TP Kinectv1 sous Unity 3d
JIN 2018-2019

Frédéric Davesne

Objectifs:

- Savoir récupérer les données de la Kinect sous Unity 3d et la piloter
  - à partir d'une librairie C# sous Unity3d interfaçant le SDK Kinect Microsoft;
  - comme client VRPN (Virtual Reality Peripheral Network) de l'application FAAST[1].
- Qualifier les données récupérées.
- Animer un ou des objets graphiques Unity à partir des données de la Kinect
  - à partir des positions des différents éléments du squelette (vecteurs 3D) ;
  - à partir des rotations d'un élément du squelette par rapport à son "père" dans le graphe de scène (quaternions -> vecteurs 4D) .
- Coder sous Unity les techniques d'I3D suivantes indépendamment du périphérique de RV:
  - Main Virtuelle Simple ;
  - RayCasting ;
  - Direction du regard.
- Utiliser un point d'entrée matériel unique ayant comme sortie un vecteur3d qui peut être injecté à chacune des trois techniques d'I3D.

---

0. Installation des composantes

1. Introduction (voir le fichier Documentation.pdf)

1.1. Essai de la Kinect en dehors de Unity

a. Lancer l'application FAAST (Flexible Action and Articulated Skeleton Toolkit) avec *Display->Background Pixels*: RGB, puis Connect.

b. *(Uniquement pour la Kinect1)* Piloter le moteur en Pan de la Kinect en allant dans *Sensor->Set Pitch* après avoir modifier la valeur de *Kinect Motor Control*.

c. Placez-vous devant la Kinect. Lorsque votre corps (ou une partie) est reconnu, celui-ci est tracké et un squelette correspondant à différentes articulations du corps apparait en surimpression de l'image RGB/DEPTH.

d. Cliquer sur *Server* et remarquer l'appariement d'un 'squelette' avec l'image RGB/DEPTH lorsque Tracker0 est associé à un identifiant.

1.2. Production d'un événement à partir d'un geste capté par FAAST puis utilisé par Unity

FAAST permet de générer des événements simples (clavier/souris) à partir de la détection de positions/vitesses/mouvements du squelette lorsque la personne est trackée par la kinect.

a. Emuler la touche 'a' du clavier

[1] Voir http://projects.ict.usc.edu/mxr/faast/

- Aller dans *Gestures -> New Gesture -> Add ->Mode Trigger Once -> Add position constraint -> right hand ->To the right of -> Torso -> 30 cm at least et associer ce geste avec l'appuie de la touche 'a'*
- Sauver le mouvement et *Start Emulator*
- Ouvrir une console de commande et essayer lorsque cette console est active ...
- Faire la même chose avec *Gestures -> New Gesture -> Add ->Mode Loop repeatedly -> Add Velocity Constraint -> right hand ->to the right -> at least 10 cm/sec*

b. Créer un projet Unity *Essai Kinect*
- Créer un répertoire *Script/Kinect/faast* dans lequel vous allez créer un fichier C# *MoveFromKeyboard.cs*.
c. Créer un *GameObject* Cube

d. Dans le script *MoveFromKeyboard.cs*, récupérer la touche clavier 'a' et déplacer le cube de 0.1 m sur l'axe X.

e. Constatez que le cube se translate lorsque vous effectuez le mouvement adéquat.

D'après vous, quelles sont les limitations de cette approche?

2. Utilisation d'un wrapper Kinect.
a. Décompresser le package *Premier Essai.unitypackage*

b. Regarder la scène *Première Scène* et cliquer sur *KinectPrefab*. Celui-ci fait l'interface entre les GameObjects de la scène courante et les données provenant de la Kinect car il contient tous les scripts associés à l'interface Kinect.

c. Créer un EmptyObject *body* puis créer un GameObject *main*_droite de type Sphère de rayon 0.1 mètre, dépendant hiérarchiquement de *body.*

d. Associer le déplacement de la sphère de la scène avec le déplacement de la main droite. Pour cela, utiliser la classe *KinectPointController* et choisissez correctement les paramètres d'entrée de cette classe C#.

e. Créer une classe C# *TraceData.cs* qui trace les données en position de la sphère en fonction du temps et qui les place dans un fichier texte *Trace.txt*. Chaque ligne sera de la forme 'temps' 'x' 'y' 'z'. Tracez la position de votre main droite et regarder le fichier créé. Que constatez-vous?

f. Grâce à cette trace de données, essayez de calculer la fréquence des données ainsi qu'une précision approximative de celles-ci. Pour cela, vous pouvez par exemple utiliser Excel sur la première colonne du fichier *Trace.txt*.

g. Modifier *TraceData.cs* de manière à ce que la trace ne s'exécute que si une personne est trackée par la Kinect. Pour cela, modifier la classe *KinectPointController* de manière à affecter un booléen *isTracked* si la main droite est trackée (voir le code de la fonction *Update()* de *KinectPointController* pour cela).

h. Créer une nouvelle scène *Squelette animé* reprenant le prefab *KinectPrefab* et incluant dans l'EmptyObject *body* tous les GameObjects de type Sphère et de rayon 0.1 mètre associés à

toutes les parties du squelette identifié par la Kinect (voir les points d'entrée de la classe *KinectPointController*).

3. Utilisation du package Kinect1.7Wrapper.unitypackage

a. Décompresser le package Kinect1.7Wrapper.unitypackage et charger la scène KinectSample.unity .

b. Regarder la hiérarchie du GameObject *rainbowMan_v6* ainsi que la classe associée à cet objet. Comment les différentes parties du squelette peuvent-elles se déplacer entre-elles? Observer en particulier dans la partie *Scene* de *Unity* le déplacement en position et rotation de différentes parties du squelette.

c. Ecrire un script *TraceDataRot.cs* qui trace les données de rotation du GameObject *right_shoulder* hérité de *rainbowMan_v6* en vous inspirant du *TraceData.*cs du 2. .Faites de même avec les données de rotation du GameObject *neck*. Qu'en déduisez-vous de la possibilité d'implémenter la technique d'I3D *Direction du regard* ?

d. Ecrire un script *avance.cs* attaché au GameObject *root_joint* permettant de faire avancer *rainbowMan_v6* dans la scène.

4. Implémentation de la technique Main Virtuelle Simple .

a. Ecrire un script *mvs.cs* ayant en entrée un *Vector3* issu de la position de la main droite captée par la Kinect, une matrice de passage Réel/Virtuel et en sortie un *Vector3* représentant la position de l'avatar de la main droite *main*_droite dans le monde virtuelle.

b. Créer un cube dans l'environnement virtuel, que vous nommerez *cible* .

c. Créer un script *EstSelectionnable.*cs ayant comme paramètre d'entrée le GameObject *main*_droite et traduisant le fait que *cible* est sélectionnable par *main_droite*. Pour cela, deux possibilités:
- La distance entre *main_droite* et *cible* est inférieure à un réel positif donné
- *main_droite* et *cible* s'intersectent. Pour cela, ajouter la propriété *RigidBody* à *main_droite* et *cible* en otant pour le moment la gravité à chacun des deux GameObjects. Cocher *Is Trigger* dans le *Box Collider* associé à *cible*. L'événement d'entrée en collision se gère grâce à la fonction `void OnTriggerEnter (Collider other).`

d. Dans le cas où *cible* est sélectionnable, changer sa couleur dans le même script *EstSelectionnable.cs* .

e. Gérer le retour de *cible* à *non sélectionnable* grâce à la fonction `void OnTriggerExit (Collider other).`

f. Trouver un moyen (i.e. un *Contrôle d'Application*) pour que l'objet *cible* passe de l'état *sélectionnable* à l'état *sélectionné*. Lorsque l'objet *cible* est à l'état *sélectionné*, il devient graphiquement (hiérarchie) fils de *main_droite*.

5. Implémentation de la technique Ray Casting .

a. Utiliser *rainbowMan_v6* pour implémenter la technique du RayCasting .Pour cela, écrire un script *raycasting.cs* ayant comme entrée deux angles issus de l'orientation du bras droit, une matrice de passage Réel/Virtuel et en sortie un Vector3 représentant l'orientation de l'avatar de la main droite dans le monde virtuel.

b. Adapter le cheminement du 4. pour achever l'implémentation de la technique du RayCasting.

FIN