# Le projet GET-IT ou la réalisation

*Le tactile est le sens à la mode dans les soirées digitales. Que cela soit à travers les téléphones mobiles, certains laptops/netbooks, ou encore le dernier gadget au goût pomme, l'avènement du touch & play est en marche. Nous allons voir comment mettre en œuvre des technologies ouvertes pour arriver à faire aussi bien, sinon mieux, que la désormais fameuse surface fenêtrée.*

*Auteurs*

- Vincent Guyot,
  Nicolas Languy,
  Charles Somxay,
  Lionel Tougne

## 1 INTRODUCTION

GET-IT (*Gecko Easy Touch Interactive Table*) est un projet à l'initiative d'étudiants-ingénieurs. Il consiste en la réalisation d'une table tactile de grande dimension (environ 1m²), performante (affichage accéléré) et à faible coût (moins de 2000 euros). On peut voir sur la photo 1 la table achevée en fonctionnement.

L'idée de ce projet est survenue à l'issue d'une démonstration d'un produit similaire, en se demandant si on ne pouvait pas arriver à un meilleur résultat à l'aide des préceptes du FLTM, « fais-le toi-même » (googler DIY pour plus de résultats). Si la réalisation d'un périphérique numérique tactile n'est pas quelque chose de nouveau **[1]**, il est relativement récent de pouvoir désormais le réaliser soi-même.

Le gecko a été choisi comme emblème car c'est un animal capable de se déplacer sur tous les types de surfaces. Il évoque également un mangeur d'insectes nuisibles, les bugs.

*Photo 1 : La table achevée en fonctionnement*

## 2 UN PEU DE THÉORIE

### 2.1 Les différents types de technologies tactiles

À ce jour, il existe plusieurs méthodes pour parvenir à cet objectif de table tactile. Trois grandes technologies existent pour détecter des points lorsque l'on touche une surface tactile :

- Les techniques se basant sur des systèmes électroniques (écran résistif ou capacitif). Ce sont les systèmes les plus répandus, ils sont utilisés, par exemple, dans les téléphones mobiles évolués ainsi que dans les nouveaux écrans d'ordinateurs tactiles.
- Les techniques se basant sur le son. Ces méthodes sont apparues depuis peu et consistent à mesurer la propagation des ondes sonores lorsqu'une personne touche la surface tactile. À l'aide de plusieurs capteurs, il devient en effet possible d'effectuer une triangulation des signaux et ainsi de localiser précisément l'endroit où se situe le doigt sur la surface considérée.

# d'une table tactile

- Les techniques se basant sur des systèmes optiques (FTIR, DI, etc.). Ces méthodes reposent essentiellement sur le traitement d'images. Pour réussir à détecter les points de contact sur la surface tactile, on utilise des signaux lumineux dans l'infrarouge ou encore de type laser.

En ce moment, tous les ordinateurs tactiles (y compris les téléphones et autres terminaux) se basent sur des systèmes électroniques. Il y a des avantages et des inconvénients à cette utilisation. Les systèmes électroniques sont aujourd'hui adaptés aux produits de taille modeste. Néanmoins, leur coût de fabrication élevé ne permet pas, pour l'instant, d'adapter cette technologie aux très grands écrans. À des fins de réduction de coûts, les produits de surfaces plus importantes doivent donc utiliser un écran plat ou encore un système optique dont la résolution de l'image va dépendre du système de projection utilisé. La précision et la fluidité seront fonction de la qualité des caméras infrarouges, en fonction de leur résolution et de leur vitesse de transfert d'information, ainsi que de la puissance de l'ordinateur qui traitera les informations. Aujourd'hui, nous constatons que les systèmes optiques commencent tout juste à émerger.

## 2.2 Les technologies basées sur l'optique

Deux technologies sont intéressantes pour ce projet, car elles sont relativement simples à mettre en œuvre. Toutes deux consistent à réaliser une table dans laquelle on place des diodes infrarouges sur le côté d'une plaque en plexiglas afin de pouvoir détecter le toucher d'un doigt sur la surface tactile à l'aide d'une caméra capable de détecter le rayonnement infrarouge.

### 2.2.1 La méthode DSI (Diffused Surface Illumination)

Le type de plexiglas utilisé ici est de type « *EndLighten Acrylic* ». La particularité de ce type de plastique réside dans le fait qu'il contient des particules réparties de façon homogène. Par un jeu de réflexion et diffraction, une partie des rayons s'échappent du milieu de la plaque en plexiglas alors que la majorité des rayons y restent emprisonnés. En posant les doigts sur la plaque de plexiglas, comme illustré sur la figure 1, on modifie l'incidence des rayons provenant des diodes infrarouges. Cela permet de détecter la perturbation locale par une caméra infrarouge placée sous la plaque de plexiglas. Ce type de technologie permet également de détecter un objet posé sur la surface de la table comme un code-barres, à la manière de ce qu'est capable de faire la table tactile Surface si on lui adjoint un programme spécifique et une caméra haute résolution.

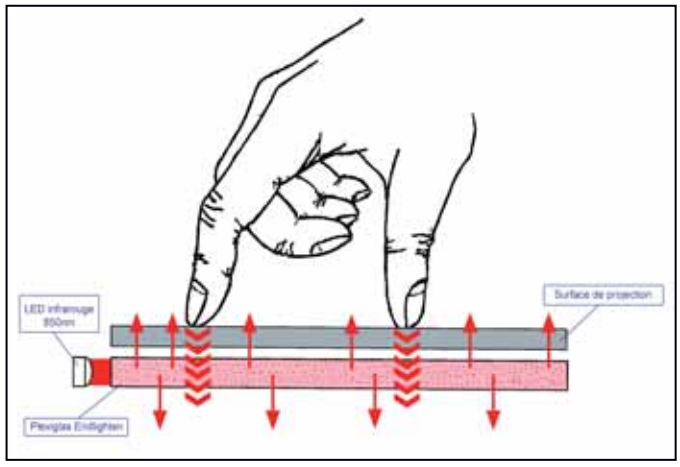


*Fig. 1 : Illustration de la technologie DSI*

### 2.2.2 La méthode FTIR (Frustrated Total Internal Reflection)

Pour mettre en œuvre cette technologie, nous devons nous doter d'une plaque de plexiglas classique sur laquelle on peut appliquer une couche de silicone. Les propriétés optiques du plexiglas et de l'onde infrarouge permettent de confiner cette dernière dans la plaque lorsqu'un doigt touche la plaque tactile, comme illustré sur la figure 2.

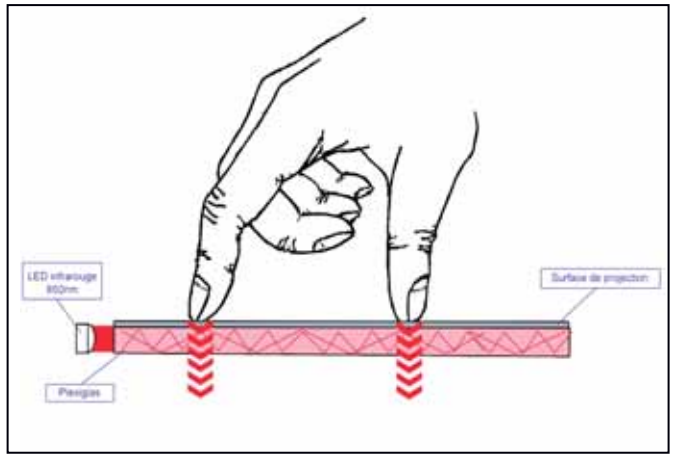


*Fig. 2 : Illustration de la technologie FTIR*

### 2.2.3 Les avantages et les inconvénients

En se basant sur les forums de Nuigroup **[2]**, une communauté open source travaillant sur les technologies tactiles, on peut résumer les avantages et inconvénients de ces deux technologies dans le tableau qui suit.

| | Avantages | Inconvénients |
|---|---|---|
| FTIR | ■ contraste élevé<br>■ utilisable avec un écran LCD | ■ inadaptée à la reconnaissance d'objets<br>■ inadaptée à une surface en verre |
| DSI | ■ permet la détection d'objets<br>■ facilité de réalisation | ■ coût élevé du « Enlighten Acrylic »<br>■ faible contraste |

### 2.2.4 La technologie retenue

Après avoir pesé le pour et le contre de chacune des deux méthodes vues précédemment, la méthode DSI a été choisie. En effet, elle permet de faire plus de choses que la méthode FTIR, comme la possibilité de travailler sur la détection des codes-barres 2D ou encore de pouvoir lire ce qui est écrit sur une feuille posée face contre la table. Notons toutefois que la méthode DSI est plus onéreuse à mettre en œuvre que la méthode FTIR.

La méthode DSI retenue met en œuvre différents composants :

- Un vidéo-projecteur pour afficher l'image.
- Une caméra infrarouge pour détecter le contact des doigts.
- Un ordinateur doté d'accélération graphique pour traiter les informations, mais également pour afficher les applications utilisées.

*Photo 2 : Une bande de diodes infrarouges en fonctionnement sur un côté de la table*

- Une première plaque en plexiglas de type « EndLighten Acrylic ». En plaçant des diodes infrarouges sur les côtés de cette plaque, nous éclairons toute sa surface. Puis, par un jeu de réflexion et diffraction, une partie des rayons s'échappe du milieu. Cela a pour effet de faire briller de façon homogène la plaque, à la manière d'une lampe. Sur la photo 2, on voit ces diodes en fonctionnement.
- Une seconde plaque, une surface de projection légèrement opaque qui permet de bloquer les images issues du vidéo projecteur, mais également d'interagir avec le système.

# 3 LES SPÉCIFICITÉS TECHNIQUES DE LA TABLE ET LE CHOIX DES COMPOSANTS

## 3.1 Le système de pointage

Le système de pointage est réalisé à l'aide de plusieurs éléments :

- une plaque EndLighten ;
- une plaque de support de projection ;
- des diodes infrarouges ;
- une caméra infrarouge.

Le principe de fonctionnement est simple, il repose sur des propriétés géométriques d'optique que nous allons expliciter.

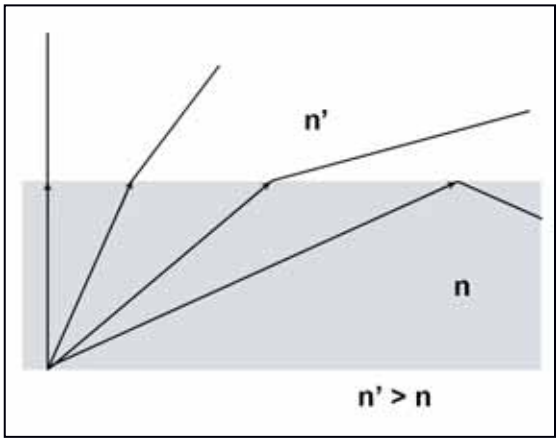


*Fig. 3 : Réfraction limite*

Pour mieux comprendre ce qui s'y passe, nous devons connaître les lois de Snell-Descartes :

- L'indice de réfraction du milieu n est le rapport entre la vitesse de la lumière dans le vide c et dans le milieu v.

  On a donc la relation : $n = c / v$.

- Soit l'angle incident O1 et l'angle réfracté O2.

  Ils sont tous deux liés par la relation : $n1.\sin(O1) = n2.\sin(O2)$.

### 3.1.1 Les diodes infrarouges et la plaque Endlighten

Des diodes infrarouges d'une longueur d'onde de 850nm sont placées tout le long de la plaque de plexiglas EndLighten.

L'indice du milieu du plexiglas est plus élevé que celui de l'air. En effet, la matière qui compose le plexiglas EndLighten est le poly-méthacrylate de méthyle, un thermoplastique qui ralentit la lumière du fait de sa haute densité.

Il en résulte que si la source de lumière incidente possède un angle trop petit, les rayons seront totalement

réfléchis, comme le montre la figure 3. Or, on remarque que les propriétés physiques sont différentes. L'angle des rayons en provenance des diodes est relativement faible. De ce fait, on pourrait s'attendre à ce que l'infrarouge reste emprisonné dans la plaque. Mais il s'avère que les rayons infrarouges s'en échappent comme si la plaque était elle-même une lampe.

Cette propriété est très intéressante, c'est en effet ce qui est recherché pour appliquer la technique DSI (*Diffused Surface Illumination*). Afin d'expliquer pourquoi les rayons lumineux sont renvoyés vers l'extérieur de la plaque de façon uniforme, nous pensons qu'il y a au niveau structurel d'autres particules. Ces dernières doivent posséder un indice de réfraction différent de celui du poly-méthacrylate de méthyle avec un effet de miroir semi-réfléchissant, permettant ainsi au niveau microscopique d'avoir un jeu de diffraction et réfraction. Les informations du constructeur étant confidentielles, il n'a pas été possible de vérifier cette théorie. Toutefois, il est mentionné dans les spécificités de la plaque la présence de particules qui caractérisent ce produit et permettent une diffusion de lumière homogène.

Il est à noter qu'avec cette méthode, beaucoup de rayons infrarouges sont perdus. Il est par conséquent nécessaire d'obtenir une grande quantité de lumière en utilisant des diodes infrarouges de grande puissance, ou encore en en plaçant une grande quantité tout autour de la plaque.

*Photo 3 : Le caméra PlayStation Eye et ses lentilles*

### 3.1.2 Le support de projection 7D006

Les propriétés de la plaque EndLighten permettent donc de rediriger les rayons infrarouges vers l'extérieur de la plaque, en particulier vers le support de projection. Ce support est la surface d'interaction homme-machine dite « tactile ». On peut voir qu'il y a un avantage à utiliser une surface de projection pour faire l'interaction. En effet, les rayons incidents en provenance de la plaque EndLighten sont très faibles, il en résulte que l'infrarouge éclaire tout ce qui se trouve à la surface de la table, aussi bien les doigts que d'autres objets, comme une feuille de papier, par exemple.

L'utilisation d'une plaque de plexiglas de type 7D006 n'est pas la seule solution pour réaliser l'affichage. Le site web **[3]** liste quelques matériaux que l'on peut utiliser. Dans un souci d'avoir un matériel résistant et une image de qualité, notre choix s'est porté vers le modèle de type 7D006, un peu plus onéreux, utilisé notamment dans la table Surface.

De plus, la granularité de sa surface permet d'éliminer les images du fond, facilitant ainsi le traitement numérique.

### 3.1.3 La caméra « PlayStation Eye » modifiée

Pour la caméra infrarouge, nous nous sommes orientés vers l'utilisation de la caméra commercialisée pour la console de jeu Sony PlayStation 3, référencée « PlayStation Eye » (voir **[4]**), qui a été modifiée pour l'occasion. Cette caméra a été retenue en raison de son débit élevé (de 60fps à 125fps) et de sa relativement haute résolution (640x480), le tout pour un coût assez raisonnable. Les modifications de ce produit concernent l'angle de vision et le filtre qui détecte les rayons infrarouges de longueur d'onde 850nm. Nous avons donc testé plusieurs types de focales pour utiliser celle qui nous conviendrait le mieux. On constate que pour obtenir un recul suffisant afin de couvrir toute la table, un miroir peut être utilisé.

Chez **[3]**, nous avons fait l'acquisition de différentes lentilles de type M12 (voir photo 3) qui nous permettent de pouvoir tester et d'utiliser celle qui nous convient le mieux. Ce lot possède les lentilles suivantes (focale - angle d'ouverture) :
1,8 mm - 100 degrés ;
3,6 mm - 73 degrés ;
6 mm - 46 degrés ;
8 mm - 29 degrés ;
12 mm - 19 degrés ;
16 mm - 15 degrés.

Ce lot de lentilles est très pratique dans la mesure où il permet de choisir en fonction de la surface couverte par la caméra. Notre caméra PS3 peut fonctionner à 60fps avec une résolution de 640x480. Or la taille de notre table est de 120cm x 75cm. Nous pouvons donc calculer approximativement la taille d'un pixel, soit 1,88mm x 1,57mm, qui est donc notre seuil minimal de détection. Nous pouvons constater que la taille de nos doigts, lorsque nous interagissons avec la table, est nettement plus importante que le seuil limite de notre caméra. De plus, la vitesse à laquelle les images sont envoyées vers l'ordinateur est suffisamment élevée pour qu'il y ait une fluidité dans les mouvements. C'est pour ces deux raisons que nous avons validé l'utilisation de ces matériels. Néanmoins, cette résolution n'est pas suffisante pour permettre la reconnaissance d'un marqueur placé sur la surface de la table, les détails de celui-ci étant trop fins pour la caméra.

Même si désormais, le noyau Linux (2.6.31+) intègre un *driver* prenant en charge cette caméra, il est nécessaire de le patcher pour obtenir de bien meilleurs résultats. La marche à suivre détaillée est donnée en **[5]**.

## 3.2 Le système de visualisation

Le système de projection permet de visualiser ce que nous voulons afficher. Il est constitué de différents éléments :

- une plaque de support de projection ;
- un vidéo-projecteur grand angle ;
- des miroirs.

C'est en fonction des spécificités classiques des vidéo-projecteurs que nous avons défini la taille de la table. Mais connaître ces spécificités n'est pas suffisant. Cela s'explique simplement par le fait qu'il faut connaître l'angle de projection ainsi que les différentes corrections possibles avant de concevoir la forme de la table. Il est donc préférable d'acheter le vidéo-projecteur en début de projet afin de pouvoir effectuer des tests préalables.

La photo 4 montre ces différents éléments en situation.

### 3.2.1 Le vidéo-projecteur

Ces différents éléments permettent d'obtenir une distance suffisante pour que l'image couvre toute la surface utile de la table. En effet, notre table est grande et il est difficile d'obtenir un recul important pour un vidéo-projecteur classique (non grand-angle). La condition sine qua non est donc de mettre en place un jeu de miroirs. Notre choix s'est donc porté naturellement vers un vidéo-projecteur grand angle.

Pour connaître la taille d'affichage de l'image, nous nous sommes concentrés sur le rapport de projection. C'est le rapport entre la distance du vidéo-projecteur à l'image par la largeur de l'image. Plus il est faible, plus courte sera la distance de projection à l'image :

- rapport de projection = (distance du vidéo-projecteur) / (largeur de l'image)

  Ainsi, avec la taille de l'image (hauteur, largeur) et la distance de projection à l'image, nous avons pu évaluer le positionnement du vidéo-projecteur dans notre table. En effet, nous avons voulu construire une table au format 16:10.

*Photo 4 : Les coulisses...*

- format de l'écran = largeur / hauteur

  Pour connaître le positionnement du vidéo-projecteur, certains sites web proposent de faire les calculs en fonction du type de vidéo-projecteur choisi et de ses caractéristiques optiques intrinsèques. Par exemple, nous pouvons consulter le site web **[6]** qui nous fait ces calculs. Pour le projecteur NEC NP500ws que nous avons choisi, nous obtenons une image de 140cm de diagonale pour une distance de 88cm entre la focale et la surface de projection.

Pour les personnes désirant utiliser un projecteur classique, il faudra se résoudre à construire une table plus petite. Une autre solution serait d'utiliser un jeu de miroirs pour obtenir plus de recul ou d'utiliser un vidéo-projecteur déporté par rapport à la table.

Notons toutefois qu'avoir une image petite limite le nombre d'utilisateurs simultanés de la table.

### 3.2.2 Le miroir

Bien que nous ayons utilisé ce système avec un vidéo-projecteur grand angle, du fait de la distance de projection relativement faible, nous avions dû utiliser un miroir pour maximiser la surface de projection (voir photo 4).

## 3.3 Unité de traitement d'information

Dans un souci de rapidité, il a semblé intéressant de ne pas concevoir les pièces électroniques, mais plutôt de les acheter toute faites. Ainsi, il n'y a eu qu'à assembler les différentes pièces de la table à réaliser.

Quant à l'ordinateur, à titre indicatif, nous avons fait fonctionner notre système sur trois ordinateurs portables de puissances différentes :

- Un Apple ayant un processeur Core 2 Duo T7200 à 2GHz et un *chipset* graphique Intel GMA950 intégré à l'accélération graphique limitée.
- Un Sony VAIO SR31M qui possède un processeur Core 2 Duo P8600 à 2,4GHz et une carte graphique accélérée ATI Radeon HD3470.
- Un Sony VAIO Core i5 520 à 2,5GHz muni d'une carte graphique accélérée nVidia330.

On peut bien sûr utiliser d'autres configurations, pour peu qu'elles aient une puissance de calcul suffisante et un affichage graphique accéléré. Par conséquent, pour pouvoir interagir avec la table, il faut une chaîne complète de traitement allant du dispositif de pointage jusqu'au logiciel *middleware* contrôlant la table. Nous détaillerons dans un premier temps tout ce qui permet au système de détecter le toucher sur la surface de la table, puis dans un second temps, le système qui permet d'utiliser ce toucher dans les applications.

### 3.3.1 L'acquisition des données : CCV (Community Core Vision )

À chaque image renvoyée par la caméra, nous voulons pouvoir détecter les objets à la surface de la table. Pour ce faire, les informations doivent avoir une continuité dans la série d'images consécutives.

On peut simplifier encore plus le problème car un humain utilise le système. Nous savons que son doigt se déplace sur une distance relativement faible entre deux images consécutives. Pour résoudre ce problème, on utilise la méthode des k plus proches voisins (k *Nearest Neighbors*). Cette méthode compare la distance euclidienne entre un point d'une image et un ensemble d'autres points de l'image suivante pour en donner une classification. Ainsi, les deux points les plus proches d'une image à la suivante seront associés à un même objet.

L'approche par infrarouge supprime une partie des informations (la lumière dans le domaine du visible). Par conséquent, pour éviter les perturbations, le programme supprime le fond comme évoqué plus haut, et par seuillage retient les informations les plus pertinentes que nous cherchons, c'est-à-dire les formes. Dans notre cas, nous pouvons alors observer les doigts et les objets posés sur la table.

Le logiciel de détection que nous avons choisi d'utiliser, CCV (*Community Core Vision*) **[7]**, implémente ces algorithmes de détection. Ce logiciel a l'avantage d'être multiplate-forme et d'être l'un des plus évolués, mais il en existe d'autres tels que reacTIVision **[8]** et Touchlib **[9]**.

### 3.3.2 Le protocole TUIO

TUIO (*Tangible User Interface Objects*) est une bibliothèque de programmation open source **[10]** qui définit un protocole et une API (*Application Public Interface*). Ce protocole permet de transmettre toute information utile pour un système interactif, tel que le toucher ou la présence d'objet. Il a été initialement développé pour le projet « reacTable » (un système tactile pour musicien, voir **[11]**). Cette norme a ensuite été implémentée par plusieurs autres projets open source travaillant dans le domaine des interfaces interactives. Dû à son adoption très large par la communauté œuvrant dans ce domaine, le protocole TUIO est rapidement devenu un standard de fait.

La bibliothèque Python PyMT **[12]** que nous utilisons pour développer est capable de lire et traiter ce protocole (v1.1). TUIO est lui-même basé sur le protocole OSC (*Open Sound Control*). Cette norme permet à un ordinateur et un périphérique de communiquer à travers un réseau IP.

Techniquement, TUIO encapsule les données binaires conventionnelles de l'OSC dans un paquet UDP. Ces données sont ensuite envoyées sur le réseau IP à travers le port UDP 3333 (port par défaut). Pour détailler un peu plus le protocole TUIO, on peut dire qu'il définit deux types de messages :

- Les messages **SET**, qui regroupent les informations sur l'état d'un objet et ses caractéristiques (taille, accélération, etc.).

- Les messages **ALIVE**, qui indiquent les objets présents sur la table grâce à une liste, avec un identifiant unique pour chaque objet.

Il est à noter que TUIO n'envoie pas de message pour signifier l'ajout ou la suppression d'objets. La stratégie mise en œuvre consiste à laisser le client déterminer la durée de vie de ces derniers à partir des messages qu'il reçoit. Ceci a pour avantage de pallier les problèmes dus aux pertes de paquets inhérentes à UDP. De plus, TUIO envoie un message **SET** à chaque changement d'état d'un objet, et un message **ALIVE** à chaque changement de la liste d'objets. Chaque envoi possède un identifiant unique, défini dans un message de type FSEQ joint au paquet. Optionnellement, on peut ajouter un message de type **SOURCE** pour identifier l'expéditeur (l'application et l'adresse).

L'implémentation étant d'ores et déjà définie, nous allons regarder comment TUIO envoie son message. Chaque paquet UDP envoyé est construit de la manière suivante :

- au début du paquet un message **SOURCE** ;
- puis un message **SET** suivi d'un message **ALIVE** ;
- enfin, un message **FSEQ** vient terminer le paquet.

Le protocole définit des profils pour différents types d'interactions (2D, 2.5D, 3D). Chacun de ces profils est décomposé en trois sous-catégories :

- un objet (comme un marqueur) ;
- un curseur (comme celui de la souris) ;
- une forme détectée (comme la forme telle qu'elle est détectée par CCV).

Dans notre cas, nous ne nous intéressons qu'à l'interaction 2D, car c'est celle que nous utilisons et celle que PyMT lit. De plus, il est à noter que la bibliothèque PyMT n'utilise pas la catégorie « forme ».

Ci-dessous, voici les différentes variables lues par cette bibliothèque :

| Identifiant | Définition | Type |
|---|---|---|
| i | Class ID ( = marker ID) | **Int32** |
| x,y | Position | **float32** compris entre 0 et 1 |
| a | Angle | **float32** compris entre 0 et 2Pi |
| w, h | Dimension | **float32** compris entre 0 et 1 |
| X,Y | Vecteur Vitesse (Vitesse, Direction) | **float32** |
| A | Vecteur Rotation (Vitesse de Rotation, Direction) | **float32** |
| m | accélération rectiligne | **float32** |
| r | accélération angulaire | **float32** |

Plus précisément, voici comment la bibliothèque décompose ces variables en fonction des différents profils :

- Curseur 2D : xy
  - **xyXYm**
  - **xyXYmh**
- Objet 2D : xy
  - **ixyaXYAmr**
  - **ixyaXYAmrh**

Le protocole TUIO évolue, il devrait ouvrir la voie vers de plus grandes possibilités quant aux utilisations d'interfaces tactiles.

## 4 LE MONTAGE

Pour ce projet, nous avons voulu construire une table de grande dimension (55'' de diagonale, soit environ 1m40) afin de permettre à plusieurs utilisateurs d'interagir simultanément avec le système. Cette table étant un prototype, nous avons décidé de la construire en bois, qui est moins coûteux que d'autres matériaux comme l'aluminium et plus facile à monter que du plastique, par exemple.

Afin d'utiliser au mieux les capacités d'affichage, nous avons conçu une table au format 16:10 (voir figure 4), qui correspond au rapport d'affichage du vidéo-projecteur retenu. Les dimensions de la plaque de projection sur la table sont de 1280mm x 810mm et la plaque EndLighten fait 1277mm x 802mm. Remarque : il est nécessaire de garder une bordure de 4mm entre la plaque EndLighten et les montants de la table pour pouvoir intercaler les diodes infrarouges et le U en aluminium.

Avant d'acheter les plaques, nous conseillons d'abord de construire toute la structure pour les accueillir. En effet,

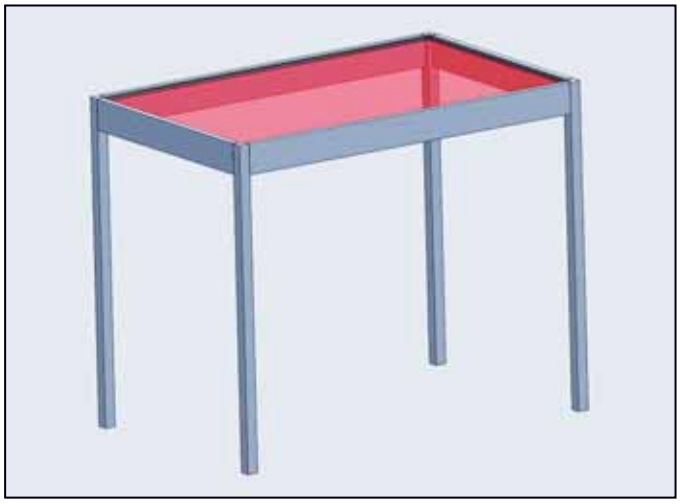

*Fig. 4 : La table au format 16:10*

l'usinage des pièces en bois n'étant pas précis au millimètre, des réajustements de taille pourront être pris lors de la commande des plaques.

La taille de l'image projetée, relativement grande, nécessite l'utilisation d'un vidéo-projecteur grand angle pour minimiser le recul de projection. Malgré cette utilisation, notre table requiert une hauteur d'au moins 105cm.

En figure 5, nous pouvons voir l'extrémité supérieure d'un des pieds de table dont nous avons extrudé le coin au ciseau à bois. Ce détail, bien que mineur, permet de pouvoir faire reposer une partie de la masse des plaques sur les pieds, tout en conservant un certain niveau d'esthétisme.

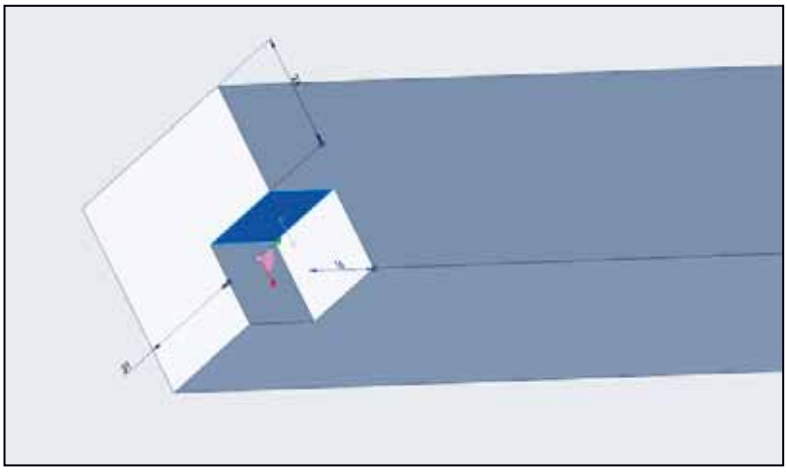

*Fig. 5 : Détail du haut d'un pied de la table*

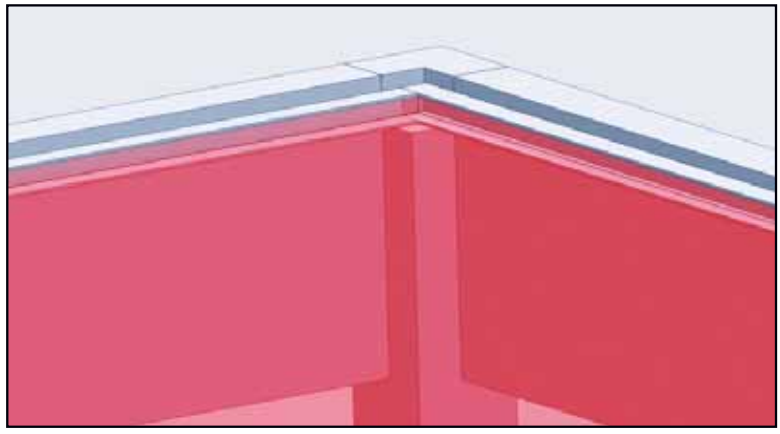

*Fig. 6 : Fixation du U sur les tasseaux*

Sur les tasseaux de la table, nous fixons un U en aluminium qui va servir à insérer un ruban de diodes (voir photo 5) et du plexiglas EndLighten. La figure 6 illustre ce montage.

Bien entendu, il est fortement conseillé de caler la plaque (avec du feutre ou autre) pour éviter d'abîmer les diodes. Enfin, le tout est vissé aux pieds de la table. De plus, nous devons positionner le U en aluminium de telle manière que l'on puisse poser la plaque de projection par-dessus.

En outre, une astuce consiste à avoir une bordure de table assez grande. Ce détail donne deux avantages. Le premier est qu'il permet aux utilisateurs de s'accouder dessus sans que la manche de veste ne perturbe le bon fonctionnement du système. Le second est qu'il va permettre d'accueillir un vidéo-projecteur sur la face latérale de la structure. Ainsi, l'image projetée ne sera pas perturbée par l'ombre du vidéo-projecteur.

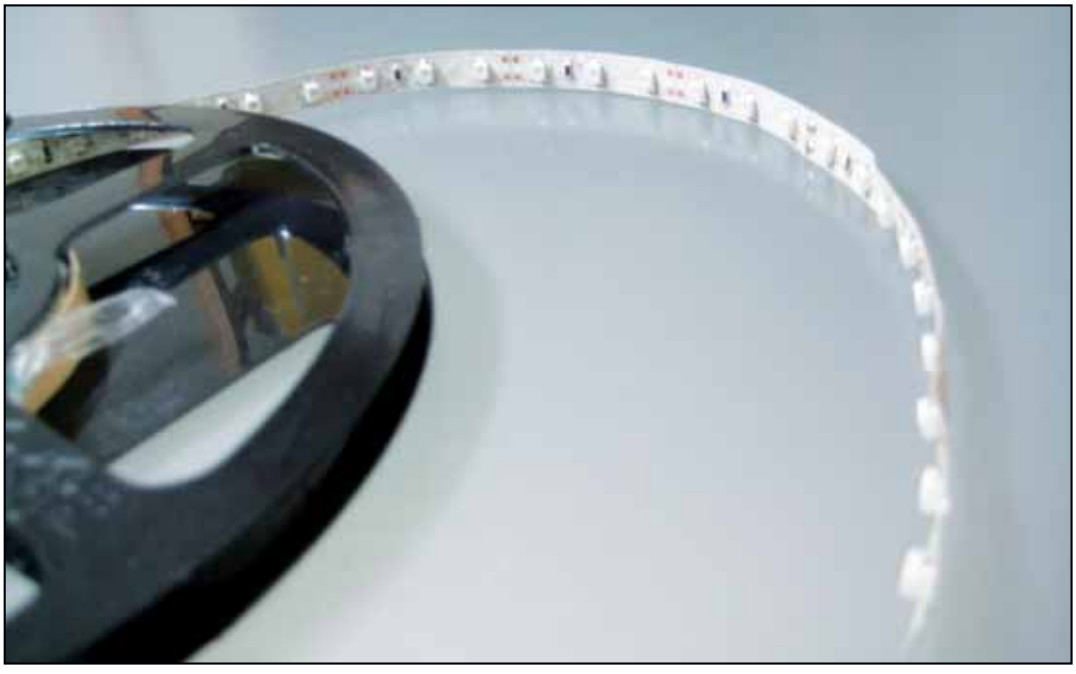

*Photo 5 : Ruban de diodes prêt à poser*

Nous venons d'illustrer avec quelle facilité il était possible de construire une table avec des matériaux peu coûteux. Il est cependant tout à fait possible de construire une table plus esthétique et de travailler davantage l'ergonomie et la robustesse, ou encore de transformer le système en tableau mural.

# 5 LA PARTIE LOGICIELLE

Le choix du système d'exploitation utilisé s'est porté vers la dernière mouture de la distribution Linux grand public Ubuntu, la version 10.04 sortie récemment, pour pouvoir bénéficier des versions des bibliothèques de programmation les plus récentes. Nous choisissons la version d'Ubuntu en 32 bits en raison des bibliothèques utilisées. La version 64 bits n'est donc pas appropriée pour l'instant.

Sur le système d'exploitation fraîchement installé, nous devons mettre à jour la liste des dépôts, installer les bibliothèques de développement ainsi qu'un éditeur (**vim** dans notre cas) :

```
~$ sudo apt-get update
~$ sudo apt-get install build-essential vim ubuntu-restricted-extras
```

## 5.1 Installation de CCV

Comme précédemment évoqué, nous voulons utiliser le logiciel CCV, qui bénéficie d'un développement soutenu par la communauté du libre. De ce fait, pour pouvoir obtenir ses sources, nous devons installer un système de gestion de sources. SVN (Subversion) est un logiciel permettant de gérer les modifications apportées aux différents documents en archivant chaque version. En ligne de commandes, nous installons cet outil de la façon suivante :

```
~$ sudo apt-get install subversion
```

Nous allons ensuite suivre les indications postées sur le forum *NUI Group Community* **[13]**.

La commande qui suit va permettre de créer dans le répertoire courant un dossier **ccv-Linux/** où sera téléchargée la dernière version de CCV :

```
~$ sudo svn co http://nuicode.svnrepository.com/svn/tbeta/trunk/tbeta/
   ; Linux/ ccv-Linux
```

Pour pouvoir faire des développements logiciels sur CCV, nous installons un IDE (*Integrated Development Environment*). Cet IDE est **CODE::BLOCKS**. Il a pour avantage d'être multiplate-forme, extensible et configurable. La raison de ce choix est que CCV a été compilé par l'intermédiaire de cet environnement. Cela va nous permettre de pouvoir compiler CCV pour avoir un exécutable utilisable. Dès à présent, nous pouvons lancer les scripts qui installeront la plupart des dépendances entre CCV et **CODE::BLOCKS**.

```
~$ cd ccv-Linux/scripts/ubuntu
~$ sudo chmod +x *.sh
~$ sudo ./install_codeblocks.sh
~$ sudo ./install_dependencies.sh
```

Ouvrons maintenant le fichier **Community Core Vision.cbp** dans **CODE::BLOCKS**, qui se trouve dans le répertoire **ccv-Linux/apps/addonsExamples/Codeblocks_8_linux/**. Il faut que nous fixions des liens à la console :

```
~$ cd /ccv-Linux/libs/fmodex/lib
~$ sudo rm libfmodexp.so libfmodex.so
~$ sudo ln -s libfmodex-4.22.00.so libfmodex.so
~$ sudo ln -s libfmodexp-4.22.00.so libfmodexp.so
```

De plus, il est à noter qu'Ubuntu ne possède pas de base de codecs pour lire la vidéo dans CCV. Afin d'éviter une erreur lors de l'exécution du logiciel, nous devons modifier une donnée dans un dossier. Ainsi, lors du démarrage, le logiciel lira un flux de données provenant d'une caméra et non plus la vidéo de base :

```
~$ cd /ccv-Linux/apps/addonsExamples/Codeblock_8_linux/bin/data
~$ sudo vi config.xml
```

La commande qui vient d'être exécutée édite le fichier **config.xml**. Dans le bloc **CAMERA SETTINGS**, il faut modifier les valeurs **<USE CAMERA>0</USE CAMERA>** en **<USE CAMERA>1</USE CAMERA>**, **<WIDTH>320</WIDTH>** en **<WIDTH>640</WIDTH>**, **<HEIGHT>240</HEIGHT>** en **<HEIGHT>480</HEIGHT>** et **<FRAMERATE>30</FRAMERATE>** en **<FRAMERATE>60</FRAMERATE>**.

Dès à présent, nous pouvons compiler le logiciel via l'interface graphique de **CODE::BLOCKS**. Pour ce faire, il faut lancer **CODE::BLOCKS**, ouvrir le fichier **Community Core Vision.cbp** qui se trouve dans le répertoire **~/ccv-Linux/apps/addonsExamples/Codeblocks_8_linux/** ou en ligne de commandes, pour créer un exécutable :

```
~$ cd /ccv-Linux/apps/addonsExamples/Codeblocks_8_linux
~$ sudo chmod +x cb_build_runner.sh
~$ sudo 'codeblocks Community Core Vision.cbp'
```

Après la compilation, il faut réparer les liens :

```
~$ cd /ccv-Linux/apps/addonsExamples/Codeblock_8_linux/bin/libs
~$ sudo rm libfmodexp.so libfmodex.so
~$ sudo ln -s libfmodex-4.22.00.so libfmodex.so
~$ sudo ln -s libfmodexp-4.22.00.so libfmodexp.so
```

## 5.2 Installation du driver pour la caméra PS3

Pour lire le flux vidéo de la caméra PS3, CCV utilise les libunicap. Mais nous devons appliquer un patch pour que cette bibliothèque puisse fonctionner avec notre *tracker* CCV. Pour cela, nous avons suivi les instructions indiquées en **[14]** :

```
~$ wget http://unicap-imaging.org/downloads/libunicap-0.9.8.tar.gz
~$ tar xvf libunicap-0.9.8.tar.gz
~$ cd libunicap-0.9.8
~$ wget http://melka.one.free.fr/mt/files/unicap-gspca.patch
~$ patch -p0 < unicap-gspca.patch
```

Maintenant que le patch est appliqué, on va pouvoir installer cette bibliothèque :

```
~$ sudo apt-get install intltool pkg-config libraw1394-dev libraw1394-11
~$ ./configure
```

On doit avoir cette sortie à la console :

```
Configuration:
  libraw1394 version: RAW1394_1_1_API
  installation goes to: /usr/local
```

```
Plugins:
  video-to-1394:      yes
  IIDC 1394 camera:   yes
  video-4-linux:      yes
  video-4-linux v2:   yes
```

Enfin, on va la compiler et l'installer :

```
~$ make
~$ sudo make install
```

Nous allons maintenant installer un patch pour le driver de la caméra PS3. Pour ce faire, nous devons dans un premier temps installer des paquets supplémentaires :

```
~$ sudo apt-get install kernel-pakage libncurses5-dev bzip2 linux-source
```

Puis, en tant que super utilisateur, nous allons décompresser les sources du noyau Linux :

```
~$ sudo su
~$ cd /usr/src
~$ tar --bzip2 -xvf linux-source-2.6.32.tar.bz2
~$ ln -s linux-source-2.6.32 linux
```

Nous pouvons maintenant télécharger le driver pour remplacer l'ancienne version, en suivant les indications de **[5]** :

```
$ cd /usr/src/linux
$ wget ucfilespace.uc.edu/~thrunml/ov534.c
$ rm drivers/media/video/gspca/ov534.c
$ mv ov534.c drivers/media/video/gspca/
```

Nous pouvons alors recompiler le driver :

```
~$ cd /usr/src/linux
~$ cp /usr/src/linux-headers-$(uname -r)/Module.symvers /usr/src/linux
~$ make oldconfig
~$ make modules_prepare
~$ make SUBDIRS=drivers/media/video/gspca modules
```

Et enfin, nous pouvons installer le driver :

```
~$ cp drivers/media/video/gspca/gspca_ov534.ko /lib/modules/$(uname -r)/
   ; kernel/drivers/media/video/gspca
~$ depmod
```

Nous pouvons dès à présent utiliser le nouveau driver :

```
~$ modprobe -r gspca-ov534
~$ modprobe gspca-ov534 videomode=04
```

Les différents modes de vidéo proposés sont alors :

```
00: 640x480 15fps
01: 640x480 30fps
02: 640x480 40fps
03: 640x480 50fps
04: 640x480 60fps
10: 320x240 30fps
11: 320x240 40fps
12: 320x240 50fps
13: 320x240 60fps
14: 320x240 75fps
15: 320x240 100fps
16: 320x240 125fps
```

En lançant CCV comme suit, nous remarquons que le débit plafonne à 30fps. Nous pensons qu'il s'agit probablement de la version de CCV qui pose problème et attendons la sortie de la prochaine version :

```
~$ cd /Logiciels/ccv-Linux/apps/addonsExamples/Codeblocks_8_linux/bin
~$ sh clickToLaunchApp.sh
```

## 5.3 Installation des programmes de test de PyMT :

Les instructions qui suivent sont donc exécutées dans le terminal par le super utilisateur.

La première étape consiste à installer les différentes bibliothèques dont dépend PyMT :

```
~$ sudo apt-get install python-setuptools python-pygame python-opengl
   ; python-numpy python-gst0.10 python-cairo python-imaging
   ; gstreamer0.10-plugins-good
```

La seconde étape consiste à installer la bibliothèque PyMT :

```
~$ sudo easy_install pymt
```

Pour vérifier que tout est bien en place, il faut importer la bibliothèque PyMT. Le premier import avec cette bibliothèque va permettre de le configurer :

```
~$  python
Python 2.6.5 (r265 :79063, Apr 16 2010, 13 :57 :41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pymt
```

À ce stade, nous allons télécharger des exemples de code concernant PyMT :

```
~$ wget http://pymt.googlecode.com/files/pymt-examples-0.4.tar.gz
```

Une fois le téléchargement terminé, il faut décompresser l'archive pour pouvoir utiliser les données :

```
~$ tar xzf pymt-examples-0.4.tar.gz
```

Nous pouvons dès à présent aller dans le dossier que nous venons de créer pour lancer l'exemple, qui s'affiche comme dans la figure 7.

```
~$ cd pymt-examples-0.4/
~$ python laucher.py
```


*Fig. 7 : Programme d'exemple de PyMT*

# 6 LA CALIBRATION

La calibration de CCV est une étape très importante qui consiste à définir les différents paramètres qui permettent la visualisation d'un objet sur la surface tactile de la table :

```
~$ cd /Logiciels/ccv-Linux/apps/addonsExamples/Codeblocks_8_linux/bin
~$ sh clickToLaunchApp.sh
```

Sur la sortie écran qui apparaît se trouvent des paramètres à régler, comme illustré en figure 8.

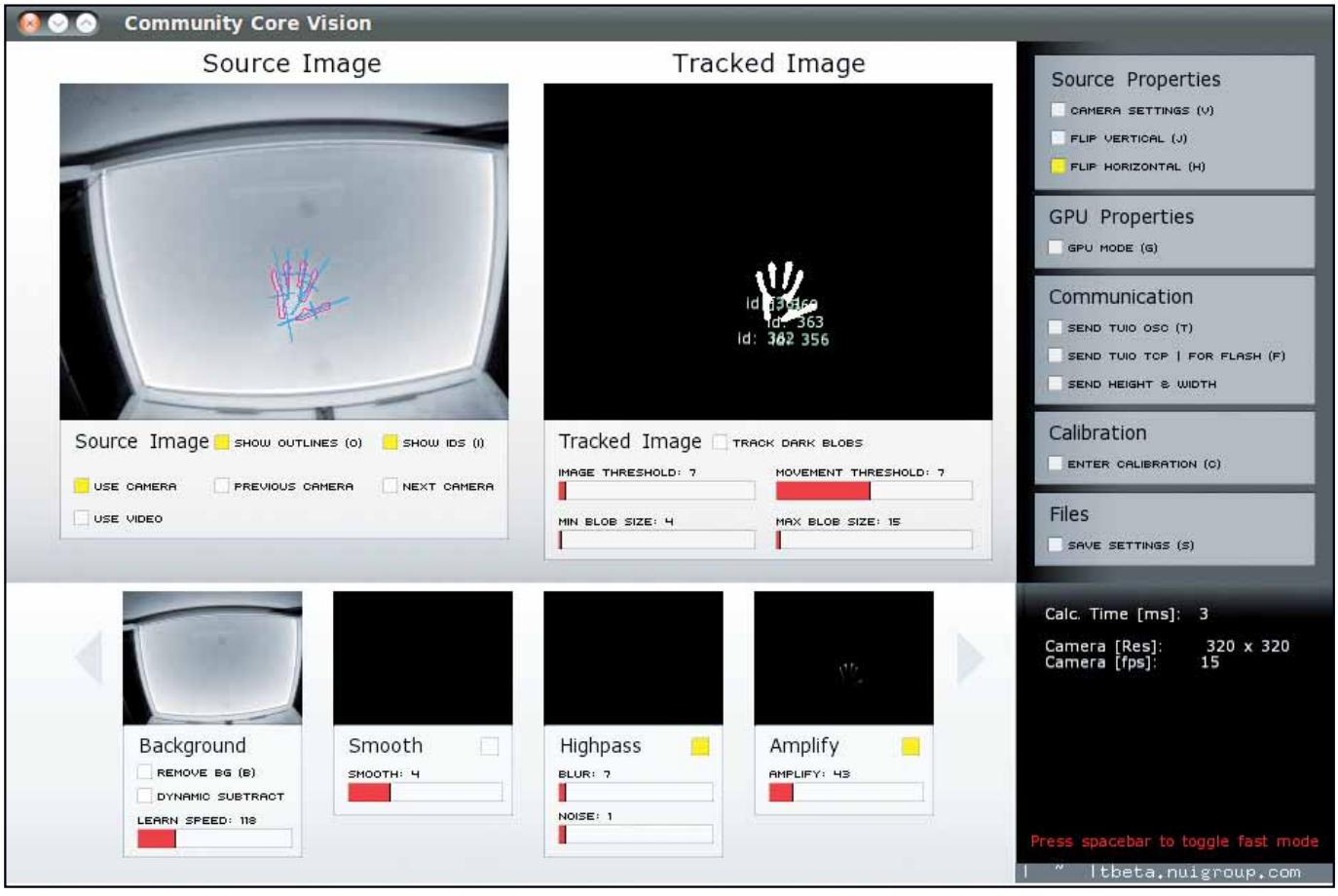


*Fig. 8 : Programme de calibration de CCV*

Nous allons voir à quoi ces derniers correspondent, grâce aux explications disponibles en **[15]**.

Intéressons-nous à « Tracked Image » (« suivi d'image ») :

- « Min blob size » correspond à la taille minimale de l'objet détecté.
- « Max blob size » correspond à la taille maximale de l'objet détecté.
- « Image Threshold » correspond au seuillage en fonction de l'intensité.
- « Mouvement Threshold » permet de définir la sensibilité de la détection des mouvements. Plus cette valeur est élevée, plus le mouvement devra être ample pour pouvoir être détecté.

La partie « Background » (« arrière-plan ») :

- « Remove BG » permet de retirer les artefacts de départ. Pour ce faire, cette fonction capture l'image de départ et la soustrait aux flux d'images à venir.
- « Dynamic Substract » permet d'ajuster l'image de fond lorsque l'environnement lumineux change fréquemment. Cette fonction retire dynamiquement les éléments immobiles.
- « Learn Speed » détermine la vitesse d'apprentissage de l'image de fond.

Smooth :

- « Smooth » permet de lisser l'image et de filtrer les bruits.

Highpass :

- « Blur » permet de contrôler le flou et de laisser les parties les plus brillantes.

- « Noise » permet d'enlever les bruits après le contrôle du flou et ainsi d'obtenir des formes plus nettes.

Amplify :

- « Amplify » donne la possibilité de jouer sur la saturation de la lumière pour amplifier la détection dans le cas où l'éclairage est trop faible.

Une fois le paramétrage achevé, il faut sélectionner dans la partie Communication, ***SEND TUIO OSC***, qui permet d'envoyer les messages avec le protocole TUIO et ***SEND HEIGHT & WIDTH*** pour envoyer les coordonnées. On peut ensuite sauvegarder ces paramètres en sélectionnant ***SAVE SETTINGS*** puis entrer en calibration via ***ENTER CALIBRATION***. Cela va ouvrir une fenêtre qui permettra d'effectuer les réajustements afin de pallier aux distorsions de la lentille utilisée, comme sur la figure 9.

Par défaut, nous remarquons que nous avons une vitesse de 15 images par seconde, ce qui est très faible pour une utilisation fluide. Nous allons donc chercher à augmenter la vitesse de la caméra.


*Fig. 9 : Fenêtre de calibration de CCV*

# 7 LET'S CODE...

Nous avons choisi d'utiliser le langage de programmation Python, car la bibliothèque PyMT existe et a l'avantage de prendre en compte directement l'accélération graphique OpenGL ainsi que la réception des points via le protocole TUIO sur le protocole UDP. A contrario, nous aurions pu utiliser Flash, qui lui nécessite de convertir les points vers le protocole TCP et a l'inconvénient de consommer plus de ressources pour le même type d'application. Les férus de Flash pourront donc se pencher sur cette alternative.

## 7.1 Hello Tactile World #1

```
from pymt import *
label=MTLabel(label='Hello World',color=(0.5,0.5,0,1))
runTouchApp(label)
```

Ci-dessus, nous voyons un petit programme en Python qui utilise la bibliothèque PyMT. Nous allons utiliser ici un *widget* `MTLabel` pour afficher un texte à l'écran. On lui donne en paramètres le texte à afficher ainsi que la couleur d'affichage. Enfin, on lance la boucle d'événements de PyMT avec la commande `runTouchApp`.

## 7.2 Hello Tactile World #2

```
from pymt import *
label=MTLabel(label='Hello World',color=(0.5,0.5,0,1))
scatter=MTScatterWidget(size=(100,100))
scatter.add_widget(label)
runTouchApp(label)
```

Ce second code affiche un texte avec lequel nous pouvons interagir. Pour ce faire, on utilise encore un **MTLabel** comme précédemment pour afficher un texte. Ensuite, on crée un **MTScatterWidget** qui possède la propriété d'être déplaçable et dont la taille peut être modifiée avec deux doigts. Afin que le texte puisse bénéficier des propriétés de ce dernier, nous plaçons le **MTLabel** dans le **MTScatterWidget** grâce à **add_widget**. Pour finir, on lance la boucle d'événements, **runTouchApp**.

## 7.3 Un programme plus complet

Pour les plus curieux qui veulent avoir un exemple de programme *multitouch*, nous avons développé une application amusante qui simule l'attraction gravitationnelle entre les planètes. Nous allons commenter ce programme, mais les lecteurs les plus pressés pourront directement le télécharger à partir de **[16]** pour le tester.

Pour construire un tel programme, nous allons utiliser les bibliothèques **pymt**, **math** et **OpenGL.GL** :

```
1   import random
2   from pymt import *
3   from math import sqrt
4   from OpenGL.GL import *
```

Avant de commencer à implémenter tout l'algorithme, nous devons nous intéresser à l'objet en lui-même, c'est-à-dire définir ce qu'est une planète. La définition d'une planète va se faire dans une classe à part. Ainsi, s'il y a des changements à effectuer, nous pourrons nous référer à cette classe.

```
5   class Planet(MTWidget):
6   """
7 une petite planete avec les paramétres
8 - pos (x,y) parametre pos dans MTWidget
9 - vecteur vitesse
10 - masse
11 """
12
13 def __init__(self,**kwargs):
14   self.vitesse= kwargs.setdefault('vitesse',[0,0])
15   self.masse = kwargs.setdefault('masse',2)
16   self.radius = 2.0
17   self.color = (random.random(), random.random(), random.random(), 1.0)
18   super(Planet,self).__init__(**kwargs)
19
20 def draw(self):
21   glColor4f(self.color[0], self.color[1], self.color[2], self.color[3])
22   drawCircle(pos=(self.x,self.y),radius=self.radius)
```

Nous allons définir une planète comme étant un **MTWidget** dans la bibliothèque de PyMT. Pour ce faire, nous allons utiliser l'héritage (ligne 5) pour avoir les propriétés de **MTWidget** et pour avoir la possibilité de redéfinir les méthodes dont on a besoin. Nous commençons par caractériser une planète par sa vitesse, sa masse, son rayon et sa couleur (lignes 13 à 18). L'avantage de **MTWidget** est que sa forme est modifiable. Elle est définie par la méthode **draw()** à laquelle nous apportons nos modifications. En effet, nous voulons qu'elle ait une forme ronde pour représenter une planète. Nous allons donc définir la couleur en utilisant une méthode de la bibliothèque OpenGL, puis définir le cercle par sa position et son rayon (lignes 20 à 22).Une fois les planètes définies, nous allons nous intéresser aux phénomènes physiques. Dans notre cas, nous allons nous concentrer sur la physique relativiste, qui énonce que la force (F) d'attraction entre deux corps est fonction de leurs masses (M et M'), de la distance (D) qui les sépare et d'une constance de gravitation (G) selon la formule suivante :

$$\vec{F} = \vec{G}\frac{M.M'}{D^2}$$

De plus, nous devons connaître le principe fondamental de la dynamique qui énonce que la somme des forces est égale au produit de la masse (M) par l'accélération (a) :

$$\sum \vec{F} = M.\vec{a}$$

Enfin, voici la formule de la quantité de mouvement (q), qui est le produit de la vitesse (v) par masse (M) :

$$q = v.M$$

```
23   class Univers(MTWidget):
24 """Phénomène physique dans l'univers"""
25
26 gravitation =  6
27
28 def __init__(self,**kwargs):
29   super(Univers, self).__init__(**kwargs)
30   self.planetList = []
31
32 def on_touch_down(self,touch):
33   planet = Planet(x=touch.pos[0],y=touch.pos[1])
34   self.add_widget(planet)
35   self.planetList.append(planet)
36   self.on_collision()
37   return True
```

Nous voulons maintenant mettre en place l'algorithme qui définit les interactions entre les différentes planètes. Mais nous allons définir préalablement par quoi nous représentons notre univers. Nous voulons définir notre univers comme étant un **MTWidget** pour les mêmes raisons que celles citées précédemment (ligne 23). Dans notre cas, nous la représentons par une **planetList** qui est une liste des planètes que nous posons sur la table (lignes 28 à 30). Puis, nous devons redéfinir la méthode **on_touch_down()** de **MTWidget** pour appliquer d'autres propriétés. En particulier, lorsque nous posons un doigt sur la table, nous ne voulons qu'ajouter une planète à la liste avec sa position, puis l'ajouter dans le widget et faire en même temps un test de collision que nous expliquerons plus tard (lignes 32 à 37).

```
38 def deplacement(self):
39   Qx = 0.0
40   Qy= 0.0
41   if len(self.planetList) >= 1 :
42     for planet1 in self.planetList :
43       for planet2 in self.planetList :
44         if planet1 != planet2:
45           largeur = planet2.x - planet1.x
46           longeur = planet2.y - planet1.y
47           distance = sqrt(pow(largeur,2) + pow(longeur,2))
48           force = Univers.gravitation * (planet2.masse * planet1.masse /
           ; pow(distance,2))
49           A0 = force / planet1.masse
50           Normlarg = largeur / distance
51           Normlong = longeur / distance
52           planet1.vitesse[0] -= A0 * Normlarg
53           planet1.vitesse[1] -= A0 * Normlong
54       if planet1.vitesse[0] > 1 :
55         planet1.vitesse[0] = 1
56       if planet1.vitesse[0] < -1 :
57         planet1.vitesse[0] = -1
58       if planet1.vitesse[1] > 1 :
59         planet1.vitesse[1] = 1
60       if planet1.vitesse[1] < -1 :
61         planet1.vitesse[1] = -1
62       planet1.x -= planet1.vitesse[0]
63       planet1.y -= planet1.vitesse[1]
64     for planet in self.planetList :
65       if planet.x > self.width:
66         planet.x = 1
67         planet.y = self.height - planet.y
68       if planet.x < 0:
69         planet.x = self.width -1
70         planet.y = self.height - planet.y
71       if planet.y > self.height:
72         planet.y = 1
73         planet.x = self.width - planet.x
74       if planet.y < 0:
75         planet.y = self.height -1
76         planet.x = self.width - planet.x
```

Dans la classe **Univers**, nous devons à présent implémenter le code du déplacement. Les planètes étant contenues dans une liste, nous allons faire une double boucle **for** pour voir l'interaction gravitationnelle entre les planètes deux à deux.

Le principe utilisé dans la boucle est simple. Dans un premier temps, il faut calculer la distance entre les deux corps (lignes 45 à 47), puis calculer la force gravitationnelle à partir de la formule de Newton (ligne 48). Nous pouvons donc trouver l'accélération (ligne 49) et la vitesse, en intégrant cette dernière (lignes 50 à 53), puis calculer la nouvelle position de la planète (lignes 62 et 63). Pour finir avec cette méthode, nous devons vérifier si la planète ne sort pas de la fenêtre (lignes 64 à 66).

```
77 def on_collision(self):
78   Qx = 0.0
79   Qy= 0.0
80   if len(self.planetList) >= 1 :
81     for planet1 in self.planetList :
82       for planet2 in self.planetList :
83         if planet1 != planet2:
84           largeur = planet1.x - planet2.x
85           longeur = planet1.y - planet2.y
86           distance = sqrt(pow(largeur,2)+pow(longeur,2))
87           if distance < (planet1.radius + planet2.radius) :
88             Qx += (planet1.masse * planet1.vitesse[0]) +(planet2.masse
             ; * planet2.vitesse[0])
89             Qy += (planet1.masse * planet1.vitesse[1]) +(planet2.masse
             ; * planet2.vitesse[1])
90             sommeMasse = planet1.masse + planet2.masse
91             if planet1.radius > planet2.radius :
```

```
92            planet1.vitesse[0] = Qx / sommeMasse
93            planet1.vitesse[1] = Qy / sommeMasse
94            planet1.masse = sommeMasse
95            planet1.radius += 1.0
96            planet1.x -= planet1.vitesse[0]
97            planet1.y -= planet1.vitesse[1]
98            if planet1.radius > 16 :
99              planet1.radius = 16
100           if planet1.masse > 16 :
101             planet1.masse = 16
102           self.planetList.remove(planet2)
103           self.remove_widget(planet2)
104         else :
105           planet2.vitesse[0] = Qx / sommeMasse
106           planet2.vitesse[1] = Qy / sommeMasse
107           planet2.masse = sommeMasse
108           planet2.radius += 1.0
109           Planet2.x -= planet2.vitesse[0]
110           Planet2.y -= planet2.vitesse[1]
111           if planet2.radius > 16 :
112             planet2.radius = 16
113           if planet2.masse > 16 :
114             planet2.masse = 16
115           self.planetList.remove(planet1)
116           self.remove_widget(planet1)
```

Le test de collision va définir ce qu'il se produit lorsque deux corps entrent en contact. Nous allons ici aussi faire une double boucle **for** pour calculer la distance qui sépare les planètes deux à deux. Ce calcul va permettre de connaître les cas où il y a une collision (ligne 87). Dès lors, nous allons séparer le problème en deux cas, afin de permettre à la planète la plus volumineuse de rester sur l'écran et d'éliminer l'autre (lignes 91 à 116). Nous allons donc calculer la vitesse résultante en appliquant la formule de la quantité de mouvement, puis calculer la position.

```
117 def on_update(self):
118   self.deplacement()
119   self.on_collision()
120   self.on_draw()
121   super(Univers,self).on_update()
```

Pour finir avec cette classe, nous redéfinissons la méthode qui rafraîchit l'image. Dans cette dernière, nous devons appeler en boucle la méthode qui calcule le déplacement des planètes, puis vérifier s'il y a des collisions pour enfin remplacer l'image affichée.

```
122 if __name__ == '__main__':
123 w = MTWindow(style={'bg-color': (0, 0, 0, 1),'draw-background':True })
124 univ = Univers(size=(1280,800))
125 w.add_widget(univ)
126 w.draw()
127 runTouchApp()
```

La dernière partie est le **main**. C'est ici que nous allons créer l'objet **Univers** pour le mettre dans une fenêtre de type **MTWindow**. Puis on lance la boucle d'événements avec **runTouchApp()**.

Il y a cependant quelques remarques à faire à propos de ce programme. Pour simplifier le problème, nous avons fixé la constante de gravitation (ligne 26) au lieu de la calculer. De plus, pour que ce programme soit jouable, nous avons limité la taille et la masse des planètes (lignes 98 à 101 et 111 à 114). Et pour plus de cohérence avec le calcul de la vitesse résultante lors d'une collision, nous avons limité la vitesse (lignes 54 à 63).

## CONCLUSION

Avec l'achèvement de ce projet, on peut remarquer que la réalisation d'une table tactile « comme vue à la TV » est désormais à la portée de tous, pour peu que l'on sache faire preuve d'un peu de patience et de minutie.

Le *middleware* est désormais pleinement utilisable, il ne manque plus que la killer-application qui va faire décoller l'usage de ce nouveau type d'interface homme-machine.

N'en doutons pas, de l'effervescence du monde libre pour cette technologie ne manquera sans doute pas de bientôt produire le nouveau gestionnaire de fenêtres 3D à la mode ou le jeu à l'interface révolutionnaire « qui tue ». À moins que l'engouement ne vienne du monde professionnel à travers une interface de travail collaborative surproductive.

En tout cas, les différents points évoqués ne sont que quelques idées parmi tant d'autres. Une fois la table construite, il ne reste plus qu'à laisser les idées faire leur chemin et laisser libre cours à son imagination pour créer les programmes de demain. ■

### Liens


- **[1]** brevet américain 4561017 du 24 décembre 1985, intitulé « *Graphic input apparatus* »
- **[2]** http://nuigroup.com/forums/
- **[3]** http://peauproductions.com/
- **[4]** http://en.wikipedia.org/wiki/PlayStation_Eye
- **[5]** http://bear24rw.blogspot.com/2009/11/ps3-eye-driver-patch.html
- **[6]** http://www.projectorcentral.com/
- **[7]** http://ccv.nuigroup.com/
- **[8]** http://reactivision.sourceforge.net/
- **[9]** http://nuigroup.com/touchlib/
- **[10]** http://www.tuio.org/?tuio11
- **[11]** http://www.reactable.com/
- **[12]** http://pymt-txzone.net/
- **[13]** http://nuigroup.com/forums/viewthread/8691/
- **[14]** http://melka.one.free.fr/mt/index.php/2010/02/how-to-use-ccv-under-linux/
- **[15]** http://wiki.nuigroup.com/Getting_Started_with_tbeta
- **[16]** http://www.gnulinuxmag.com/table/