Université de Limoges

DELOUVEE Nathan
MALLET Nathanael

RAPPORT DU PROJET CONCEPTION

Université de Limoges

Sommaire

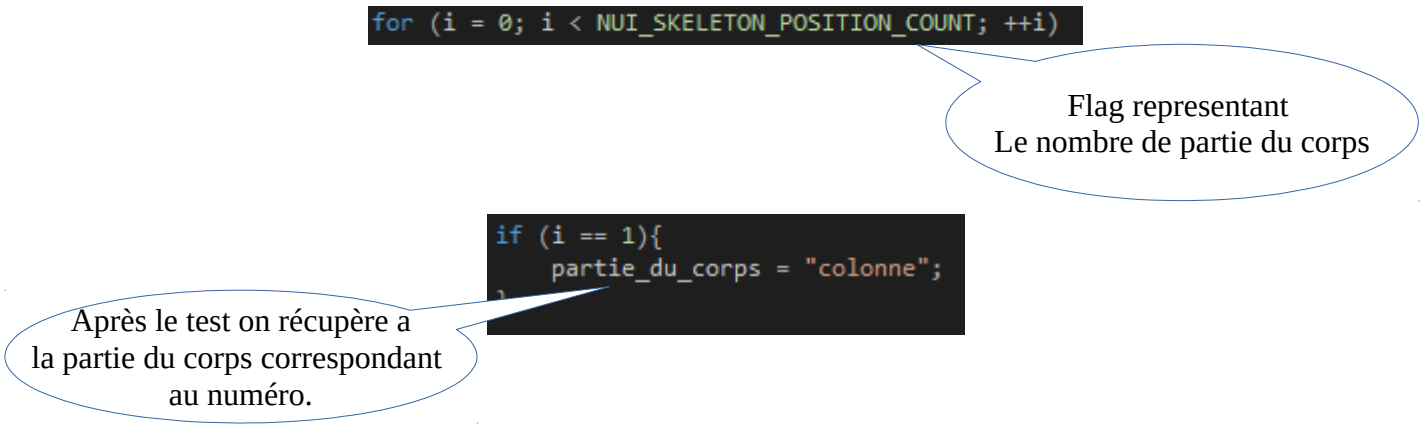
Introduction

Ayant deux projets professionnel distincts nous avons demandés à des professeurs s'il existait un projet regroupant image et sécurité. On nous a donc proposé de travailler sur un système d'authentification par mouvement à l'aide d'une kinect.

Pour ce faire nous avons fait des recherches de façon à nous équiper des logiciels appropriés et à comprendre le fonctionnement de ces outils comme le logiciel SDK (Software Development Kit) qui permet de récupérer les codes des applications liées à la kinect. Nous nous sommes concentré sur l'application permettant de détecter et dessiner le squelette. Nous allons voir dans un premier temps la récupération et trie de données brut. Puis nous expliquerons le traitement de données. Au final, nous allons parler de l'implémentation de l'interaction avec l'utilisateur.

I°) Récupération et trie donnée brut

Tout d'abord, le premier problème à régler est de comprendre le code qui contient des éléments nouveaux tels que la manipulation de fenêtres Windows. Nous devons ainsi identifier la méthode « DrawSkeleton » qui dessine le squelette en parcourant chacun de ses points et récupère les coordonnées de ce dernier affichées sur l'écran. En se basant sur les ressources documentaires de Microsoft et les commentaires dans le code nous pouvons voir que les points sont identifiés clairement par des numéros allant de 0 à 19 (« flag »). Pour des raisons de facilité nous avons choisi de stocker la partie du corps correspondante ainsi que ses coordonnées dans le fichier texte dédié à la tâche qui est exécutée (tentative d'authentification ou enregistrement de mot de passe). Nous optons pour l'utilisation de « ofstream » en écrivant à la suite du fichier grâce aux options « ios::out » (permettant l'écriture) et « ios::app » (permettant d'écrire à la suite).



II°)Traitement de données

Un deuxième problème se pose, le flux de données énorme écrit dans le fichier texte qui contient parfois des valeurs identiques. La solution évidente est la compression. Elle consiste à supprimer les doublons ou même les valeurs très proche (dans un écart donné qui sera identifié comme « epsilon ») de façon à réduire ce flux sans modifier la prise du mot de passe ou l'identification. Pour cela, à chaque coordonnée récupérée, nous la comparons à la précédente +/- « epsilon ». Si la coordonnée est en dehors de l'intervalle de la valeur précédente avec l'epsilon alors

nous l'écrivons dans le fichier. Nous créons donc la fonction « compressions() » qui prend en paramètres les deux coordonnées précédentes et les deux coordonnées actuelles dont chacune est un entier représentant le x et le y.

Entête de la fonction compression.

```
bool compression(int x_prec, int y_prec, int point_actuel_x, int point_actuel_y){
```

```
if (!compression(x_precedent, y_precedent, m_Points[i].x, m_Points[i].y)){
    fichier << "position " << partie_du_corps;
    cmp++;
    fichier << "(" << m_Points[i].x << "," << m_Points[i].y << " )" << endl;
}
```

Utilisation de la fonction et écriture dans le fichier.

Nous arrivons au problème principal la comparaison de deux mouvements.
Le principe est simple, nous demandons à l'utilisateur de faire un mouvement. Puis nous le récupérons et traitons. Par la suite nous comparons le mouvement qui sert de mot de passe et celui durant la tentative d'authentification en se basant sur leurs coordonnées.
Avec du bon sens, nous remarquons que l'authentification est plus focalisée sur le haut du corps, le bas nous est donc indifférent. Ce qui nous facilite la comparaison.
Grâce au fichier texte du mot de passe, nous stockons les points dans un conteneur.
Pour cela nous nous sommes concentré principalement sur les mouvements des mains ainsi nous récupérons les coordonnées de ces dernières, et les stockons dans une collection de la classe « point » (la classe point est une classe qui a pour attribut deux double avec un constructeur).

```
pch = strtok(tmp, " :,()");
```

```
if (i == 3 && part.compare("mainGauche")){
    y=stod(pch);
    Point p = Point(x, y);
    CoordMainG.push_back(p);
}
```

On utilise strtok pour garder seulement les informations importantes et les ajouter à la collection

A ce moment, nous pouvons constater que lors de la tentative d'authentification, l'utilisateur peut se placer à droite ou à gauche de l'écran. Même en ayant effectué un mouvement identique, notre méthode de comparaison utilisant les coordonnées, nous indique que les mouvements sont différents. Nous choisissons donc d'utiliser plutôt des vecteurs et nous les comparons, ce qui nous

permet de résoudre le problème de la position de l'utilisateur.

Nous utilisons les deux collections de points appartenant aux mains , nous parcourons chaque collection, pour laquelle nous calculons la distance de deux points successifs en soustrayant leurs coordonnées. Puis nous les stockons dans une autre collection. Dans notre programme, les collections Dpd et DpG pour les vecteurs de la main droite et gauche qui provienne du fichier de mot de passe et la collection TestDpd et TestDpG qui proviennent du fichier de la tentative de connexion.

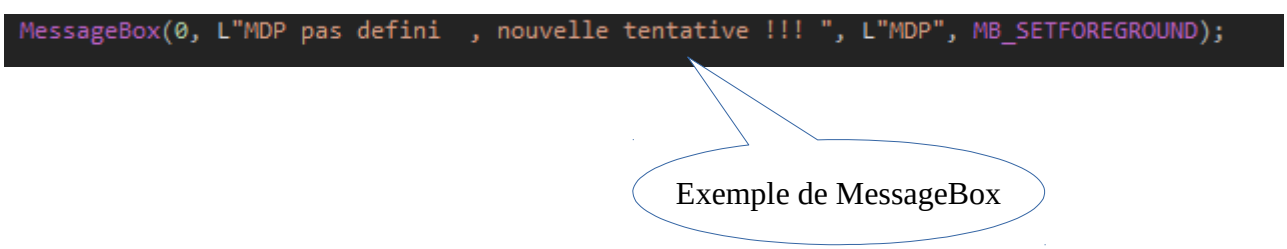
Nous parcourons d'abord la collection de TestDpD ou TestDpG puis nous comparons chaque vecteur avec ceux de DpD ou DpG respectivement.
A chaque test réussi, nous incrémentons un compteur, lorsque nous trouvons successivement un nombre de vecteur supérieur ou égale au nombre vecteur du mot de passe, nous mettons une variable booléenne a true.

Si la variable booléenne est à true nous validons la comparaison pour la main testée.
D'autre part, la duplication d'un mouvement n'étant pas forcément précise, nous devons introduire un « epsilon » pour cette phase comme vu précédemment de façon à accorder à l'utilisateur une marge d'erreur. Bien entendu, cette méthode introduit le problème du faux positif. Un personne faisant approximativement le geste enregistré comme mot de passe d'un autre utilisateur pourrait s'authentifier s'il reste dans la marge donnée par l'epsilon.
De plus, la Kinect peut avoir des problèmes à dessiner le squelette dépendamment de l'environnement. En effet, nous remarquons que lorsque nous passons une main devant le corps, il y a une probabilité que le Kinect ne trouve plus la main.

III°)Interface

L'ajout de la console est seulement pour nous aider à voir les problèmes et appliquer les solutions. Elle permet aussi d'afficher les modifications faites au fichier comme la compression et l'ajout de coordonnées. Comme le code est en VC++, cela nous permet de mieux comprendre l'environnement en nous rapprochant de ce que nous connaissons.

Nous implémentons aussi des MessageBox de façon à faciliter les interactions avec l'utilisateur. Une MessageBox n'est rien d'autre qu'une fenêtre avec un ou plusieurs boutons. Nous en choisissons une déjà existante avec trois boutons « Oui », « Non » et « Annuler » mais il est tout à fait possible de créer notre propre MessageBox. L'utilisation de cette dernière se fait, dans le code, pour demander, au lancement de l'application, si l'utilisateur veut s'authentifier ou créer un mot de passe. Nous en utilisons aussi pour dire à l'utilisateur si son authentification a réussi ou non et si une authentification est demandée sans l'existence de mot de passe.



Exemple de MessageBox

Conclusion

Nous avons pu découvrir le codage en application et avons été face à de nouveaux problèmes que nous avons pu résoudre avec l'aide notamment de nos encadrants. Le sujet était intéressant, et l'implémenter a été une très bonne expérience.
Pour une future amélioration, nous pourrions implémenter une version sécurisée en modifiant la méthode de stockage des données. Une implémentation de la comparaison de plus de points comme ceux des bras et de la tête pourrait aussi être possible.

Ressources

- SDK 1.7/SDK 1.8
- Microsoft Visual Studio 2013
- SkeletonBasics-D2D
- msdn.microsoft.com

Guide Vidéo

https://www.youtube.com/watch?v=pADOOggYO_w

https://www.youtube.com/watch?v=-fUhyb7JFYk