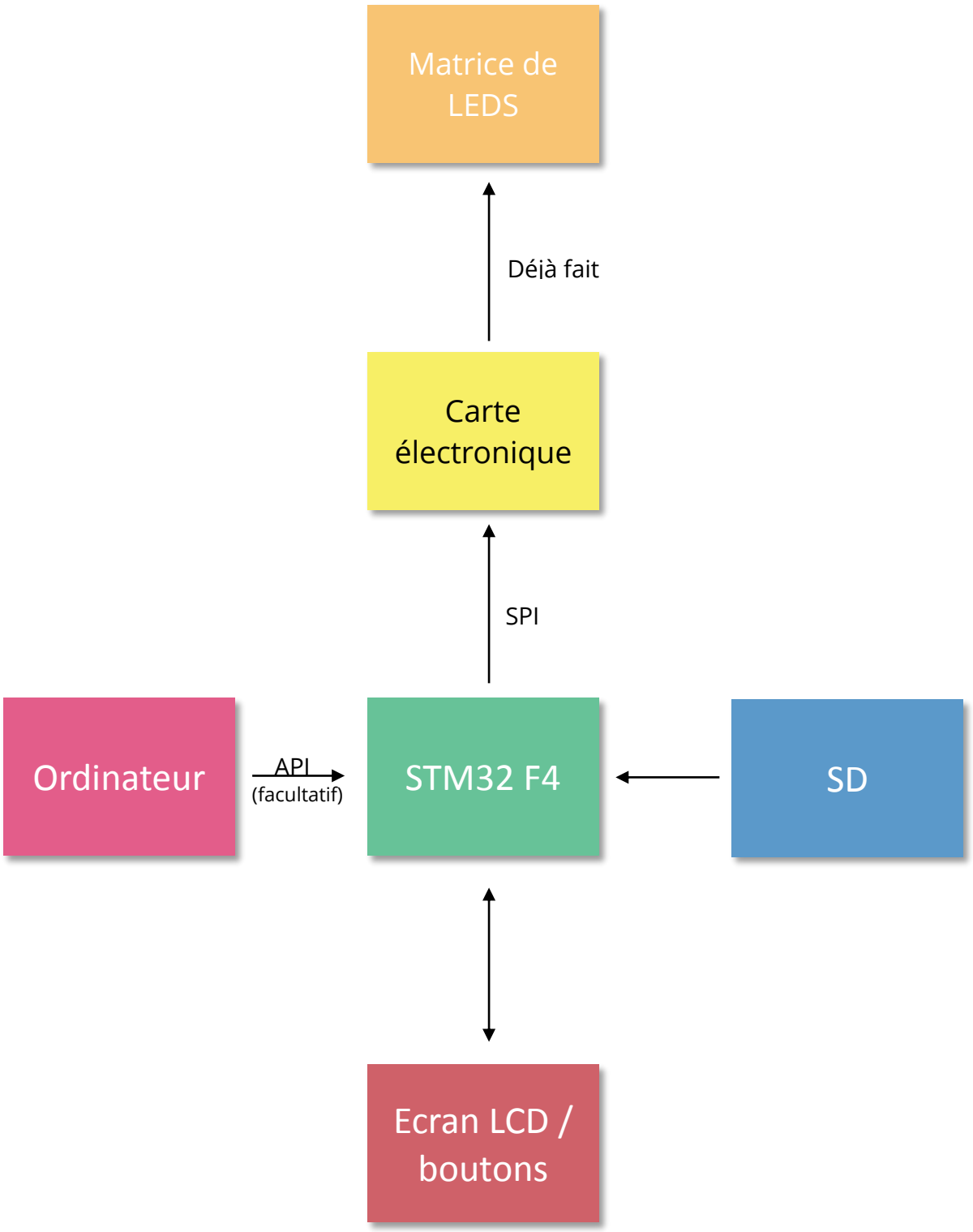
## Mise à jour

Un port USB est présent à l'arrière du cube et est relié à l'intérieur au stm32. Il est prévu pour pouvoir faire des mises à jour du firmware. Cela peut se faire par simple transfert de fichier binaire à travers une liaison USB CDC. Il faudra donc un driver USB CDC. En outre, ST fournit un bootloader à placer sur le microcontrôleur pour gérer cette mise à jour. ST fournit également un logiciel capable de communiquer depuis l'ordinateur avec le bootloader. Si vous voulez vous pencher sur cette fonctionnalité, vous avez donc tout intérêt à utiliser les éléments proposés par ST.

## Communication temps réel avec l'ordinateur

Quitte à avoir un port USB pour faire des mises à jour, il serait intéressant de s'en servir pour utiliser le cube en temps réel depuis l'ordinateur. Pour cela, il faudra implémenter un service USB avec une tâche chargée de collecter les données et une API pour les traiter. Une API est proposée en annexe 4. C'est un travail lourd mais sûrement très intéressant.

# Schéma récapitulatif des différents éléments



# Éléments fournis

## Cartes de développement

Des cartes STM32F4 DISCOVERY seront fournies pour que vous puissiez tester et déboguer votre code sur un microcontrôleur réel (et non un simulateur).

## Prototype

De plus, les cartes de développement seront fournies avec les connexions nécessaires vers un prototype pour pouvoir en utiliser toutes les fonctionnalités : matrice de LEDs, écran LCD, boutons poussoirs, carte SD et port USB.

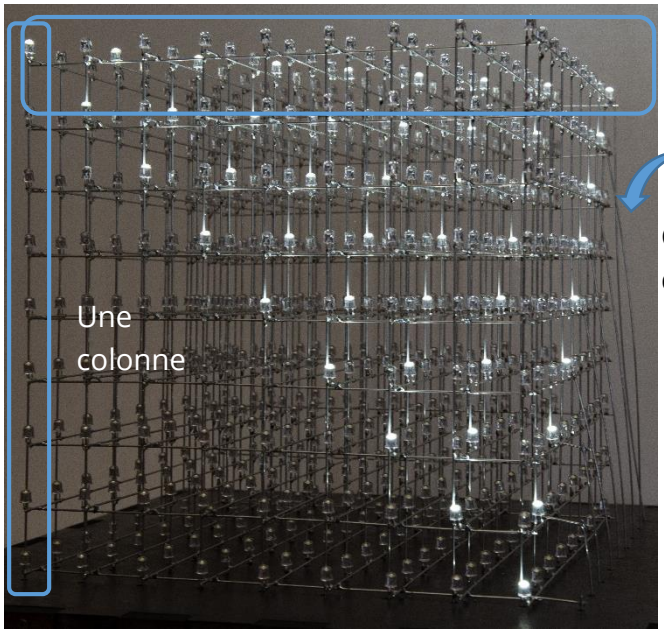
# C'est parti

Vous devrez faire preuve d'initiative et d'autonomie afin d'organiser votre travail et votre code le mieux possible. N'ayez pas peur de proposer des choses, même si elles devaient remettre en cause une partie du sujet. Ne voyez pas ce travail comme un rapport de force hiérarchique mais plutôt comme une véritable collaboration dans laquelle vous êtes invités à intervenir.

Soyez forts !

# Annexe 1 : contrôle des LEDs

La matrice de LEDs est formée de 9 * 9 * 9 LEDs. On peut envoyer du courant dans chaque plan horizontal (9 plans de 81 LEDs), et on peut bloquer ou autoriser le passage du courant dans chacune des 81 colonnes.




Quand on envoie du courant dans un plan et qu'on autorise son passage dans une colonne, une LED va s'allumer. A partir là, on peut donc choisir les points que l'on veut allumer sur un certain plan. Ensuite, on peut répéter cette logique pour chaque plan. Bien sûr, on ne peut pas traiter deux plans en même temps. Il faut donc balayer les plans un à un, et, pour chacun, choisir les colonnes à commander en fonction des LEDs à allumer.

Le passage du courant est contrôlé par des registres à décalage de ST, que nous nommerons sink drivers. Leur datasheet est disponible avec le sujet. Chaque sink driver a 16 sorties. Ils sont au nombre de 10. Un sink driver contrôle les plans et les 9 autres contrôlent les colonnes (chacun pour une rangée de 9 colonnes). Cela veut donc dire que sur chaque sink driver, 7 sorties ne sont pas utilisées. Malgré le gaspillage de composants, il est préférable de faire de la sorte pour des questions de clarté et surtout de routage.

Ces registres à décalage s'utilisent grossièrement de cette façon : on envoie un signal d'horloge (20 MHz maximum) avec un signal de données de 16 bits, puis un créneau du signal LE (latch enable) pour activer l'écriture des données. Un autre pin du sink driver, OE (output enable) permet d'éteindre entièrement le sink driver. Dans les données, le premier bit donne l'état de la sortie 0, le second de la sortie 1, etc., jusqu'au dernier bit qui donne l'état de la sortie 15. Pour allumer la première LED, il faudrait donc envoyer un 1 et quinze 0 dans le sink des plans et dans celui de la première rangée de colonnes.

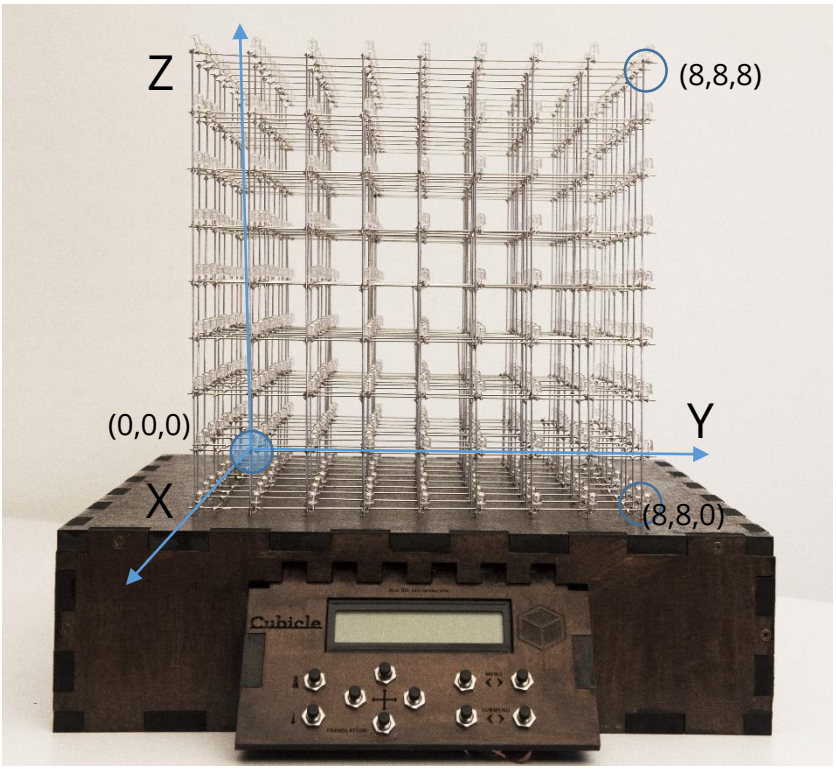
A l'heure actuelle, pour économiser le nombre de bus nécessaires, l'usage est simplifié. Tous les sinks sont chaînés entre eux, ce qui veut dire que les données qui arrivent dans le premier sink driver sont transmises au second, qui les transmet au troisième, et ainsi de suite jusqu'à arriver au dernier qui lui ne pourra plus les transmettre. Il faut donc remplir les 10 sink d'un seul coup puis enregistrer les données (avec LE). Pour simplifier encore, tous les OE sont actifs en permanence, on ne peut pas les commander. Cela peut en revanche avoir un impact sur les performances.

En effet, la vitesse du balayage est cruciale pour le bon fonctionnement de l'affichage. On pourrait imaginer avoir un contrôle individuel sur les sink pour ne toucher qu'à ceux qui

ont des informations importantes à recevoir. L'ordre d'affichage des LEDs pourrait même être optimisé astucieusement avec des algorithmes d'ordonnancement si certains d'entre vous sont à l'aise en théorie des graphes. Mais tout cela demanderait certainement une quantité de calculs non négligeable et qui, finalement, pourrait en fait réduire les performances au lieu de les accroître.

Concrètement, pour contrôler les sink drivers, le mieux est d'utiliser un bus SPI ou un autre bus qui a une horloge matérielle. Trois pins sont nécessaires : l'horloge, les données et LE. Envoyez les bits un à un puis faites un court créneau de LE (attention, asynchrone). Un schéma est disponible dans la data sheet des sink drivers.

Voici maintenant quelques précisions sur le repère utilisé par le cube à LEDs. L'origine est au fond, en bas, à gauche (ce qui veut dire que toutes les coordonnées sont positives ou nulles). Les coordonnées sont toujours entières. L'axe x va de l'arrière vers l'avant, l'axe y de la gauche vers la droite et l'axe z du bas vers le haut (cf schéma ci-dessous).



# Annexe 2 : carte SD

La carte SD contient des motifs organisés en groupe. A chaque action sur un bouton pour changer de groupe ou de motif, il faut venir charger un motif à partir de la carte SD. On pourrait tous les charger au démarrage mais ce serait très risqué pour l'espace de mémoire vive, c'est pourquoi il est préférable de les charger au cas par cas.

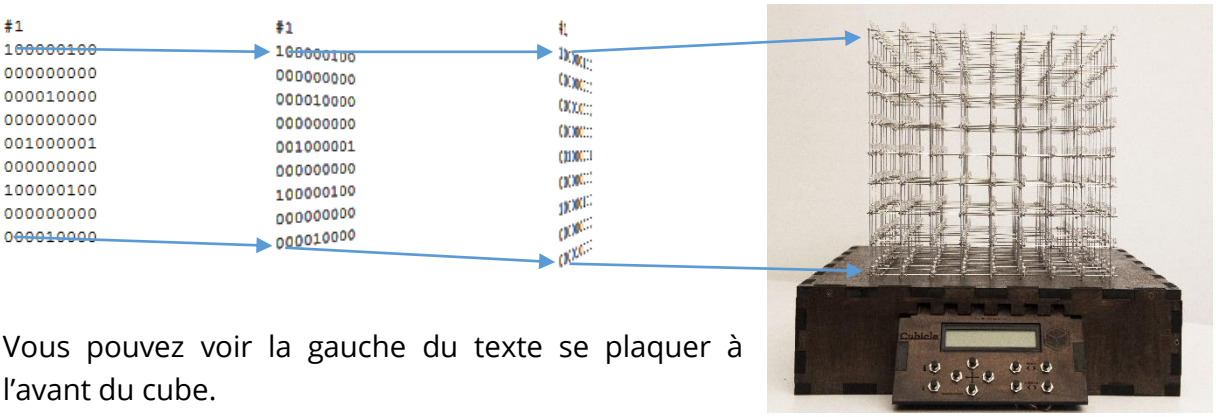
Un motif est représenté par un fichier .txt. Les motifs d'un même groupe sont stockés dans un même dossier dont le nom est le nom du groupe. A la racine, un fichier ORDRE décrit l'ordre des groupes. L'ordre des motifs au sein d'un groupe est défini par l'ordre alphabétique des noms des fichiers .txt. La raison à cela est qu'il fallait une méthode simple pour créer et organiser les fichiers à la main. Bientôt, un logiciel permettra de faire tout ça et cette partie sera certainement modifiée, mais pour l'instant, nous resterons sur le même principe.

Vous trouverez un exemple d'arborescence joint au projet.

Passons au contenu des fichiers .txt. Chaque fichier commence par le caractère @ suivi du nom du motif. Par exemple : « @1 cell » veut dire que le motif s'appelle « 1 cell ». Ensuite, les plans sont décrits un à un.

Un plan commence par la mention « #x » où x est le numéro du plan. On peut commencer par le plan 0 ou le plan 1 (et finir respectivement par le plan 8 ou 9). **Attention**, ici, un plan est un plan vertical et non pas horizontal. Le plan vertical est décrit par une coordonnée Y constante sur tous les points. En-dessous de la mention #x, on trouve 9 lignes de neuf 0 ou 1 qui décrivent chaque LED sur le plan. Parcourir une ligne de gauche à droite revient à parcourir le cube de l'avant vers l'arrière, passer à la ligne du dessous revient à passer au plan horizontal du dessous.

On peut le visualiser comme ça :




Vous pouvez voir la gauche du texte se plaquer à l'avant du cube.

Des fonctions pour parser les motifs sont fournies. Mais attention, elles sont en C++...

# Annexe 3 : spécifications du panneau de contrôle

Le panneau contient un écran LCD et 10 boutons.

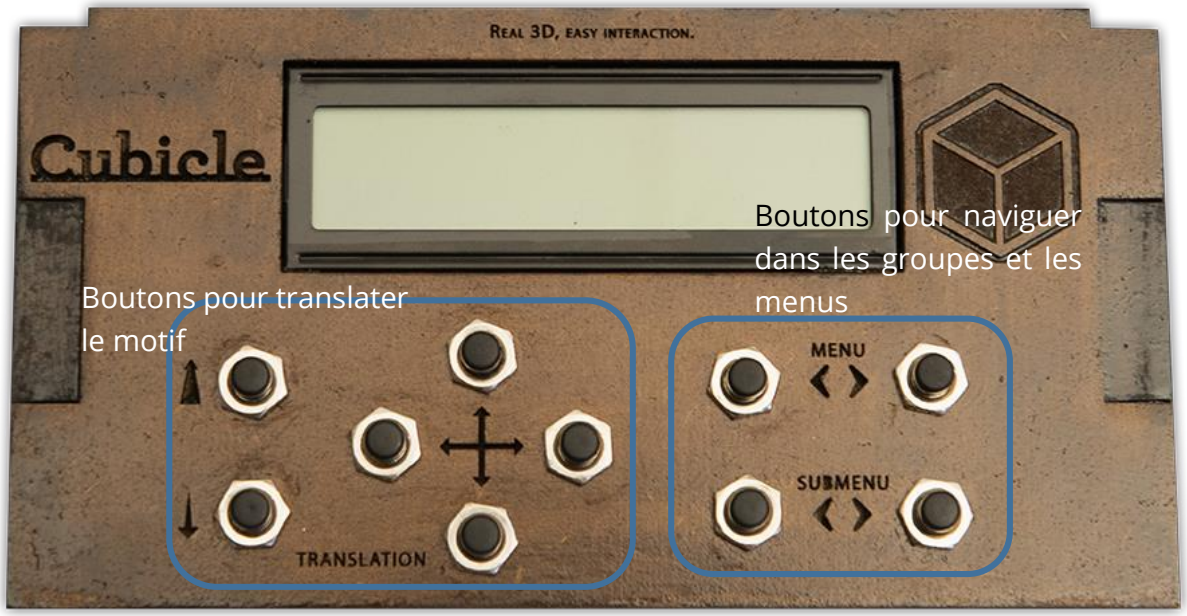
L'écran LCD fait 2 * 20 caractères. Vous trouverez sa datasheet dans les fichiers du projet.

La ligne du haut est utilisée pour afficher le nom du groupe et celle du bas pour afficher le nom du motif.

Pour chaque ligne, 5 caractères à gauche sont réservés pour afficher le numéro du groupe ou motif actuel et les 15 caractères restant pour afficher le nom, comme on peut le voir ci-dessous.


Cela signifie que le nom d'un motif ou d'un groupe ne peut pas dépasser 15 caractères.

Voici maintenant une explication des boutons.




- On peut translater dans les 3 directions (donc 6 sens) de l'espace.
- Passer d'un groupe à l'autre affiche le premier motif du nouveau groupe.
- La navigation est circulaire : passer au groupe ou motif suivant si on est au dernier affichera le premier de la liste, et de même, si on demande le motif ou groupe précédent alors qu'on est au premier, on obtiendra le dernier.

# Annexe 4 : proposition d'API

*NB : cela ne vous sera utile que si vous décidez d'entamer le second bonus.*

Pour qu'un ordinateur et le cube puissent interagir, on peut envisager ouvrir une liaison série par une connexion USB (CDC). Les deux périphériques s'échangeraient des trames d'octets qui suivraient les spécifications suivantes.

Toute trame commence par B4 A6 E2, contient un quatrième octet de description, des données et finit par un hash (deux octets de CRC par exemple). Les données ne peuvent excéder 20 octets, donc la trame ne peut excéder 26 octets.

L'octet n°4 décrit l'opération. Les opérations de 0x10 à 0x70 concernent le transfert de l'ordinateur vers le cube, tandis que les opérations de 0x80 à 0xF0 concernent le transfert du cube vers l'ordinateur vers le cube. Les valeurs 00 et FF sont réservées pour des usages spéciaux.

A chaque réception de trame, le firmware vérifie le hash. S'il n'est pas concordant, il renvoie un code d'erreur : B4 A6 E2 FF (+ hash). Si tout est bon, il ne renvoie rien.

Opérations (ordinateur -> cube) :

- 0x10 : transférer une partie d'image (si dans la structure de données, deux octets décrivent une ligne de LEDs alors 9*2 = 18 octets décrivent un plan entier ; on peut donc insérer un octet qui correspond au numéro du plan puis 18 octets de données (NB : on pourrait également compresser les données pour accélérer le transfert))
- 0x20 : mettre à jour la première ligne de l'écran (+ chaîne de caractères de 20 octets)
- 0x21 : mettre à jour la seconde ligne de l'écran (+ chaîne de caractères de 20 octets)

Opérations (cube -> ordinateur) :

- 0x80 : un bouton a été enfoncé (suivi du numéro de bouton)
- 0x81 : un bouton a été relâché (suivi du numéro de bouton)