POLYTECH LILLE

Université de Lille 1 SCIENCES ET TECHNOLOGIES

Rapport de Projet

Plate-forme tangible

Szwechowiez Maxime et Maia Stéphane

IMA4

Mai 2016

Remerciements

Nous tenions dans un premier temps à remercier, toute l'équipe pédagogique de Polytech'Lille et les responsables de la formation Informatique, Microélectronique et Automatique de nous avoir aidé à réaliser notre projet.

Nous souhaitons également remercier nos encadrants à savoir M. Boé, M. Vantroys ainsi que M. Redon pour leur disponibilité et leurs réponses à nos questions.

Sommaire

Remerciements 2
Introduction 4
Cahier des charges 5
Matériel utilisé 5
Objectifs et déroulement du travail 5
Réalisation du programme 7
Partie Jeu 7
Présentation de la technique de TileMapping 7
Déroulement de la programmation du jeu 8
Partie Réalité augmentée 11
Présentation de la bibliothèque 11
Mise en place du programme 12
Partie Rassemblement 15
Difficultés 15
Solutions 15
Problèmes rencontrés et améliorations possibles 17
Problèmes rencontrés 17
Améliorations possibles 17
Conclusion 19

Introduction

Le contexte du sujet se place dans une époque où de plus en plus de jeux vidéo utilisent la technologie de la réalité augmentée. Sur un air de Mario, nous avons voulu nous essayer à cette expérience.

A la différence d'un jeu de type Skylanders ou Amiibo où le joueur peut amener son personnage à l'intérieur du jeu, nous avons décidé que l'utilisateur devra pouvoir créer l'intégralité de son niveau à l'aide de cubes.

Nous avons décidé de nous séparer le travail, c'est-à-dire une personne pour concevoir la partie réalité augmentée et une autre pour la partie graphismes et gameplay du jeu.

Nous verrons donc dans un premier temps le cahier des charges et les objectifs, puis nous présenterons le travail réalisé concernant la partie réalité augmentée puis celui effectué pour la partie jeu.

Cahier des charges

Matériel utilisé

Pour réaliser notre projet nous n'avons eu besoin que d'une webcam afin de réaliser les acquisitions d'image au fur et à mesure du temps ainsi que d'un ordinateur afin de pouvoir jouer au jeu.

Nous nous sommes également servis de l'imprimante 3D du Fabricarium de Polytech afin de réaliser nos cubes de réalité augmentée.

Objectifs et déroulement du travail

En ce qui concerne le cahier des charges il nous a été demandé de concevoir un système de plateforme tangible permettant l'intégration d'éléments réels dans un univers virtuel. Ces éléments devront avoir la forme de cubes et être reconnus peu importe leur position, c'est-à-dire qu'il n'est pas nécessaire de les poser sur un élément tangible pour permettre la détection. Le temps de détection et d'intégration devait être le plus rapide possible pour ainsi permettre la jouabilité.

Concernant le déroulement du travail, nous avons dans un premier temps réfléchi au type de jeu que nous souhaitions réaliser. Pour avoir une idée plus précise de la meilleure solution possible nous avons demandé aux gens autour de nous afin de connaitre leur avis. Nous leur avons posé la question de quel type de position des cubes ils préféraient entre un positionnement à plat des cubes avec une vue du dessus ou alors d'un positionnement en hauteur avec une vue de face. La deuxième possibilité fut choisie à l'unanimité et nous sommes donc partie sur une conception de ce type à l'aide d'un jeu axé Mario – like.

Nous avons ensuite tenté d'intégrer la bibliothèque C que nous avons utilisée sur une Raspberry Pi afin de rendre notre projet plus portatif mais cela fut soldé par un échec car la Raspberry n'est pas assez puissante pour utiliser cette bibliothèque.

C’est à ce moment que nous avons décidé de nous partager le travail en deux comme expliqué précédemment avec un travaillant sur la partie réalité augmenté et l’autre sur la partie jeu.

La quatrième grosse partie de notre projet consistait en la réunion de nos deux programmes afin de faire tout fonctionner ensemble.

Réalisation du programme

Partie Jeu

Présentation de la technique de TileMapping

Pour la partie jeu, nous nous sommes d'abord renseignés sur les différentes techniques pour programmer un jeu comme Mario. Nous avons découvert la librairie SDL qui est une librairie du langage de programmation C. Cette bibliothèque permet de développer des applications graphiques en 2D comme des jeux de plateforme, ce qui correspond tout à fait à notre projet de jeu Mario. Cette bibliothèque permet notamment d'utiliser une technique appelé TilesMapping qui est une technique permettant de créer n'importe quelle niveau d'un jeu brique par brique. Cela coïncide exactement avec nos cubes dans la réalité. Pour utiliser le TilesMapping, nous aurons en fait un fichier texte externe au programme qui contiendra, pour chaque type de cube (cube de ciel bleu, cube de brique, cube de lave ...) leur type (brique= mur, ciel ou nuage = ciel , lave = danger) mais aussi la matrice du niveau, qui va être lu par notre programme. Notre programme va parcourir la matrice est pour chaque valeur coller dans la fenêtre de l'image du cube correspondant à un cube compris dans un TileSet (bitmap comportant tous les différentes images de cubes ex : 5 → image de ciel).

Illustration 1: TileSet

Illustration 2: Un jeu réalisé grâce à la technique de TileMapping

Une fois que notre programme aura parcouru l'intégralité de la matrice, on aura à l'écran un niveau de jeu agréable. Le gros avantage de cette technique est l'infinité de solutions possibles en création de niveau, car on peut mettre n'importe quel cube à n'importe quel endroit. On peut aussi modifier les images correspondant au cube, par exemple on peut imaginer un TileSet avec les mêmes cubes mais recouvert de neige pour faire le même niveau avec celle-ci.

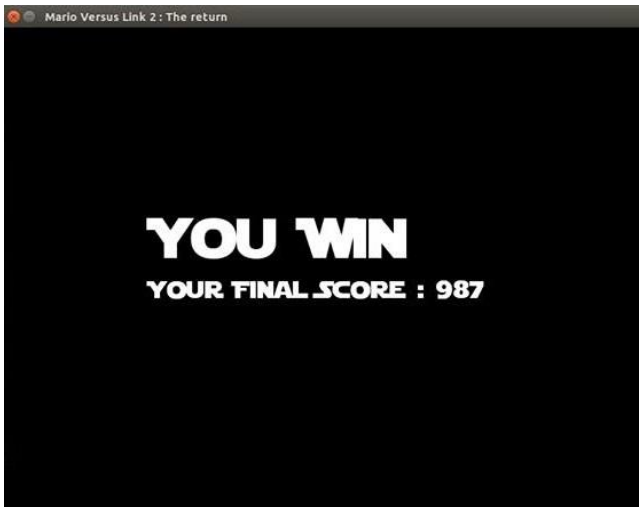
Déroulement de la programmation du jeu

Pour nous permettre d'avoir un niveau tout en longueur et qui dépasse donc la largeur de la fenêtre d'affichage, nous avons implémenté à notre programme un scrolling, c'est à dire un balayage de la fenêtre sur le niveau, en suivant la position du personnage. Une fois le personnage arrivé à une certaine limite du bord de la fenêtre, si il continue d'avancer dans ce sens , la niveau va défiler à l'écran le temps du déplacement du personnage.

Ensuite arriva la plus grande difficulté de cette partie : la gestion des collisions. Pour avoir une impression d'avoir de vrai mur à travers lesquels nous ne pouvons traverser, l'ordinateur va imaginer à l'avance nos déplacements. Si on dit à notre personnage de se déplacer, l'ordinateur va créer un personnage « bis » invisible à l'écran, va ensuite le déplacer comme le demande le joueur et tester si le personnage se retrouve dans un

mur. Si ce n'est pas le cas, le personnage bis deviendra le vrai personnage affichable à l'écran, inversement s'il se retrouve dans un mur il retournera une erreur qui empêchera le personnage visible de se déplacer, on aura donc bien une impression de bloquer contre un mur. De la même façon, avec les dangers, excepté qu'au lieu d'être bloqué on sera expulsé vers le haut comme si le personnage c'était blessé, et il perd une vie.

Pour ajouter de l'attraction à notre jeu, nous aussi mise en place à un système de vie avec des cœurs. Le personnage part avec dix vies donc dix cœurs pleins. Dès qu'il perd une vie, un cœur se vide. Une fois qu'il est à court de vie, il a perdu et se retrouve devant un écran Game Over. Nous avons aussi mis en place un système de score qui est enfaîte un timer décroissant. Le but du jeu étant d'arriver jusqu'au bout du niveau en un minimum de temps pour avoir le score maximum possible. Si le score arrive à 0, le jeu est perdu, le joueur arrive devant un écran Game Over. Si le joueur gagne on arrive sur l'écran de victoire.



Nous avons aussi ajouté un Menu à notre jeu, permettant au joueur de choisir le personnage qu'il souhaite.

L'animation et l'affichage d'un personnage se fait à partir d'un fichier image appelé charset, comprenant toutes les positions que peux avoir notre personnage, il est donc possible de rajouter autant de personnage que l'on veut tant qu'on a le fichier image correspondant. Notre programme va simplement faire défiler les différentes positions de notre personnage. Pour avoir la sensation que notre personnage se déplace on va afficher l'image avec le personnage avec la jambe droite devant, puis celle où il a la jambe gauche devant, et ainsi de suite.

Partie Réalité augmentée

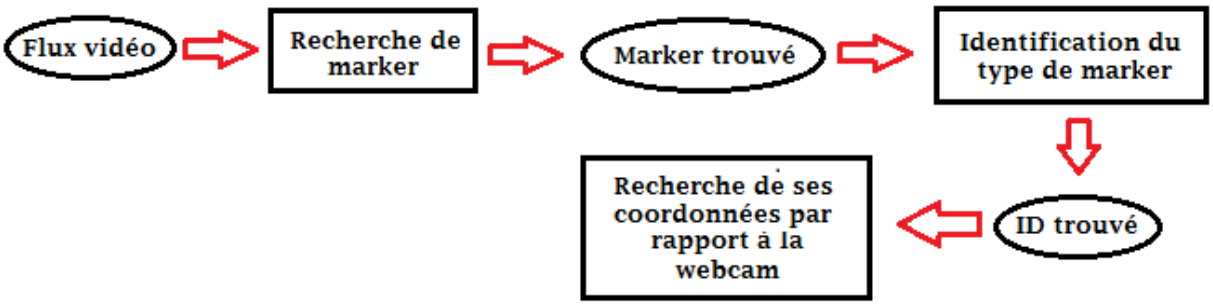
Présentation de la bibliothèque

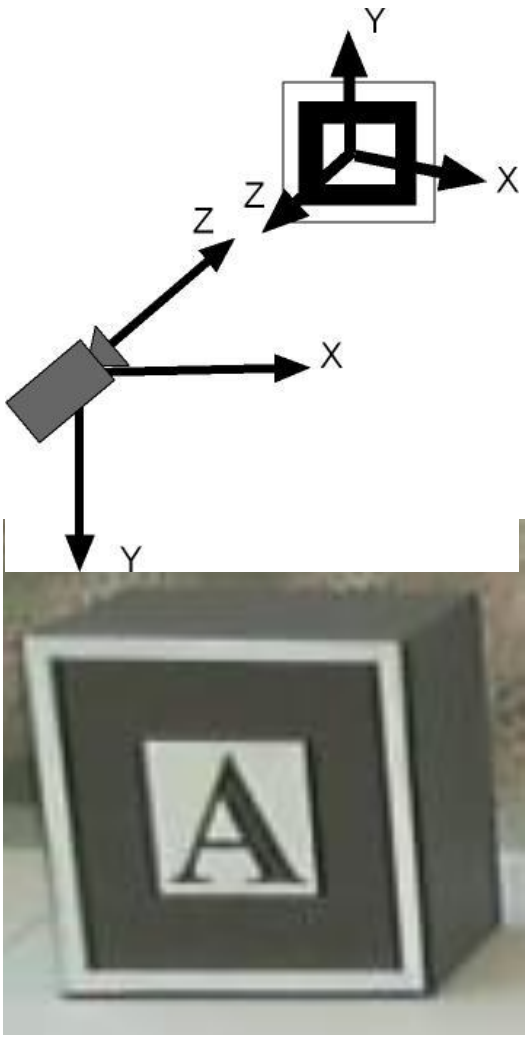
Pour la réalisation de notre projet nous avons décidé d'utiliser la bibliothèque Artoolkit qui est une bibliothèque C en open source dédiée à la création d'interfaces de réalité augmentée. Cette bibliothèque propose de multiples fonctionnalités allant de la détection de pattern appelé ArTag qui sont en fait des carrés aux bords noirs avec un symbole au centre permettant de les distinguer comme par exemple :

L'intérêt de cette bibliothèque est de permettre la détection des différents types de pattern et également leur position par rapport à la webcam mais également de pouvoir afficher sur l'écran, différents types d'images 3D sur l'écran de l'ordinateur à la position du pattern.

Mise en place du programme

Nous avons mis de côté cette dernière fonctionnalité car nous n'en avions pas l'utilité dans notre projet. La partie de la bibliothèque que nous avons utilisée fonctionne donc de la manière suivante :





Comme nous l'avons expliqué précédemment, la bibliothèque permet la détection de la position du marker par rapport à la position de la caméra. Ainsi comme on peut le voir sur l'illustration à gauche, il est difficile d'obtenir de cette façon de pouvoir placer les cubes dans un jeu vidéo. Pour remédier à ce problème nous avons décidé d'instaurer un cube de référence. Grâce à cela si le logiciel détecte notre cube de référence (à savoir le cube A). Comme nous l'avons expliqué précédemment dans la partie Jeu, celui repose sur le principe du Tile Mapping, qui est donc une matrice constitué de nombres qui vont identifier le type d'image à intégrer dans le jeu. Le cube A

sera donc le chiffre le plus en bas à gauche de notre matrice. Par la suite notre programme récupère les coordonnées de tous les markers qu'il détecte à l'écran et les stocke dans des variables associées à chacun de nos markers.

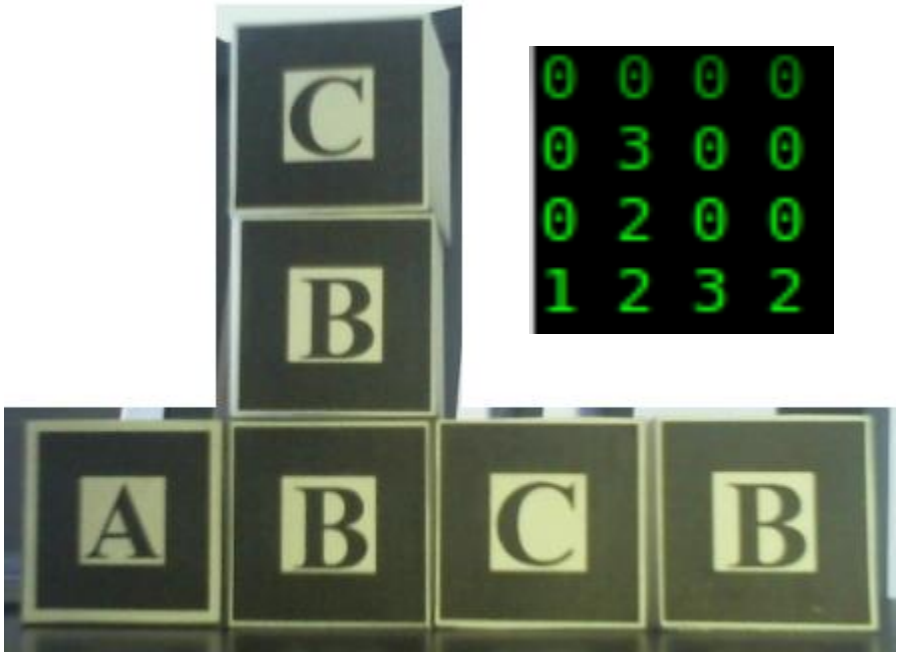
Ces coordonnés sont ensuite comparées aux coordonnées du cube de référence A, c'est-à-dire que nous utilisons une boucle for qui va étudier le nombre de cube qu'il est possible de placer entre le cube A et le cube dont on cherche à déterminer les coordonnées de la façon suivante :

```
if(marker_info[i].id == 1)
{
        for(h=0;h<=NBR_CUBE;h++)
        {
                coord_cube1y=patt_trans[1][3];
                if(coord_cube1y>=((coordy-7)-h*40) && coord_cube1y<=((coordy+7)-h*45))
                {
                        for(j=0;j<=NBR_CUBE;j++)
                        {
                                coord_cube1x=patt_trans[0][3];
                                if(coord_cube1x>=((coordx-6)+j*45) && coord_cube1x<=((coordx+6)+j*45))
                                {
                                        presence[j][h]=2;
                                }
                        }
                }
        }
}
```

Dans ce code :

- Coordx, coordy représentent les coordonnées de notre cube de référence.
- Patt_trans[1][3], Patt_trans[2][3] sont les coordonnées d'un marker trouvé par le programme en temps réel, il est nécessaire de les stockées car elles se modifient en temps réel.
- Presence[j][h] est la matrice utilisé pour le TileMapping, elle prend la valeur que nous lui imposons afin de différencier chaque type de cube dans le jeu, ici 2 représente un cube de sol basique alors que 5 représente par exemple un cube de ciel.

Nous avons donc effectué des tests en positionnant des cubes de différentes façons et d'afficher la matrice présence afin d'observer si son comportement est bien en adéquation avec ce que nous souhaitions et on a obtenu par exemple :



On voit donc que le programme de reconnaissance et fonctionnel.

Partie Rassemblement

Difficultés

La principale difficulté que nous avons rencontrée et le fait que les deux programmes doivent fonctionner en temps réel et de façon synchrone car comme nous l'avons expliqué précédemment le TileMapping requiert de lire dans un fichier texte.

Solutions

Pour pallier à ce problème nous avons décidé de mettre en place un système de fonctions threadés, c'est-à-dire que grâce à cela les deux programmes peuvent fonctionner en même temps sans avoir à lancer plusieurs exécutables.

Nous avons donc deux threads, l'un contenant le programme de réalité augmentée et l'autre le programme du jeu. Le point positif de ce fonctionnement en plus d'être moins encombrant pour l'utilisateur et de permettre de pouvoir utiliser un autre jeu. En effet le programme de réalité augmentée étant assez générale il est possible de coder un autre jeu de type vue du dessus par exemple et utilisant le TileMapping pour pouvoir totalement changer de jeu sans avoir à modifier le programme entièrement.

Nous avons dû cependant procéder à certains ajustements car nos deux programmes fonctionnaient correctement de façon indépendante mais une fois la réunion effectuée nous avons pu constater certains problèmes.

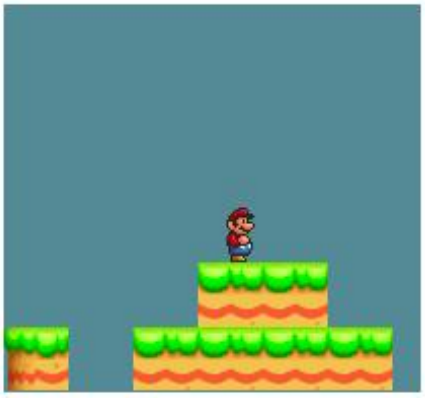
Le premier problème fut l'écriture et la lecture du fichier du TileMapping car nous obtenions des erreurs de segmentations fault entrainant une coupure de notre programme car il arrivait que les deux threads aient à utiliser le fichier texte en même temps. La solution que nous avons trouvé pour résoudre ce problème fut de mettre en place un système de Mutex afin d'empêcher que la lecture et l'écriture se passe en même temps.

Dans un second temps nous avons constaté que le programme réécrivait entièrement le fichier texte même si aucun changement ne se

produisait. On utiliser donc des ressources la plupart du temps pour rien. Ainsi dans le but d'optimiser un peu mieux notre programme nous avons mis en place une fonction détectant si un changement à lieu dans le placement des cubes et ainsi permettre l'écriture dans notre fichier texte. Dans le cas contraire le programme continue son exécution normale. Cette fonction est la suivante :

```
int checkmat(int presence[LARGEUR][HAUTEUR])
{
    int i,j;
    cpt = 0;
    for(j=LARGEUR-1;j>=0;j--)
            {
                for(i=0;i<LARGEUR;i++)
                {
                    cpt = cpt + presence[i][j];
                }
            }
    return cpt;
}
```

Après une série de test nous avons obtenu un résultat fonctionnel comme le montre l'exemple suivant :



Problèmes rencontrés et améliorations possibles

Problèmes rencontrés

Malgré le fait que le cahier des charges est respecté et que notre programme fonctionne, celui-ci possède certains défauts.

Le premier est que la présence de lumière est très importante pour la visibilité et la détection des cubes, car nous avons été plusieurs fois confronté au fait que nos cube n'était pas détecté par manque ou par trop de lumière.

La seconde difficulté est la taille des cubes utilisés. En effet, afin de pouvoir construire le niveau en entier il faut beaucoup de cube, mais beaucoup de cube implique beaucoup d'espace et que l'angle de vue de la caméra soit suffisamment important pour tous les voir. Nous avons tenté de miniaturiser les cubes mais ce que nous gagnions en taille nous le perdions en distance par rapport à la caméra.

Améliorations possibles

Une des améliorations des plus importantes que nous aurions aimé avoir réalisé et la construction d'un bras mécanique servant de support à une webcam ainsi qu'à un pico projecteur. En effet le principe de scrolling que nous avons utilisé associé à une webcam fixe nous a empêchés de mettre en place une projection du décor du jeu sur les cubes afin de pouvoir savoir en temps réel où était le personnage. Mais à l'aide du bras mécanique il serait possible de déplacer la webcam en fonction du déplacement du personnage, même si cela signifierait de devoir changer de cube de référence en fonction de la position du personnage. Cela

améliorerait le côté ludique du jeu mais également le confort de l'utilisateur.

Une autre amélioration sur laquelle nous travaillions dans les dernières semaines de notre projet est l'intégration d'ennemis dans le jeu et la possibilité d'en rajouter à l'aide de cubes spéciaux.

Conclusion

Ce projet nous a permis d'apprendre premièrement à programmer une application graphique en C, ce qui était une première fois, ce qui s'avéra être fort intéressant. Il nous a permis aussi d'apprendre à nous renseigner sur des technologies existantes, de peser le pour ou contre des différentes technologies qui s'offraient à nous, et d'en choisir une. Ceci étant très probablement un des travails qui nous sera demandé lorsque nous serons ingénieur.

Il nous a permis enfin d'améliorer notre compétence en travail d'équipe et en programmation.

Un des points fort de notre projet, est qu'on aurait pu lui donner une dimension commerciale. On trouve dans n'importe quel supermarché des petites figurines en plastique qui interagissent avec des jeux vidéo (ex : Amiibo), et pourtant ce sont de simple figurines, alors que notre projet permet de créer tout l'univers en entier. On combine à la fois l'esprit du jeu Mario mais aussi l'esprit de briques lego qu'on assemble pour former notre propre univers. Tout ceci donnant à notre projet un coté très attractif, et donc qui pourrait plaire à de nombreuses personnes.

Réaliser ce projet nous a tout autant fortement plu, et nous aura donc apporté compétences et divertissement.