

# Apprendre à développer pour Kinect V2

En créant 3 projets en C#



Rémi Jacquemard – Fin2

ETML – Lausanne

2015

Tutoriel créé dans le cadre des TPI

Supervision : Patrick Chenaux, Deniz Mutlu et Serge Wenger



## Table des matières

<b>1</b>	<b>Introduction .....</b>	<b>3</b>
1.1	Le rôle de ce tutoriel .....	3
1.2	Kinect .....	3
1.3	A qui se destine ce tutoriel ? .....	5
1.4	Marche à suivre .....	6
<b>2</b>	<b>Mise en place du poste de travail .....</b>	<b>7</b>
2.1	Composants et logiciels nécessaires .....	7
2.2	Installation .....	7
	Installation du SDK .....	7
	Connecter Kinect au PC .....	8
	Vérifier le bon fonctionnement de Kinect .....	9
<b>3</b>	<b>Découverte de Kinect et son API .....</b>	<b>11</b>
3.1	Les <i>data sources</i> .....	12
	AudioSource .....	12
	ColorFrameSource .....	13
	BodyFrameSource .....	13
	BodyIndexFrameSource .....	14
	DepthFrameSource .....	15
	LongExposureInfraredFrameSource et InfraredFrameSource .....	16
3.2	Les <i>frames</i> .....	16
<b>4</b>	<b>Premier projet : Kinect Draw .....</b>	<b>18</b>
4.1	Description du projet .....	18
4.2	Mise en place et prérequis .....	19
4.3	Structure et design du projet .....	19
	Les couches .....	19
4.4	Petite introduction à la structure de ce projet Windows 8.1 .....	21
4.5	La classe <i>KinectCoreWindow</i> .....	22
	L'emploi de cette classe .....	23
4.6	Développement et conseils .....	25
	Commencer le projet .....	25

Le dessin .....	27
<b>4.7 Correction .....</b>	<b>28</b>
<b>5 Deuxième projet : Kinect PowerPoint.....</b>	<b>29</b>
5.1 Description du projet .....	29
5.2 Mise en place et prérequis .....	30
5.3 Structure et principe du projet.....	30
5.4 La détection de mouvement avec <i>Visual Gesture Builder</i> .....	32
Kinect Studio .....	32
Le logiciel <i>Visual Gesture Builder</i> et les technologies de détection.....	33
L'implémentation à l'aide de <i>VisualGestureBuilderFrameSource</i> .....	35
5.5 Principe de la détection de mouvement dans le projet.....	38
Récupérer la liste des <i>body</i> présents.....	38
La classe <i>GestureDetector</i> fournie .....	39
5.6 Introduction au développement de cet add-in PowerPoint .....	41
Le déploiement d'add-in Office .....	42
5.7 Développement et conseils .....	43
Commencer le projet.....	43
Récupérer les frames des <i>datasources</i> .....	43
Implémenter la détection de mouvement .....	45
Détecter les mouvements .....	46
5.8 Correction .....	46
5.9 Le concept de l'ajout du pointeur.....	46
<b>6 Pour aller plus loin... ..</b>	<b>48</b>
6.1 Ajouter des fonctionnalités aux projets .....	48
<b>7 Sources .....</b>	<b>50</b>

# 1 Introduction

## 1.1 Le rôle de ce tutoriel

Ce tutoriel a pour but d'initier au développement pour Kinect V2, ou Kinect pour Xbox One.

Une description générale de l'API<sup>1</sup> fournie par Microsoft sera effectuée. Le matériel et les logiciels nécessaires seront décrits, ainsi que leur mise en place.

Deux projets permettront de se familiariser avec le développement pour Kinect :

- Kinect Draw est une petite application qui permet de dessiner à l'écran sans toucher physiquement aucun périphérique, uniquement à l'aide de Kinect.
- Kinect PowerPoint est un plugin pour PowerPoint qui permet de changer de slide lors d'une présentation avec un geste de la main.

Ce tutoriel présentera aussi quelques liens pour aller plus loin dans le développement de Kinect, quelques projets possibles, etc.

## 1.2 Kinect

*Kinect* est un périphérique d'Interaction Homme-Machine (IHM) fabriqué par Microsoft. Il permet d'interagir avec une machine sans aucun contact physique, à contrario de la souris ou du clavier, par exemple. Sur ce point, la sortie de la première version de ce périphérique était assez révolutionnaire. En effet, pour un coût peu élevé, le grand public pouvait acquérir ce capteur à brancher à sa Xbox 360. Il est d'ailleurs le périphérique de jeu s'étant vendu le plus rapidement, avec en moyenne 133'333 unités vendues par jour à son lancement (Guinness World Records).



Figure 1.1 : « La manette, c'est vous ! » est le slogan de Microsoft pour présenter Kinect

<sup>1</sup> Application Programming Interface

La première version de *Kinect* est sortie en novembre 2010 un peu partout dans le monde. Initialement, elle était prévue pour être uniquement connectée à la *Xbox*. Mais très vite, le périphérique a intéressé les développeurs tiers, et des SDK<sup>1</sup> non officiels sont apparus, tel que *libfreenect*. En Juin 2011, *Microsoft* a sorti le premier SDK en même temps que *Kinect for Windows*, un capteur à connecter directement au PC en USB, apportant plus de fonctionnalité que la version *Xbox 360*.

La deuxième version de *Kinect* – celle utilisée dans ce tutoriel – est sortie en novembre 2013. Elle faisait partie intégrante de la *Xbox One* de *Microsoft*. Le seul moyen de la posséder était d'acheter la console. En Septembre 2014, il est devenu possible d'acheter le capteur uniquement.

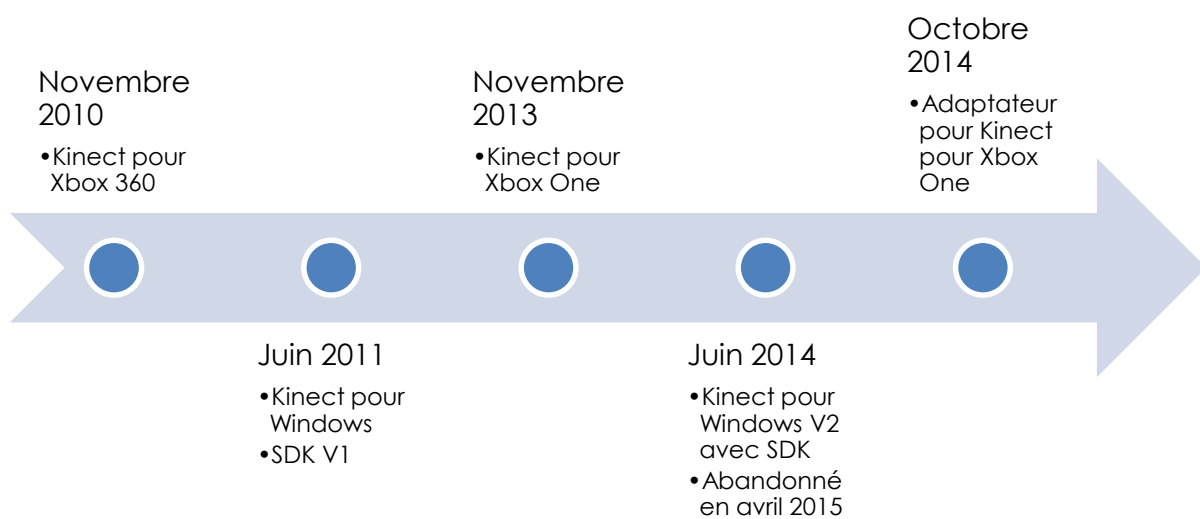


Figure 1.2 : L'histoire de Kinect

De la même manière que pour la version 1, une version spécifique pour Windows est sortie en juin 2014. Mais 4 mois plus tard, il est devenu possible d'acheter chez *Microsoft* un adaptateur pour brancher *Kinect* pour *Xbox One* directement sur le PC, si bien que la caméra *Kinect for Windows V2* n'a plus été produite à partir d'avril 2015.



Figure 1.3 : La première version de Kinect à gauche, la deuxième à droite

<sup>1</sup> Kit de développement, ou *Software Development Kit* en anglais

### 1.3 A qui se destine ce tutoriel ?

Ce tutoriel se destine originellement aux élèves de 3<sup>ème</sup> et 4<sup>ème</sup> années de la section informatique de l'ETML (Ecole Technique – Ecole des Métiers de Lausanne). Il s'appuiera sur les modules ICH-103 (Programmation séquentielle) et ICH-303 (Programmation événementielle) – où leurs équivalents ICT – déjà vus. Comme la programmation orientée objet ne fait pas partie du programme de l'ETML, des efforts ont été faits pour que le code évite l'utilisation de classe excessivement. Malgré tout, il serait tout de même souhaitable d'en avoir quelques notions (le principe de classe et l'utilisation de celle-ci). Toute personne connaissant la programmation séquentielle, événementielle et orientée objet en C# et disposant des codes de bases fournis devrait être capable de suivre ce tutoriel.

Les projets sont développés à l'aide de Visual Studio. Le code présenté suivra les dernières normes en vigueur de l'ETML (version 1.1.0). En voici certaines de ces spécificités :

- Chaque fichier créé contient un entête contenant l'auteur, la date, l'entreprise (ETML) et un bref résumé :  

```
///ETML  
///Author : Luke Skywalker  
///Date : 19.01.2014  
///Summary : Algorithm which transforms the dark force into light...
```
- Pour chaque méthode, l'utilisation des commentaires XML de Visual Studio est utilisée et les éventuelles sources sont indiquées
- Des commentaires de fin d'accolade sont utilisés seulement si ceux-ci améliorent la visibilité
- Les noms de variable utilisent le « camelCase », **ne** sont **pas** préfixées, mais restent explicites :  
Exemple : titleLabel (de type Label), name (de type string), isCursorMoving (de type booléen), etc.

Ce document est à la disposition des élèves de l'ETML sur le k:/.

**Information importante :** les normes de l'ETML indiquent que les commentaires doivent-être écrits en anglais. Cependant, pour que la compréhension de ce tutoriel soit meilleure, ceux-ci seront en français.

## 1.4 Marche à suivre

Une archive ou un dossier de base est fourni pour suivre ce tutoriel, ordinairement sur un CD. Celui-ci comprend principalement :

- Le dossier *ProjectsForTutorial* contenant les projets abordés dans ce tutoriel :
  - Un code de base pour chaque projet, qui permet à l'élève de suivre ce tutoriel
  - Un code final pour chaque projet, qui est une proposition de correction
- Le SDK 2.0 de Kinect (version 2.0.1410.19 en mai 2015)

Lorsqu'il sera demandé dans ce tutoriel de récupérer certains dossiers ou fichiers depuis l'archive fournie, c'est de celle-ci dont il est question.



Figure 1.4 : Le packaging du capteur Kinect, dans sa deuxième version



## 2 Mise en place du poste de travail

### 2.1 Composants et logiciels nécessaires

- Un PC
  - Un processeur 64-bit dual-Core cadencé à 3.1 GHz
  - Un port USB3 disponible
  - 4Go de RAM
  - Une carte graphique supportant DirectX11
  - Windows 8 minimum, 8.1 pour pouvoir suivre ce tutoriel
- Le capteur Kinect pour Xbox One
- L'adaptateur Kinect (V2) pour PC
- Une version de Microsoft Visual Studio 2013
- Pour le projet *Kinect PowerPoint* : Le logiciel Microsoft Office PowerPoint 2013

**Information importante** : il sera nécessaire d'avoir les droits administrateurs sur le PC pour installer tous les composants

### 2.2 Installation

Si ce n'est déjà fait, installer une version de Microsoft Visual Studio 2013. Aucune version spécifique n'est requise.

Les étapes suivantes doivent être effectuées dans l'ordre :

1. Installation du SDK
2. Connecter Kinect au PC
3. Vérifier l'installation à l'aide de *Kinect Configuration Verifier*

#### Installation du SDK

Récupérer le SDK 2.0 de Kinect for Windows, disponible dans l'archive fournie ou à l'adresse qui suit, puis l'installer :

<http://www.microsoft.com/en-us/download/details.aspx?id=44561>

A ce stade, le dossier *Kinect for Windows SDK V2* a été ajouté dans le menu démarrer :

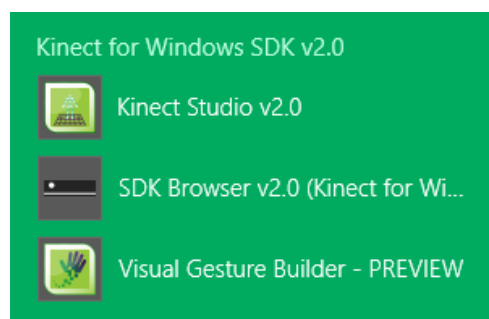


Figure 2.1 : Le dossier du SDK de Kinect

Voici ce qu'il contient :

- **Kinect Studio v2.0**  
Ce logiciel permet d'enregistrer un clip à partir de Kinect. Il sera utilisé et décrit plus loin dans ce tutoriel
- **SDK Browser v2.0**  
Ce petit programme permet d'installer des composants supplémentaires, des exemples de projets ou d'exécuter des utilitaires annexes
- **Visual Gesture Builder**  
Ce logiciel permet de créer des mouvements à partir d'un clip vidéo. Il sera utilisé et décrit plus loin dans ce tutoriel

### Connecter Kinect au PC

Connecter Kinect au PC tel qu'expliqué dans le guide d'installation rapide, qu'on peut trouver avec l'adaptateur Kinect pour Windows.

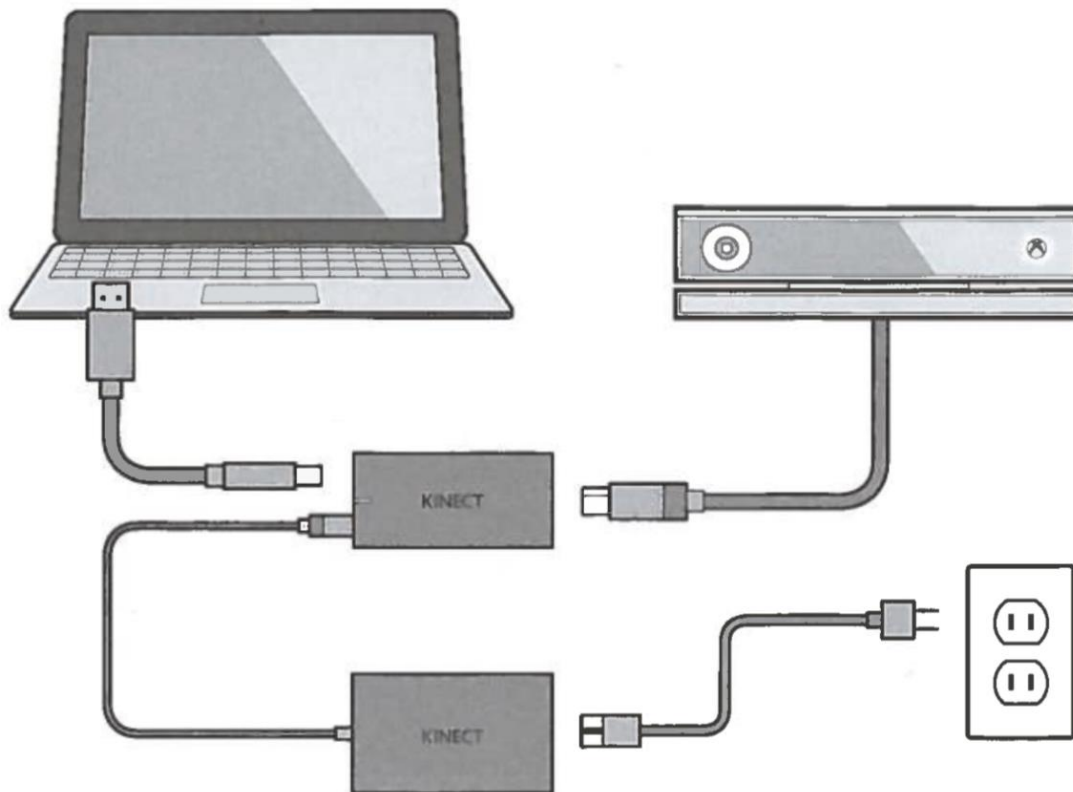


Figure 2.2 : Connexion de Kinect V2 au PC

Autoriser l'installation des drivers si demandé et attendre la fin de celle-ci.

## Vérifier le bon fonctionnement de Kinect

Lancer l'utilitaire *SDK Browser* depuis le dossier du SDK de Kinect V2. Exécuter ensuite *Kinect Configuration Verifier*:

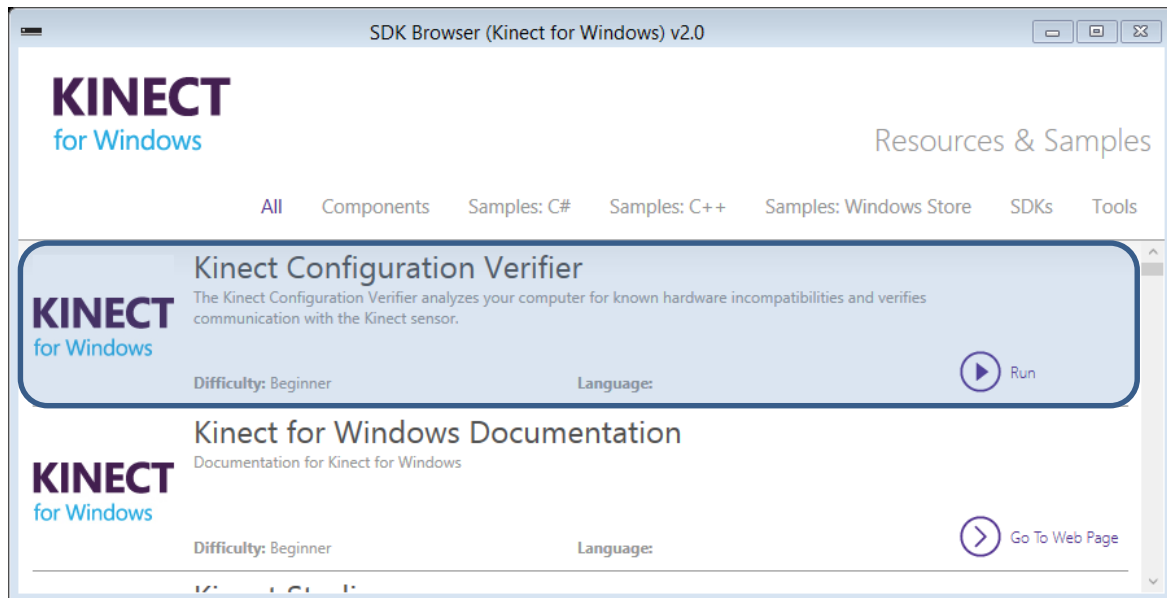


Figure 2.3 : Exécuter Kinect Configuration Verifier

Voici à quoi il ressemble :

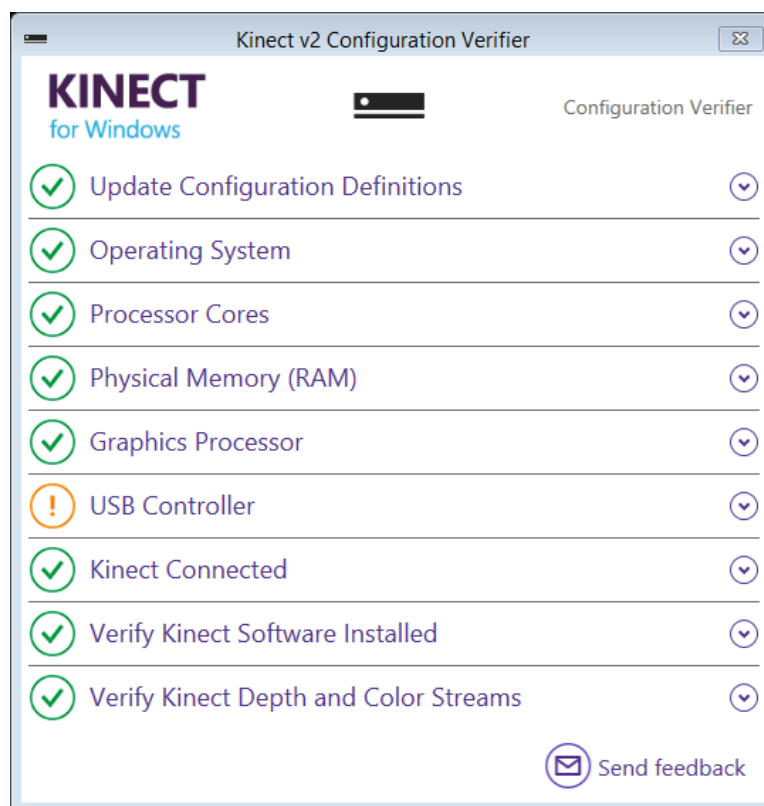




Figure 2.4 : Kinect v2 Configuration Verifier

On trouve plus d'information sur chacun de ces points en dépliant les sous-menus. Dans le cas présenté, on remarque un avertissement indiquant un problème potentiel de compatibilité du contrôleur USB.

 **USB Controller** 



Checks your USB controllers for Kinect for Windows v2 compatibility

Result: Supported USB 3.0 port detected with unknown bandwidth. Kinect may or may not be compatible with your hardware.

For more information, visit:  
[Kinect for Windows v2 System Requirements](#)  
[Kinect for Windows v2 Forums](#)

Figure 2.5 : Avertissement concernant le contrôleur USB

On nous informe qu'il se peut que les ports USB 3.0 ne soient pas compatibles. Mais en déployant le volet *Kinect Depth and Color Streams*, on remarque que tout semble bien fonctionner, 30 images par seconde étant capturées par Kinect.

 **Verify Kinect Depth and Color Streams** 

Detects depth and color stream latency

Result: Depth stream detected within target frame rate

For more information, visit:  
[Kinect for Windows v2 System Requirements](#)  
[Kinect for Windows v2 Forums](#)

FPS: 30



Figure 2.6 : Les images en couleur (à gauche) et infrarouge (à droite)

**Information** : il arrive que les FPS (frame per second, ou images par seconde) descendent sous la barre des 30 si, par exemple, la pièce n'est pas assez lumineuse. Cependant, cela ne devrait pas poser de problème.

### 3 Découverte de Kinect et son API<sup>1</sup>

Kinect possède 3 types de capteurs :

- Une caméra couleur Full HD
- Un capteur de profondeur, qui fonctionne grâce à une caméra et un projecteur de rayon infrarouge
- Une barre de microphone (4 microphones, permettant de suivre et de détecter la provenance des sons)

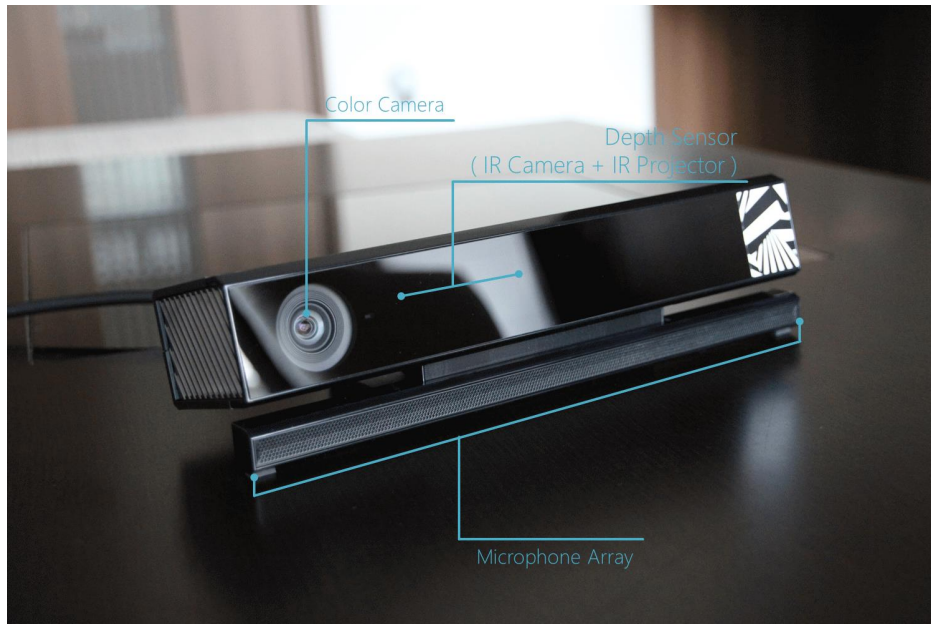


Figure 3.1 : Les capteurs de Kinect

De ces quelques capteurs, il est possible de détecter divers détails, comme les corps des joueurs présents. Kinect, via l'API, en fait des sources qui sont utilisables dans nos programmes. On les décrit dans le sous-chapitre 3.1.

Kinect V2 permet de détecter jusqu'à 6 personnes, et deux personnes peuvent être engagées en même temps.

L'engagement de joueur permet un suivi plus précis, une détection de la position des mains améliorée (on peut connaître notamment si une main est ouverte ou fermée), et permet d'éviter que plusieurs personnes interagissent avec le logiciel alors que ce n'est pas voulu.

---

<sup>1</sup> L'interface de programmation (*Application Programming Interface*) permet d'interagir avec Kinect à l'aide de classes et méthodes que Microsoft a créées.

### 3.1 Les data sources

Comme on l'a vu précédemment, l'API de Kinect nous fournit des sources qui nous permettent de récupérer des données à partir de la caméra Kinect. Elles sont au nombre de 7 :

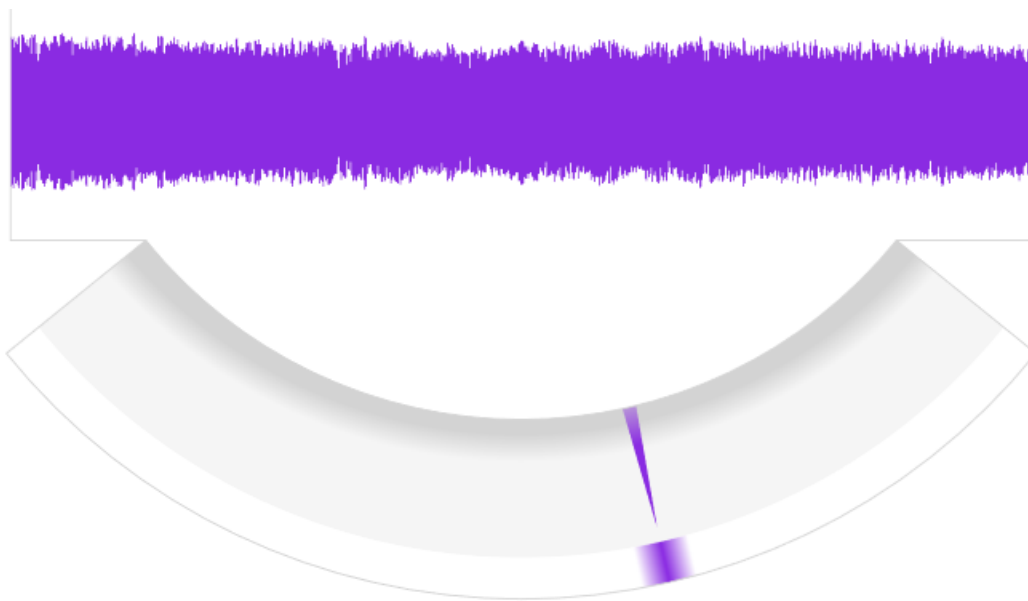
- *AudioSource*
- *ColorFrameSource*
- *BodyFrameSource*
- *BodyIndexFrameSource*
- *DepthFrameSource*
- *InfraredFrameSource*
- *LongExposureInfraredFrameSource*

Ces sources sont les fondations de l'API. On y trouve d'autres classes, mais celles-ci utilisent une, ou plusieurs, de ces sources.

Nous utiliserons quelques-unes de ces sources dans les projets que nous verrons plus loin.

#### **AudioSource**

Cette source permet de capturer l'audio de la pièce, de le suivre, de reconnaître certains mots, etc. Cette source ne sera pas utilisée dans ces projets, c'est pourquoi elle ne sera pas plus amplement définie.



*Figure 3.2 : AudioSource permet de capturer l'audio et de connaître sa provenance*



## ColorFrameSource

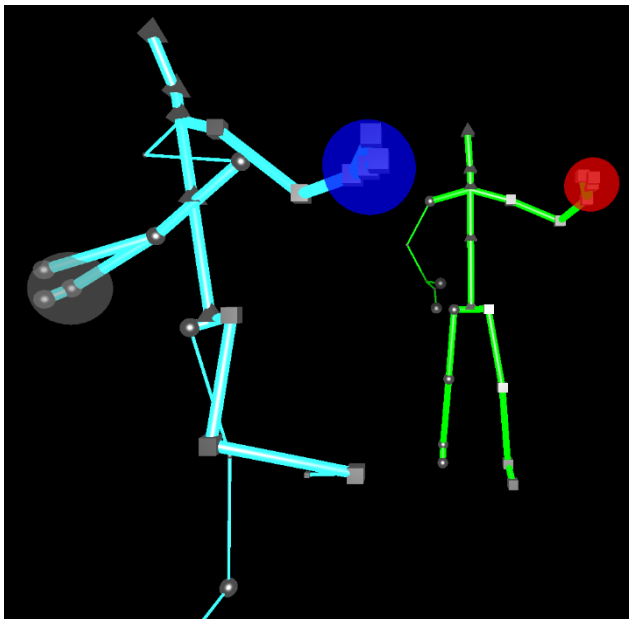
Cette source permet de récupérer l'image en couleur (1920x1080 pixels) fournie par Kinect grâce à sa caméra.



Figure 3.3 : Exemple d'image en couleur récupérable

## BodyFrameSource

Cette source permet de récupérer une liste de tous les *body*<sup>1</sup> visibles par Kinect. Pour chaque *body* détecté, on peut récupérer ses *jointures*, qui représentent le squelette, et leurs positions dans l'espace.



Il peut aussi être intéressant de savoir quel est l'état de la main d'un des joueurs (ouverte, fermée, ...), ce que permet cette source. Cependant, il est à noter qu'on ne peut détecter ces états que pour deux personnes au maximum.

Figure 3.4 : Les jointures récupérées grâce à BodyFrameSource

---

<sup>1</sup> Les corps, soit les joueurs qui sont visibles par Kinect. Seul le terme *body* sera utilisé par la suite.

### BodyIndexFrameSource

Cette source permet de récupérer une image en 2D (512x424 pixels) indiquant où se situent les corps des joueurs.



*Figure 3.5 : Les body vus par BodyIndexFrameSource*

Chaque pixel de l'image récupérée a une certaine valeur, indiquant quel joueur y est situé. Le noir est utilisé pour indiquer qu'aucun *body* n'est présent à cet endroit. Cette source peut être utilisée par exemple pour rapidement changer l'arrière-plan de l'image en couleur (image de Paris, etc.).



## DepthFrameSource

Cette source permet de récupérer une image 2D (512x424 pixels), dont la valeur de chaque pixel dépend de la distance de celui-ci :



Figure 3.6 : Le rouge correspond à ce qui est près, le violet à ce qui est loin

On voit bien sur l'image ci-dessus qu'il est facile d'obtenir la distance à laquelle chaque pixel correspond, notamment par le dégradé de couleur sur le sol, qui passe du rouge au vert.

En combinant *DepthFrameSource* et l'image en couleur, il est possible d'obtenir de bons rendus 3D :

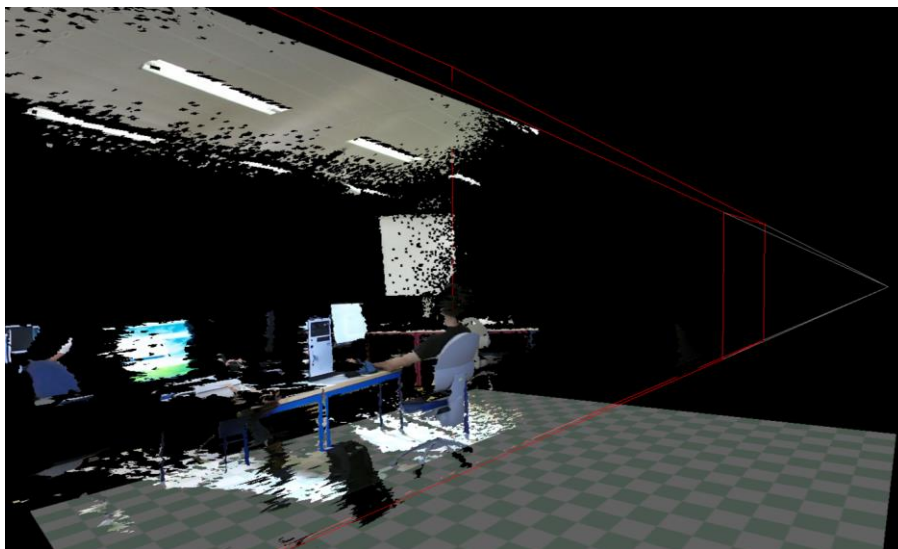


Figure 3.7 : Kinect est symbolisé par la pyramide à droite, les droites rouges en sont son champ de vision

### LongExposureInfraredFrameSource et InfraredFrameSource

Ces deux sources permettent de récupérer une image 2D (512x424 pixels) obtenue via le capteur infrarouge. On peut donc récupérer une image où la luminosité ne pose pas de problème, et même voir dans la nuit.



Figure 3.8 : Une image infrarouge

Ces sources sont utilisées par Kinect pour pouvoir notamment détecter les joueurs en condition de faible luminosité et contourner certains problèmes de reflet (soleil qui se reflète dans un verre, etc.)

*LongExposureInfraredFrameSource* permet d'avoir une image plus exposée que *InfraredFrameSource*, ce qui augmente sa qualité.

### 3.2 Les frames

La plupart des sources vues au chapitre 3.1 ci-dessus permettent de récupérer une image environ 30 fois par seconde. Ces images sont appelées *frames* dans l'API de Kinect.

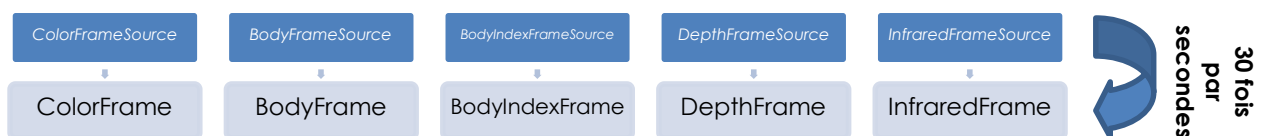


Figure 3.9 : Les frames disponibles

Une *frame* décrit une situation à un moment précis. C'est un instantané de ce que Kinect perçoit. Cela peut être une photographie, aussi bien que des données utiles au développement, tirées des sources. Par exemple, on peut connaître la position du pied d'un joueur dans l'espace à cet instant précis, ou encore, savoir si sa main est fermée ou ouverte.

Voici l'exemple qui permettra de mieux comprendre le système des *frames* :

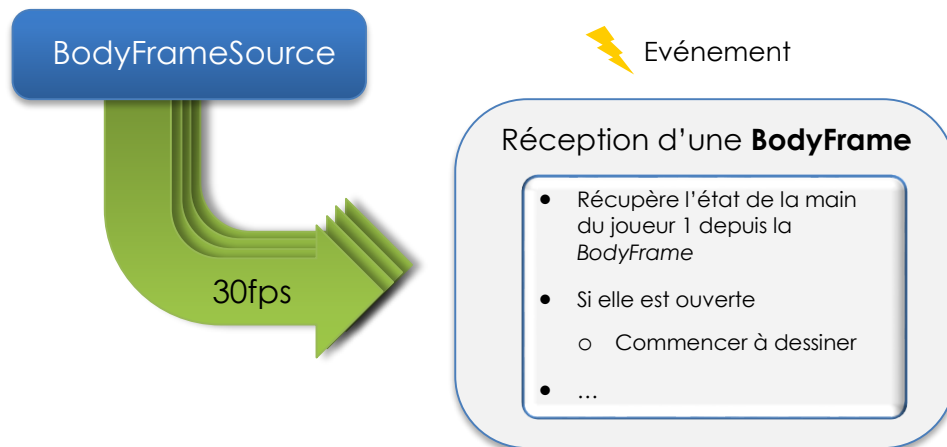


Figure 3.10 : Un exemple de l'utilisation des frames

L'exemple ci-dessus est basé sur la source *BodyFrameSource* qui, comme on l'a vu dans le chapitre 3.1, en page 13, permet de détecter la présence de joueurs et de récupérer l'état de la main.

Très vulgairement, *BodyFrameSource* déclenche toutes les 30 secondes environ un évènement, et exécute une méthode qu'on a définie. Cette méthode possède comme paramètre une *BodyFrame*, qu'on peut utiliser. Cette *BodyFrame* contient<sup>1</sup> un tableau de *body*, dans lequel, par exemple, l'état des mains de chaque joueur peut être récupéré.



Figure 3.11 : Récupération de l'état de la main droite à partir d'une *BodyFrame*

<sup>1</sup> Ce n'est pas exactement ainsi que cela se passe, comme on pourra le voir dans le *Deuxième projet : Kinect PowerPoint*

## 4 Premier projet : Kinect Draw

### 4.1 Description du projet

Ce projet permettra de dessiner de manière simplifiée à l'écran à l'aide de Kinect. La main droite est utilisée comme curseur et pinceau. Il a été choisi que le projet sera de type *application Windows Store* (application Windows 8). Ce type d'application est idéal pour Kinect, car le visuel occupe tout l'écran.

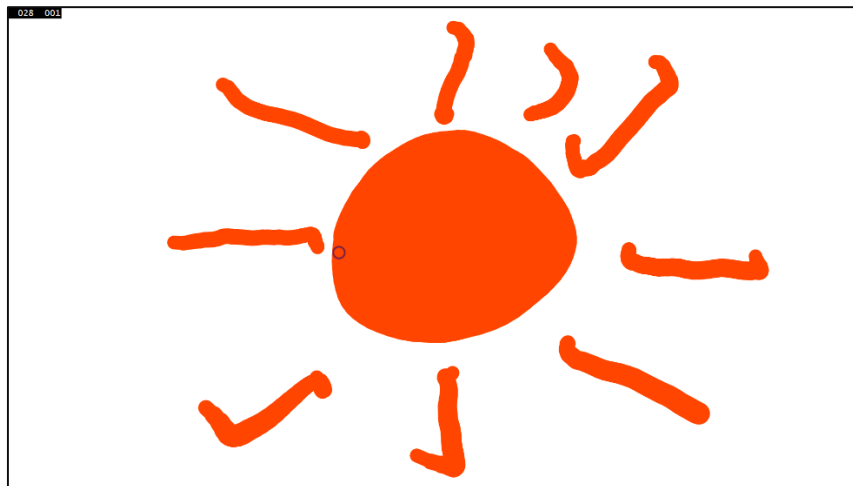


Figure 4.1 : Capture d'écran de l'application qui sera développée

La Figure 4.1 ci-dessus est une capture d'écran de l'application Kinect Draw. On peut distinguer le curseur, de forme circulaire, près du centre de l'écran.

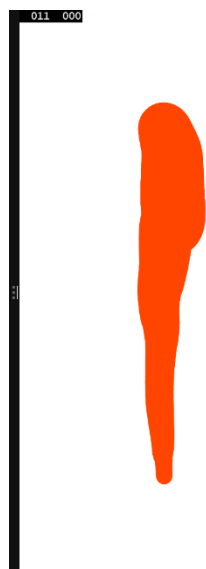


Figure 4.2 : On peut dessiner avec un pinceau plus gros en appuyant vers l'avant avec la main

Plus la main sera éloignée du corps du joueur, plus le pinceau sera gros, comme on le remarque sur la Figure 4.2. Intuitivement, il faut s'imaginer que lorsqu'on « appuie » plus fort, le pinceau « s'écrase » sur la feuille.

Dans ce projet, il ne sera pas décrit comment implémenter des fonctionnalités telles que l'effacement, ou encore le choix de couleur, car cela n'apporte aucune plus-value à l'apprentissage du développement sur Kinect. Cependant, il est assez facile de les implémenter.

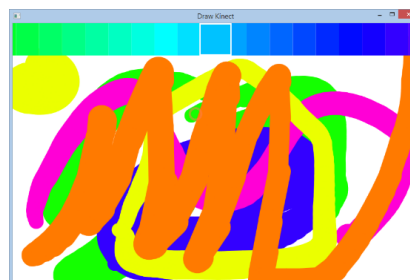


Figure 4.3 : exemple de fonctionnalité à ajouter, tel que le choix des couleurs

## 4.2 Mise en place et prérequis

*Information : ce projet a été ciblé pour les applications Windows Store 8.1. C'est pourquoi elle ne s'exécutera pas sur Windows 8.*

Depuis l'archive fournie, ouvrir le projet *Kinect Draw* (dans le dossier *KinectDraw-Base*) dans Visual Studio.

Ce projet de base contient le design de l'application et le code nécessaire pour qu'elle fonctionne. Elle n'implémente aucune fonctionnalité de Kinect. Cependant, le nécessaire a été fait pour que le développement sur Kinect puisse commencer de suite :

- La référence à l'API de Kinect a été ajoutée :
  - WindowsPreview.Kinect
- Les capacités de l'application Windows Store pour l'accès à Kinect ont été ajoutées :
  - Microphone
  - Webcam
- L'application a été ciblée pour les plateformes 64 bits  
Ceci est nécessaire pour les applications utilisant Kinect

## 4.3 Structure et design du projet

### Les couches

Lancer l'application

Voici son design :

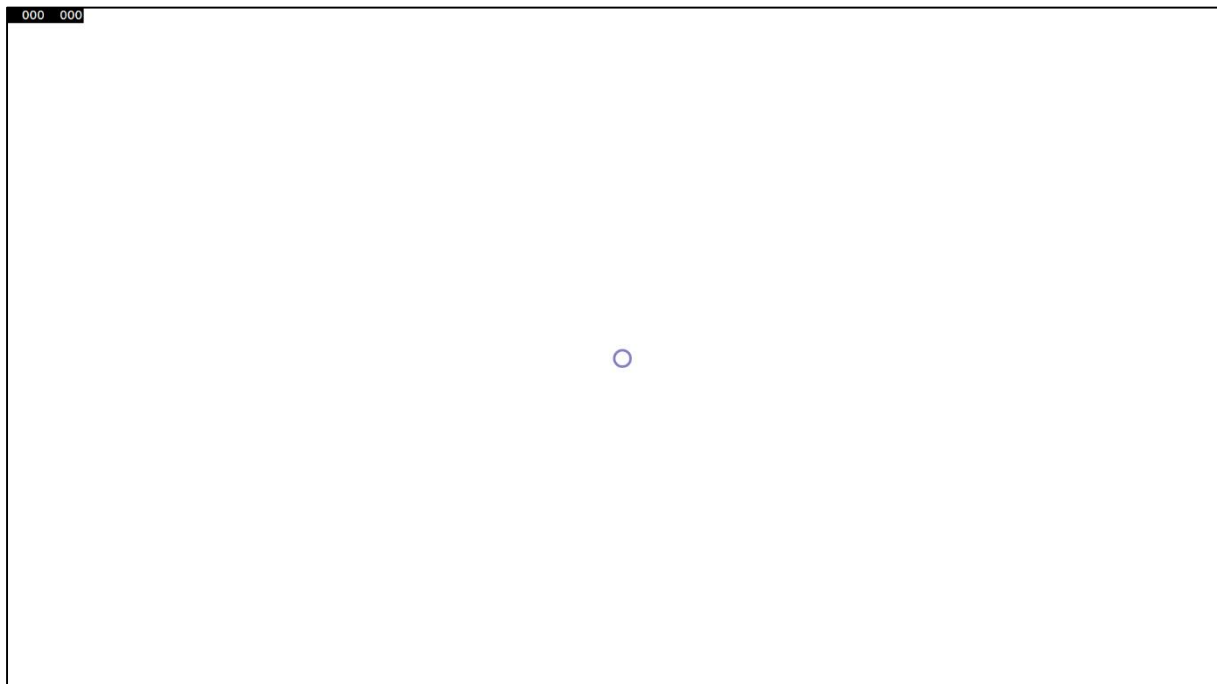


Figure 4.4 : L'application de base fournie

Cette application n'est pas utilisable. Cependant, le design a déjà été préparé, ainsi que la structure du projet, et quelques variables.

Voici comment le design de l'application a été arrangé :

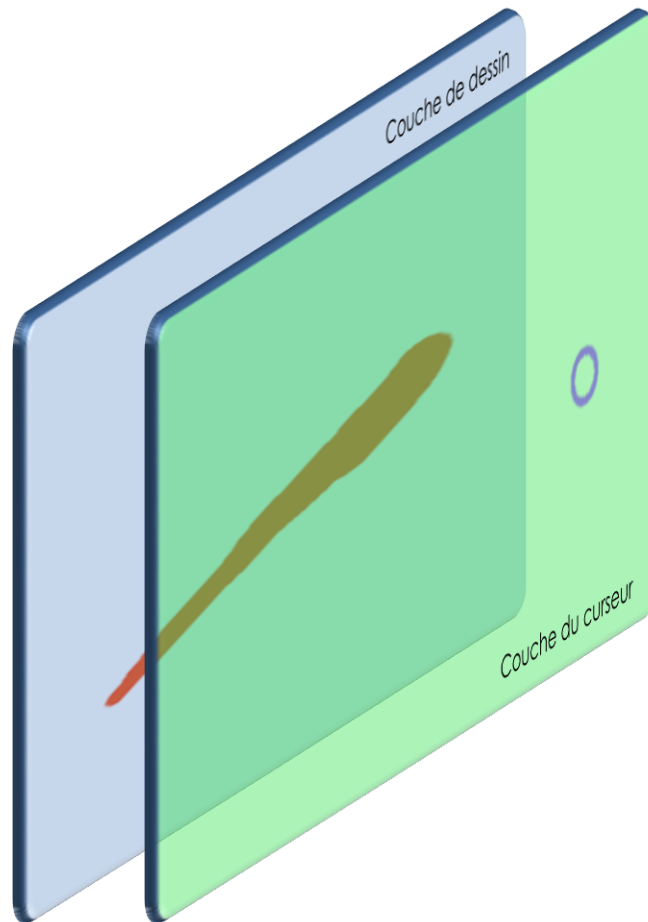


Figure 4.5 : les couches de l'application Kinect Draw

Il y a deux couches :

- La couche d'arrière-plan, soit la couche de dessin, dans laquelle on dessinera.  
Elle est appelée **drawCanvas** dans le code.
- La couche de premier-plan, qui ne sera utilisé que pour l'affichage du curseur.  
Elle est appelée **cursorMoveCanvas** dans le code.

La couche *cursorMoveCanvas* contient une ellipse qui est utilisée comme curseur. Celle-ci est déjà implémenté dans le code.

## 4.4 Petite introduction à la structure de ce projet Windows 8.1

Quitter l'application si ce n'est pas déjà fait

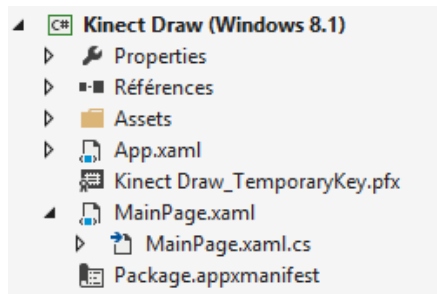


Figure 4.6 : La structure de fichier de KinectDraw

La Figure 4.6 ci-contre présente l'arborescence de fichier du projet Kinect Draw. Ce chapitre ne s'attardera pas à la structure d'une application Windows 8.1, mais présentera les points nécessaires à la réalisation du projet.

Lorsque l'application est lancée, on accédera à la page *MainPage*, représentée par les deux classes *MainPage.xaml* et *MainPage.xaml.cs*.

Le fichier *MainPage.xaml* contient le design de la page principale. C'est un fichier XML. La manière de préparer la structure d'une page est très proche de l'html. Ce fichier a déjà été mis en page dans l'archive fournie pour correspondre au deux canevas décrits au point 4.3, et ne nécessite aucune retouche.

Ouvrir le fichier *MainPage.xaml.cs*

C'est dans ce fichier qu'il sera nécessaire d'ajouter le code implémentant les fonctionnalités de Kinect.

Ce code contient déjà quelques lignes qui faciliteront l'intégration de Kinect.

On peut y trouver la classe interne *CursorPosition* qui représente le curseur sur l'écran. Elle est déjà instanciée, et disponible sous le nom de *this.cursorPosition*. Cette instance devra être utilisée dans le code pour permettre de déplacer le curseur en fonction de la position de la main.

Voici comment l'utiliser :

- *this.cursorPosition.PosX = 45 ;*  
Le curseur se déplacera au pixel 45 selon l'axe des x de la fenêtre
- *this.cursorPosition.PosX ;*  
Retourne la position du curseur courante
- De la même manière, *this.cursorPosition.PosY* permet de modifier et de récupérer la position courante du curseur selon l'axe y

Lorsque la position du curseur sera modifiée via cette instance, le curseur se déplacera automatiquement à l'affichage.

On y trouve la constante *CURSOR\_DIAMETER* qui indique le diamètre en pixel du cercle représentant le curseur à l'affichage :

```
private const double CURSOR_DIAMETER = 30;
```

Il n'est pas nécessaire de modifier cette constante, mais elle sera utilisée dans le code.



```
| private Brush drawBrush = new SolidColorBrush(Colors.OrangeRed);
```

Ce pinceau sera utilisé pour dessiner à l'écran. Il est de couleur *Orange-Rouge*, mais cette couleur peut être modifiée à convenance.

La classe contient de plus le point *lastPosition* indiquant la position du dernier point du curseur.

```
| private Point? lastPosition = null;
```

Comme nous le verrons plus loin, ce point permettra de dessiner des lignes entre la dernière position et la position courante.

Finalement, la méthode *MainPage\_Loaded* sera appelée une fois que tout sera chargé. On initialisera Kinect ici :

```
| void MainPage_Loaded(object sender, RoutedEventArgs e)
| {
|     //#####
|     // TODO: INITIALISER KINECT
|     //#####
| }
```

De plus, il est possible d'accéder à la couche de dessin *drawCanvas*. Dans cet exemple, nous ajoutons une nouvelle ligne :

```
| this.drawCanvas.Children.Add(new Line());
```

*Information : la ligne de code ci-dessus ajoute une ligne dont les positions ne sont pas définies. En conséquence, rien ne s'affiche à l'écran. Il sera nécessaire dans ce projet d'ajouter des lignes dont les positions sont spécifiées.*

## 4.5 La classe *KinectCoreWindow*

Cette classe est fournie par l'API de Kinect. Elle permet, entre autre, de récupérer la position des mains par rapport au corps d'un joueur rapidement. La main est interprétée comme un *curseur*. On comprend très vite son utilité dans ce projet.

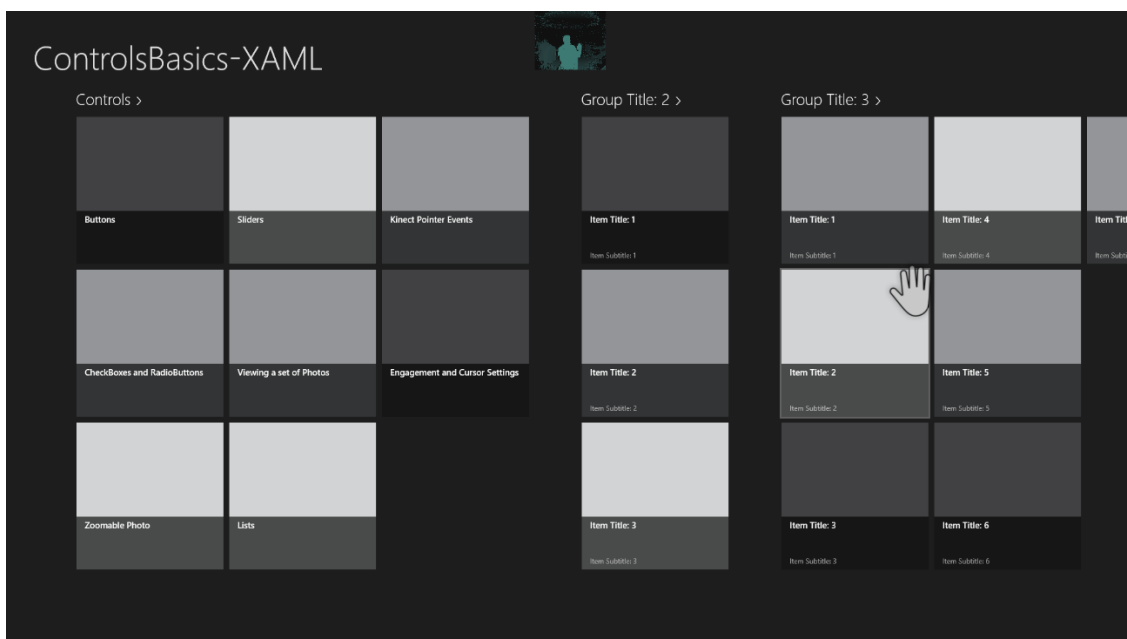


Figure 4.7 : La main est utilisée comme curseur pour interagir avec l'écran



Dans la Figure 4.7, on voit l'utilité que peut avoir la classe *KinectCoreWindow*. La position de la main est retranscrite à l'écran et elle permet ainsi d'interagir avec les contrôles.

Cette classe fonctionne sur le principe d'écran virtuel. Elle crée une zone devant chaque joueur représentant un écran. Si la main est dans cette zone, elle correspond à une certaine position réelle sur l'écran.

Cette zone est appelée *PHIZ*<sup>1</sup> par Microsoft. On voit bien à quoi elle correspond sur la figure ci-contre. Cette zone possède une certaine taille, qui varie en fonction de la personne.

Vue de face, la main est dans cette zone, à une certaine position, selon un axe x-y. C'est cette position que permet de retourner *KinectCoreWindow*.

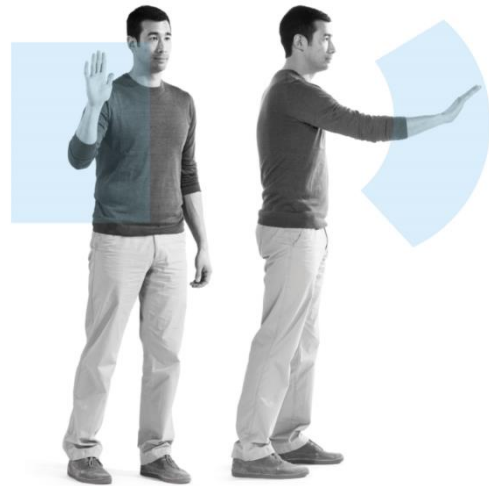


Figure 4.8 : La zone d'interaction physique

### L'emploi de cette classe

On utilise cette classe de manière très similaire aux *frames*, telles que vues dans le chapitre 0.

```
KinectCoreWindow kinectCore = KinectCoreWindow.GetForCurrentThread();  
kinectCore.PointerMoved += kinectCore.PointerMoved;
```

*KinectCoreWindows.GetForCurrentThread()* permet de récupérer un objet *KinectCoreWindow*. Cette classe contient l'évènement *PointerMoved*, qui est appelé à chaque fois que le pointeur, soit la main d'un joueur, bouge. Dès que Kinect détecte un corps, celui-ci *track* la position de ses deux mains par rapport à son corps.

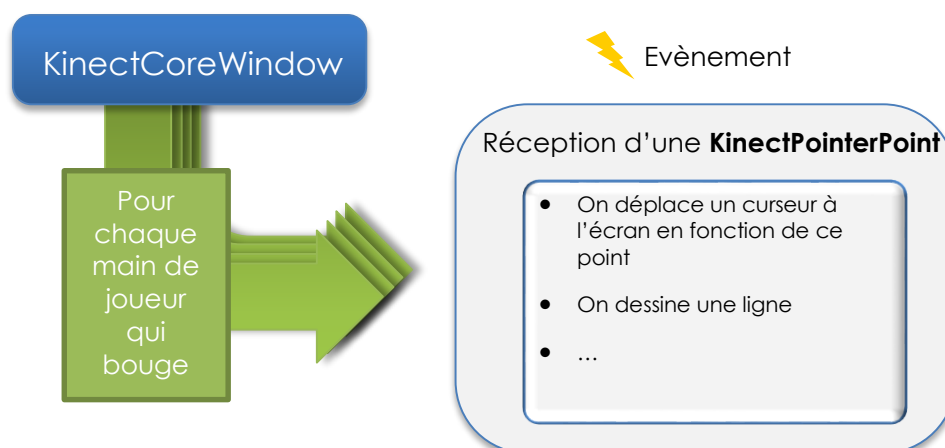


Figure 4.9 : Exemple d'utilisation de l'évènement *PointerMoved*

<sup>1</sup> Physical Interaction Zone

Lorsqu'un joueur a bougé sa main, on appelle une méthode qui contiendra en paramètre la nouvelle position de la main selon un axe x-y.

Voici la méthode appelée lorsque la main d'un corps s'est déplacée :

```
void KinectCore_PointerMoved(KinectCoreWindow sender, KinectPointerEventArgs args)
{
    KinectPointerPoint point = args.CurrentPoint;
}
```

*Remarque : Cette méthode est appelée pour chaque joueur, et pour chaque main. Par exemple, si deux corps sont détectés par Kinect, cette méthode pourra être appelée 4 fois « en même temps » : 2 mains par personne.*

Le *point* ainsi récupéré contient plusieurs informations utiles sur la position de la main dans la PHIZ. Celles-ci seront utilisées dans ce projet :

- *point.Position.X* : récupère la position selon l'axe des X du curseur, en %
- *point.Position.Y* : récupère la position selon l'axe des Y du curseur, en %
- *point.Properties.HandType* : permet de connaître quelle est la main correspondant au pointeur, soit *HandType.RIGHT*, *HandType.LEFT* ou *HandType.None* si elle n'est pas connue
- *point.Properties.HandReachExtent* : récupère en % à quel point la main a atteint le point maximum d'extension. C'est un double entre 0 et 1 (la plupart du temps) :
  - 0.0 → la main est très proche de l'épaule, le joueur « n'appuie » pas du tout
  - 1.0 → le bras est tendu, le joueur « appuie » le plus possible

Les positions X et Y du point sont des *double* donnés en %. C'est-à-dire que si la main est à gauche de l'écran, sa position X vaut 0, si elle est à droite de celui-ci, il vaut 1, comme on le voit dans la Figure 4.8 ci-dessous :

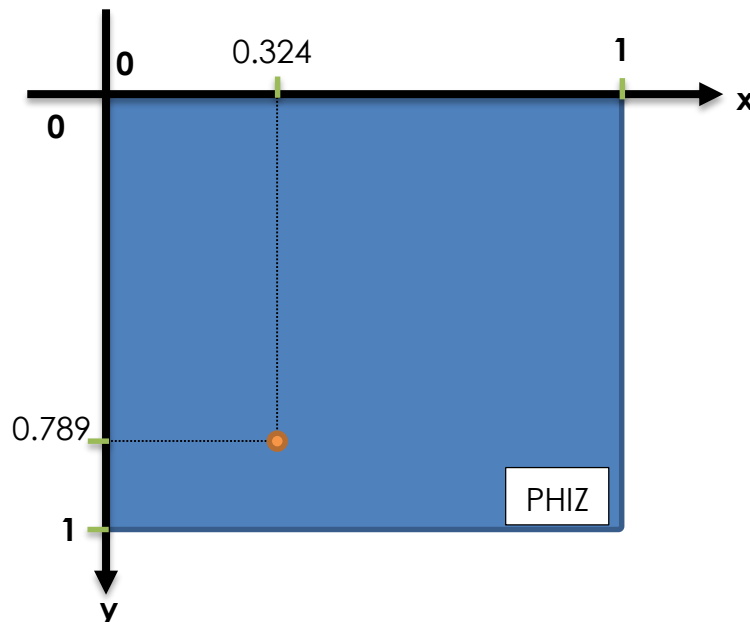


Figure 4.10 : La position de la main dans la PHIZ est retournée en %

Il faudra donc penser à traduire cette position en un système de coordonnées en pixel

## 4.6 Développement et conseils

Ouvrir le projet si ce n'est pas déjà fait

Dans ce sous-chapitre, il sera décrit comment débiter le projet, quelques points supplémentaires qui aideront au développement, ainsi que quelques pistes d'algorithme.

*Information : Il est vivement conseillé de ne pas copier-coller le code écrit dans ce tutoriel, mais de le taper en entier, ce qui permettra une meilleure compréhension et une découverte des méthodes de l'API non décrites dans ce tutoriel.*

### Commencer le projet

Ouvrir la page `MainPage.xaml.cs`

Atteindre la partie Kinect dans les déclarations (« à intégrer ici, les déclarations de Kinect ») en haut du fichier

Nous allons déclarer ici les instances nécessaires au projet utilisant les API Kinect. Il est nécessaire de suivre les instructions de ce chapitre point par point, sans en oublier.

D'une part, ajouter le code suivant dans les *déclarations de Kinect* :

```
| KinectSensor kinectSensor;
```

Cette instance représente le capteur Kinect. On l'utilisera pour, par exemple, démarrer Kinect ou l'arrêter.

D'autre part, ajouter la déclaration de l'instance de *KinectCoreWindow* qui sera utilisée pour connaître la position des mains, tel que vu dans le chapitre 4.5. Finalement, cette partie du code devrait ressembler à ce qui suit :

```
| //----- KINECT -----  
|  
| /// <summary>  
| /// Représente un capteur Kinect, permettant de le gérer (commencer la lecture, etc.)  
| /// </summary>  
| private KinectSensor kinect;  
|  
| /// <summary>  
| /// Interface de kinect qui permet de gérer l'interaction utilisateur-machine  
| /// </summary>  
| private KinectCoreWindow kinectCore;
```

Ce sont les deux seules déclarations concernant Kinect qui sont nécessaires à ce projet.

| Atteindre la fonction *MainPage\_Loaded*

---

La fonction *MainPage\_Loaded* sera appelée lorsque la page sera entièrement chargée. C'est ici que nous allons démarrer le capteur Kinect.

Nous devons instancier le capteur Kinect connecté et la classe *KinectCoreWindow* de notre projet.

| Ajouter les deux lignes suivantes à la fonction *MainPage\_Loaded*

---

```
| void MainPage_Loaded(object sender, RoutedEventArgs e)  
| {  
|     //Récupération des objets Kinect nécessaires au programme  
|     this.kinect = KinectSensor.Default();  
|     this.kinectCore = KinectCoreWindow.GetForCurrentThread();  
| }
```

Maintenant, nous devons gérer les événements lorsqu'un joueur bouge une main, tel que vu dans le chapitre 4.5. Pour ce faire :

| Taper la ligne de code ci-dessous à la fonction *MainPage\_Loaded*

---

```
| //On traite les événements permettant de faire bouger le curseur  
| this.kinectCore.PointerMoved += kinectCore_PointerMoved;
```

Lorsqu'on tape cette ligne dans Visual Studio, un pop-up apparaît. En appuyant sur la touche *TAB* deux fois, la création d'une méthode associée est automatiquement créée. Celle-ci est appelée *kinectCore\_PointerMoved*, et a, entre autre, un paramètre contenant un *KinectPointerPoint* indiquant la position d'une main d'un joueur, tel que vu au chapitre 4.5, sous-titre « L'emploi de cette classe ».

Pour finir, ajouter à la méthode `MainPage_Loaded` l'instruction indiquant que le capteur Kinect doit commencer à recevoir des images. Au final, cette méthode doit ressembler à ceci :

```
void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    //Récupération des objets Kinect nécessaires au programme
    this.kinect = KinectSensor.Default();
    this.kinectCore = KinectCoreWindow.GetForCurrentThread();

    //On traite les événements permettant de faire bouger le curseur
    this.kinectCore.PointerMoved += kinectCore_PointerMoved;

    //On ouvre Kinect (Kinect s'allume)
    this.kinect.Open();
}
```

Si l'application est lancée, on remarque que le logo Xbox du capteur Kinect s'allume, ce qui indique que des images sont reçues, et qu'on peut les utiliser dans notre application.

## Le dessin

Normalement, notre projet contient la méthode `kinectCore_PointerMoved`, créée précédemment :

```
void kinectCore_PointerMoved(KinectCoreWindow sender, KinectPointerEventArgs args)
{
}
```

Elle est appelée lorsqu'un joueur suivis déplace une de ses mains. C'est ici qu'il faudra ajouter le gros du code.

Pour cette partie, peu de code est fourni. Cependant, quelques pistes et conseils sont donnés.

Vérifier la pression du pointeur courant avant de dessiner

Il est possible de récupérer la pression du `KinectPointerPoint` de la même manière que vu dans le chapitre 4.5. Il ne faut dessiner une ligne qu'à partir d'une certaine pression. Une pression de 0.5 est un bon compromis.

*Remarque : lorsque le joueur n'appuie plus, aucune ligne ne doit être dessinée !*

Ajouter une ligne

Lors du développement de ce projet pour le tutoriel, il a été vu que dessiner des lignes entre chaque point était la meilleure façon de faire, avec un résultat visuel meilleur.

C'est pourquoi le point local `this.lastPosition` a été ajouté, indiquant la position du dernier point pressé, soit le début de la ligne. Ce point peut être :

- `this.lastPosition == null` : indique qu'il n'y a pas de dernier points, c'est-à-dire que la ligne n'a pas de commencement. Cela arrive, par exemple, au lancement de l'application, ou encore lorsque le joueur ne veut plus dessiner (on a relâché la pression)

- *this.lastPosition* peut aussi indiquer des valeurs X-Y, qu'on peut récupérer avec *this.lastPosition.Value.X*, respectivement *this.lastPosition.Value.Y*

Si on connaît la position du dernier point pressé, et que le point courant est lui aussi pressé, on peut dessiner une ligne entre ces deux points sur le canvas de dessin, appelé *drawCanvas* (voir chapitre 4.4).

Gérer le nombre de pointeurs (2 par personne)

Pour ce projet « découverte », il est conseillé d'admettre qu'une seule personne est présente et détectée par Kinect. En effet, décider de la personne qui dessine actuellement nécessite plus de détails et d'information sur le concept d'engagement.

C'est pourquoi, si plusieurs personnes sont détectées par Kinect, la méthode *kinectCore\_PointerMoved* pourra être appelée plusieurs fois en « même temps », et le *KinectPointerPoint* différera à chaque fois.

Il est donc conseillé de :

- Se tenir assez près de Kinect, à moins de 2 mètres, et bien en face, de telle sorte que le capteur ne détecte qu'une seule personne. Si la caméra a détecté plusieurs personnes, redémarrer l'application.
- Ajouter une condition pour que, par exemple, seul la main droite puisse être utilisée. Le *KinectPointerPoint* courant ne doit être traité que s'il provient de la main droite. On a vu dans le chapitre 4.5 la propriété *point.Properties.HandType*, qui peut être utile. Ceci a pour but que seul un pointeur par personne ne soit traité, une personne possédant 2 mains.

L'épaisseur du pinceau

Il est possible de dessiner en fonction de la pression effectuée par le joueur, c'est-à-dire que le diamètre de la ligne augmente en fonction de la distance séparant l'épaule de la main (utiliser *point.Properties.HandReachExtent*).

Cependant, dans un premier temps, cette fonctionnalité peut être ignorée totalement, et ajoutée ultérieurement. Il est donc conseillé de commencer à dessiner des lignes d'une épaisseur de 30 pixels, puis, une fois le code fonctionnel, modifier le diamètre de cette ligne en fonction de la pression effectuée.

## 4.7 Correction

Un code est disponible dans l'archive fournie, dans le dossier *KinectDraw-Final*. Il est possible de s'en inspirer pour certains algorithmes. Cependant, ce code n'est pas présenté comme l'unique solution, mais comme proposition. Il est d'ailleurs nettement perfectible.

## 5 Deuxième projet : Kinect PowerPoint

### 5.1 Description du projet

Ce projet est un module complémentaire à PowerPoint. Il permettra de passer d'un *slide*<sup>1</sup> à l'autre à l'aide d'un geste de la main/bras. La main gauche ou la main droite peuvent être utilisées, et on peut avancer au slide suivant, ou revenir au slide précédent.

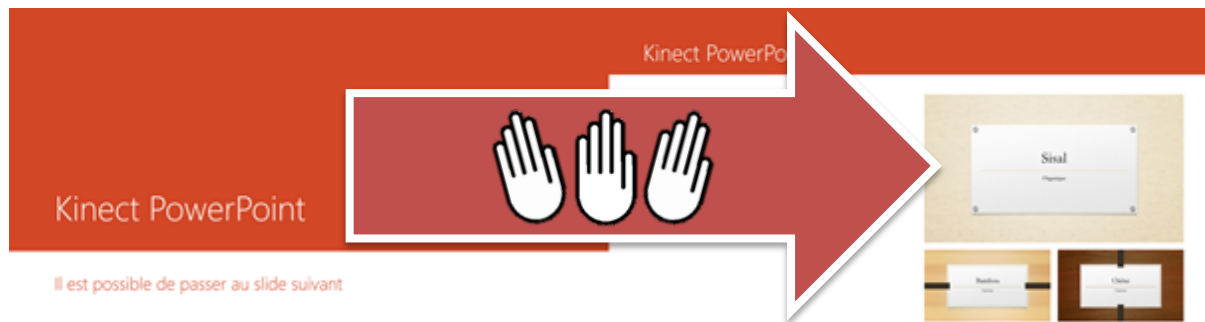


Figure 5.1 : Le changement de slide peut s'effectuer d'un geste

La figure ci-dessous montre le mouvement à effectuer. Si on veut, par exemple, passer à la slide suivante, il est nécessaire de :

1. Tendre un bras à droite
2. Le ramener vers la gauche

Ceci permet de simuler un geste de « balayement ». Lorsqu'on swipe de droite à gauche, il faut l'interpréter comme si on poussait la slide courante à gauche pour faire apparaître le slide suivant.



Figure 5.2 : Le mouvement pour changer de slide

Aucune capture d'écran de cet add-in ne peut être prise. En effet, il ne contient pas d'interface graphique.

<sup>1</sup> Slide : une diapositive de PowerPoint

## 5.2 Mise en place et prérequis

Depuis l'archive fournie, ouvrir le projet *KinectPowerPoint* (dans le dossier *KinectPowerPoint-Base*) dans Visual Studio.

Ce projet de base contient le code nécessaire pour que l'add-in fonctionne, ainsi que quelques lignes de code relatives à Kinect.

De plus, le nécessaire a été fait pour que le développement sur Kinect puisse commencer de suite :

- Les références de l'API de Kinect nécessaires ont été ajoutées
  - Microsoft.Kinect
  - Microsoft.Kinect.VisualGestureBuilder
- L'application a été ciblée pour les plateformes x86  
Ceci permet que l'add-in développé dans ce projet puisse être intégré dans les deux versions x86 et x64 de PowerPoint.
- Les technologies de détection de mouvements ont été intégrées au dossier de l'add-in :  
C'est le dossier *vgbtechs*, qui contient les deux technologies de détection de mouvement dont nous parlerons plus tard.
- Le fichier de base de données de mouvement *SwitchKinect.gbd* est fourni et intégré à l'add-in :  
Ce fichier sera abordé plus loin dans ce tutoriel.
- Le fichier contenant la classe *GestureDetector* est dans un premier temps fourni :  
L'utilité de cette classe sera expliquée, et pourra être développée

**Information :** Il est nécessaire d'avoir sur son PC une version de PowerPoint 2013

## 5.3 Structure et principe du projet

Ouvrir le projet KinectPowerPoint si ce n'est pas déjà fait
Vérifier que les configurations soient « debug » et « x86 » (au haut de Visual Studio)
Démarrer l'add-in

PowerPoint est exécuté. On ne remarque aucune particularité au lancement de l'add-in, mis à part que le logo du capteur Kinect s'allume.

En effet, le code fourni contient déjà le nécessaire pour que l'initialisation du capteur soit faite.

Ce projet permettra que plusieurs personnes puissent changer de slide. Ainsi, contrairement au précédent projet *Kinect Draw*, le fait que plusieurs *body* soient détectés ne posera pas de problème à l'utilisation.



La figure ci-dessous présente le cheminement de l'add-in :

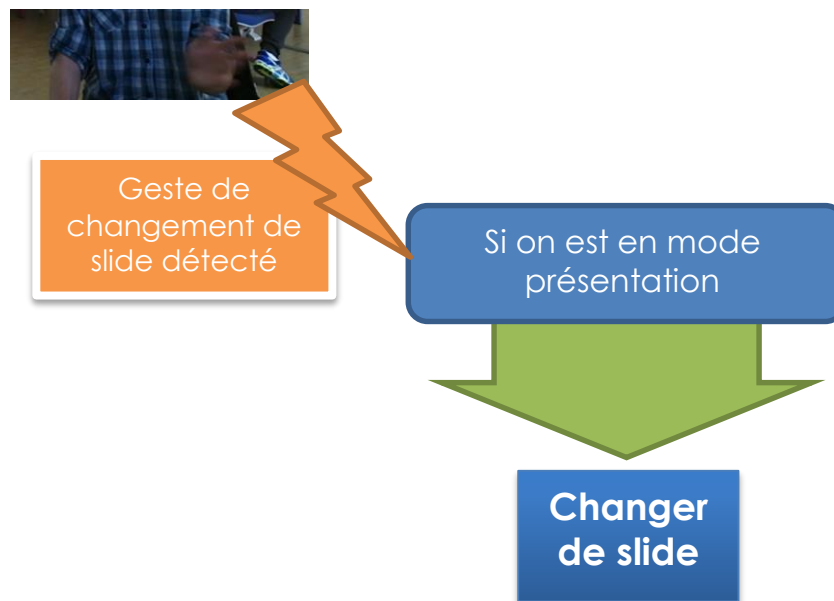


Figure 5.3 : Le processus de changement de slide

Le principe de base est simple :

1. À l'aide de la classe *VisualGestureBuilderFrameSource* qui sera décrite au chapitre 5.4, des mouvements spécifiques pourront être détectés. Ils seront au nombre de 4 :
  - Le geste pour passer au slide suivant de la main droite, appelé *OutIn\_Right*, pour signifier le fait que le bras droit passe de l'extérieur à l'intérieur
  - Le geste pour passer au slide suivant de la main gauche, appelé *InOut\_Left*
  - Le geste pour passer au slide précédent de la main droite, appelé *InOut\_Right*
  - Le geste pour passer au slide précédent de la main gauche, appelé *OutIn\_Left*
2. Lorsqu'un de ces gestes est détecté, on vérifie si on est en mode présentation, c'est-à-dire que l'utilisateur présente ses slides, et ne les modifie pas
3. Si c'est le cas, on envoie la commande à PowerPoint permettant de passer au slide suivant, respectivement au slide précédent

**Information** : ici, il est écrit que l'add-in se chargera de changer de slide, cependant, il est possible que certaines animations aient été ajoutées et qu'elles doivent apparaître l'une après l'autre au clic. C'est ce clic qui est simulé par l'add-in.

## 5.4 La détection de mouvement avec *Visual Gesture Builder*

*Visual Gesture Builder* est le système de détection de mouvement fourni dans l'API de Kinect. Un logiciel utilitaire nécessaire à la détection de mouvement porte aussi ce nom.

La figure ci-dessous montre les étapes à mettre en place pour détecter certains mouvements spécifiques avec l'API fournie par Microsoft :

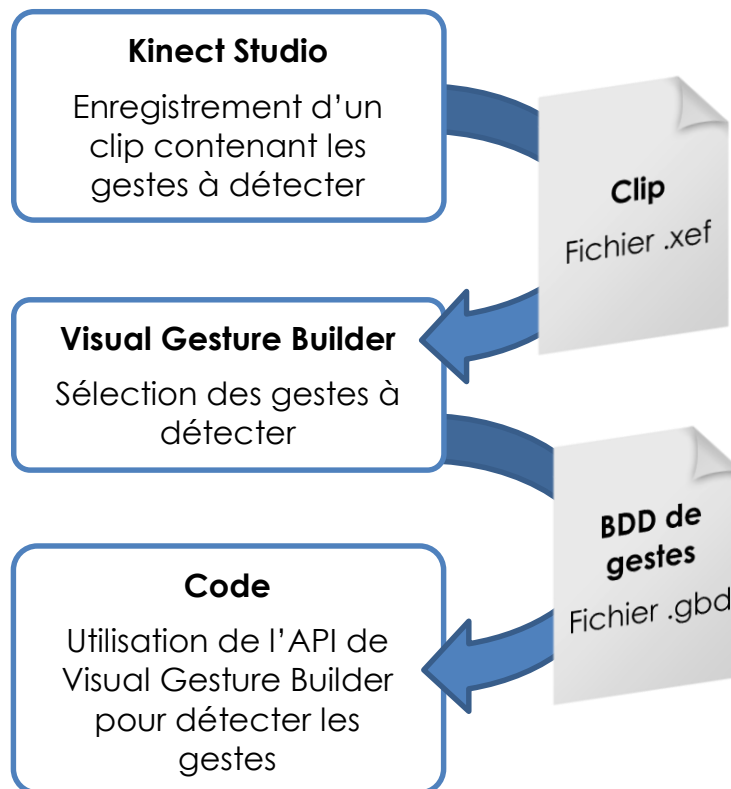


Figure 5.4 : La détection de mouvement grâce à *Visual Gesture Builder*

L'utilisation de deux logiciels s'impose : *Kinect Studio* et *Visual Gesture Builder*. Il n'est pas nécessaire de les télécharger : ils font en effet partis du SDK de Kinect.

### **Kinect Studio**

Premièrement, il est nécessaire d'utiliser le logiciel *Kinect Studio*.

Celui-ci permet entre autres d'enregistrer, rejouer et éditer des clips de *data sources* que fournit Kinect. Ce sont les mêmes *data sources* que celles vues au chapitre 3.1. On peut donc choisir d'enregistrer dans un même fichier .xef les données reçues de la caméra couleur (*ColorFrameSource*) et de la caméra infrarouge (*InfraredFrameSource*).

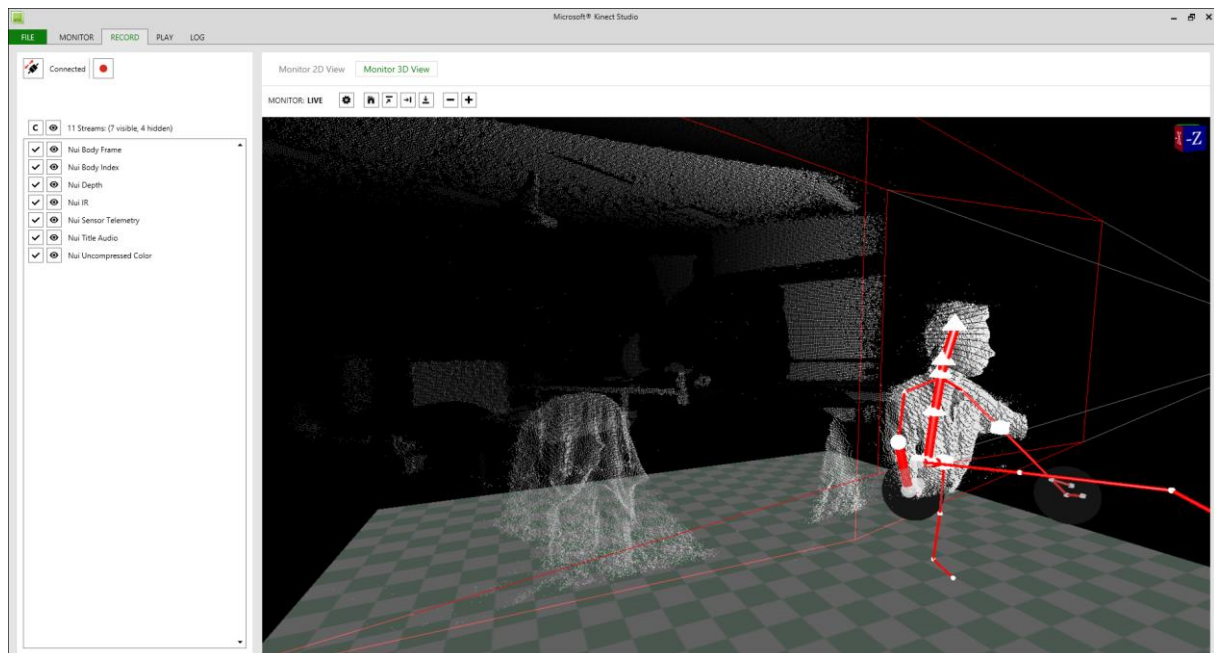


Figure 5.5 : Le logiciel Kinect Studio

Pour la détection de mouvement, ce logiciel est utilisé pour enregistrer une personne pendant qu'elle effectue certains mouvements spécifiques. Ce sont ces mouvements qui pourront être détectés dans le code. Par exemple, dans ce projet *Kinect PowerPoint*, un clip .xef a été enregistré lorsqu'une personne a effectué les gestes de balayement tels que vus aux chapitres 5.1 et 5.3 (de gauche à droite avec les deux bras, et vice-versa).



### Le logiciel *Visual Gesture Builder* et les technologies de détection



Deuxièmement, le logiciel *Visual Gesture Builder* est utilisé. Il permet de créer un fichier de base de données contenant les spécificités de certains mouvements qu'on aimerait détecter par la suite. Par exemple, on pourrait y trouver la description du geste « lever la main » : la main est premièrement en dessous de la tête, puis au même niveau avec une certaine vitesse, et, finalement, au-dessus de la tête.

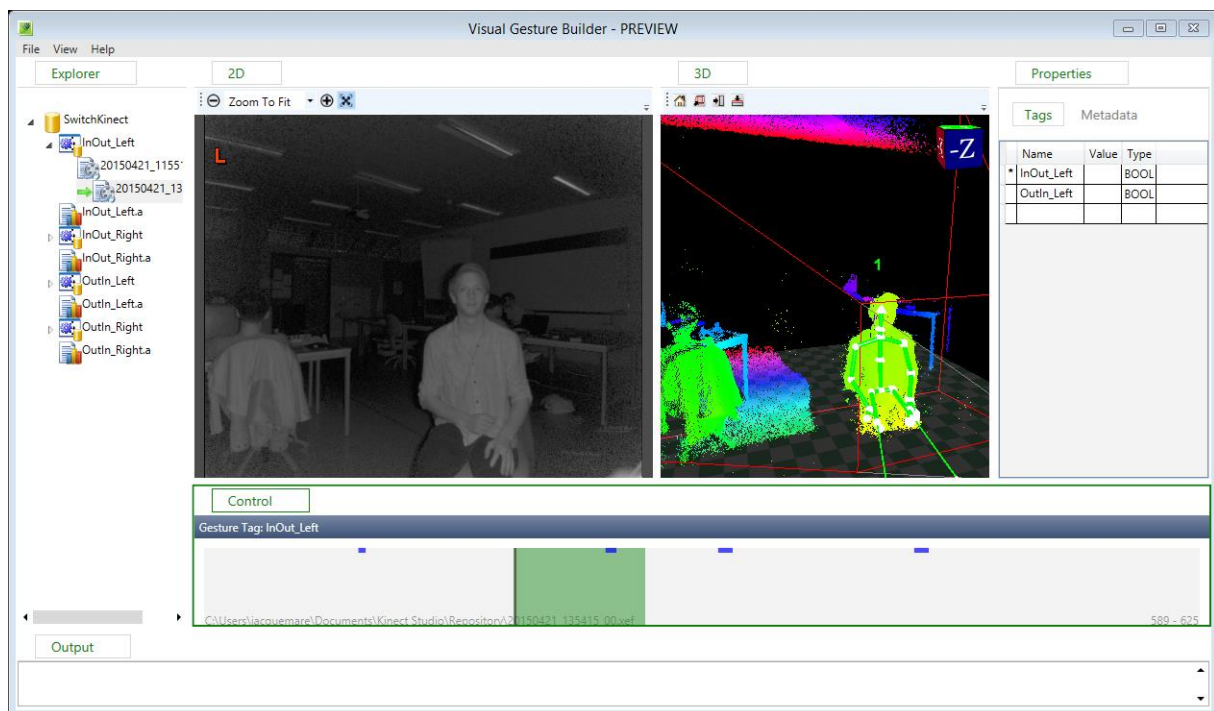


Figure 5.6 : Le projet Visual Gesture Builder de Kinect PowerPoint

Ce logiciel permet de créer un nouveau projet de base de données, et d'y importer divers clip .xef sur lesquels on indique les moments clés d'un mouvement. De ce projet, on peut construire un fichier de base de données .gbd grâce au *machine learning*<sup>1</sup>.



Deux types de *gesture*<sup>2</sup> détectables sont disponibles dans VisualGestureBuilder :

- Les gestes dits « discrets », utilisant la technologie de Microsoft AdaBoostTech. Ce sont des gestes qui peuvent être soit détectés, soit non-détectés ; c'est un booléen.
- Les gestes avec progression, utilisant la technologie de Microsoft RFRProgressTech. Ce sont des gestes qui indiquent un float en % du geste cherchant à être produit. Par exemple, si on souhaite, dans un jeu Kinect, animer un swing de golf, on a besoin de la progression de ce swing. C'est donc cette technologie qui devrait être utilisée.

*Dans le cas du projet Kinect PowerPoint, la technologie AdaBoostTech est utilisée. En effet, même si le geste en lui-même (déplacement du bras de droite à gauche pour swiper) a une idée de progression, il est nécessaire de connaître uniquement si elle est détectée ou non.*

<sup>1</sup> Le *machine learning*, ou l'apprentissage automatique en français, est une discipline qui permet à l'ordinateur « d'apprendre »

<sup>2</sup> Le geste, ou le mouvement

## L'implémentation à l'aide de `VisualGestureBuilderFrameSource`

Cette classe est fournie par l'API de Kinect. Ce n'est pas une source fournie directement par le capteur, tel que vu au chapitre 3.1 *Les data sources*. En effet, cette classe utilise ces *data sources*, notamment `BodyFrameSource`, pour détecter les mouvements des joueurs. Cependant, sa structure est en tout point concordante avec les *data sources* : une `VisualGestureBuilderFrame`, contenant les données relatives à la détection de mouvement, est déclenchée 30 fois par seconde.

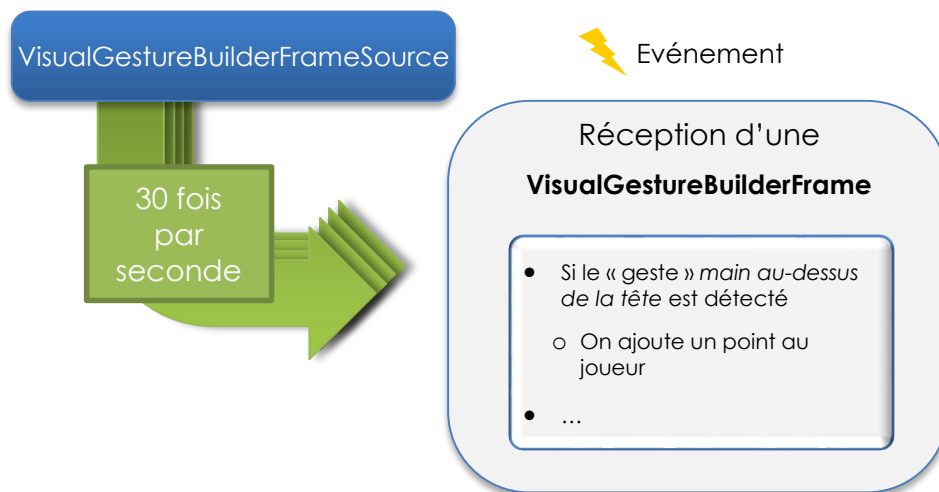


Figure 5.7 : Exemple d'utilisation de `VisualGestureBuilderFrame`

Cet évènement contient une `VisualGestureBuilderFrame` en paramètre comportant un *tableau* de *gesture*, qui dresse la liste de toutes les gestes détectés.

Deux tableaux sont disponibles lors de la réception de l'évènement :

- Le tableau `DiscreteGestureResults` : comme son nom l'indique, il contient la liste des gestes de type discret, détecté à l'aide de la technologie *AdaBoostTech*, tel que vu en page 34.
- Le tableau `ContinuousGestureResults` : il contient la liste des gestes avec une progression, détecté à l'aide de la technologie *RFRProgressTech*, comme vu en page 34

Dans ce projet, uniquement la technologie `DiscreteGesture` sera utilisée.



Figure 5.8 : Le contenu d'une `VisualGestureBuilderFrame`

On voit ci-dessus le contenu d'une de ces *frame*. Elle contient un tableau de gestes, comme par exemple `DiscreteGestureResults`. Ce tableau peut contenir

le geste « Balayement de la main droite vers la gauche », ou appelé *OutIn\_Right* dans le code, tel que vu au chapitre 5.3. Il peut aussi ne rien contenir dans le cas où aucun geste n'est actuellement détecté.

Pour chaque geste, on trouve aussi plus d'information, tel que un booléen indiquant si c'est la première fois qu'on le détecte. En effet, un mouvement prend un certain temps, et peut s'étendre sur plusieurs *frame*, comme on le voit dans la figure ci-dessous :

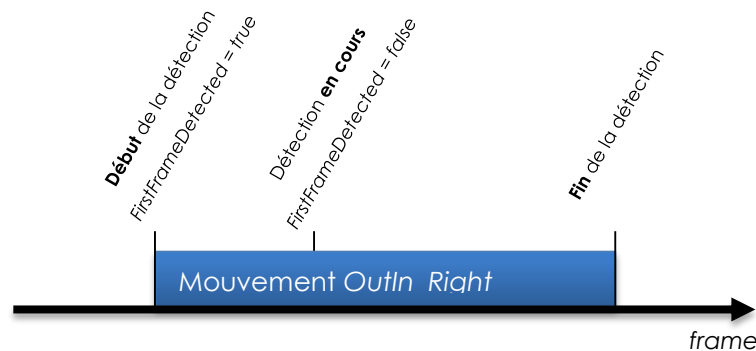


Figure 5.9 : La détection d'un mouvement est en cours sur plusieurs *frame*

Ainsi, lorsqu'un mouvement est pour la première fois détecté, *FirstFrameDetected* est vrai. Mais 33 ms plus tard, soit à la *frame* suivante, le mouvement est toujours détecté : on reçoit donc dans le tableau *DiscreteGestureResults* le mouvement *OutIn\_Right*, mais *FirstFrameDetected* est faux. Lorsque le mouvement est terminé, ce tableau ne contient plus le geste *OutIn\_Right*.

Dans ce projet, il sera nécessaire de vérifier que *FirstFrameDetected* est vrai pour lancer la commande de changement de slide. En effet, si on ne suit pas cette condition, l'instruction pour passer au slide suivant s'exécutera toute les 33 ms pendant l'entier du temps que durera le geste.

Il est à noter qu'il est essentiel d'implémenter plusieurs *VisualGestureBuilderFrameSource* dans le code : en effet, on doit en instancier 6, soit le nombre de *body* détectable par Kinect. Ainsi, chaque évènement déclenché sera lié à un *body* particulier, comme on le voit dans la Figure 5.10 ci-dessous :

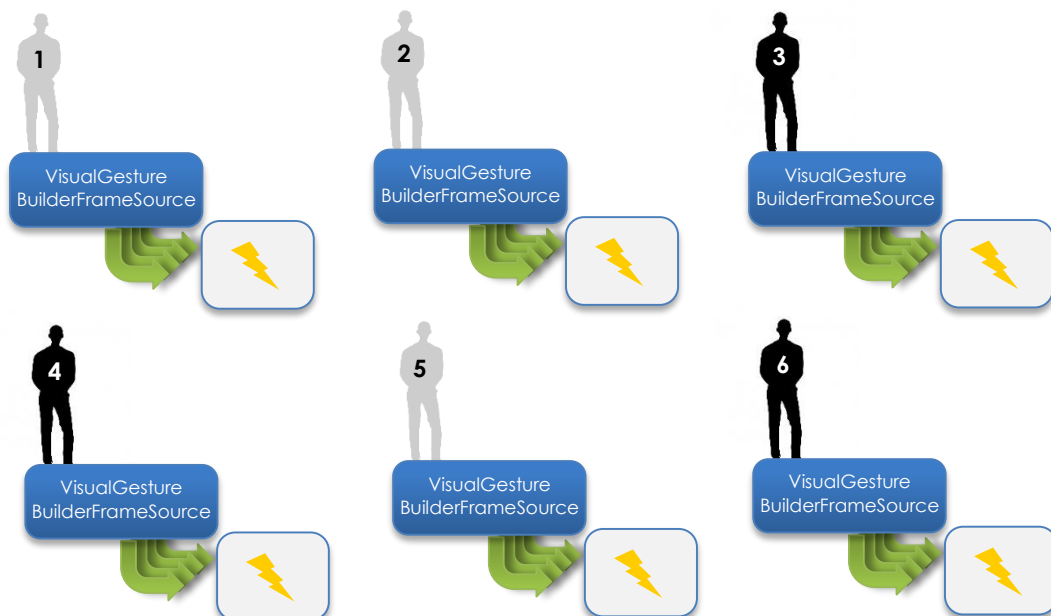


Figure 5.10 : Les *VisualGestureBuilderFrameSource* instanciée

Pour chaque personne qui pourrait être détecté, on crée une nouvelle instance de *VisualGestureBuilderFrameSource*. Ici, seules 3 personnes sont présentes devant le capteur : les joueurs 3, 4 et 6. Cependant, les 6 instances de *VisualGestureBuilderFrameSource* auront été créées.

Lorsqu'on instancie une *VisualGestureBuilderFrameSource*, on indique l'ID du joueur associé, comme on le voit ci-dessous :

```
gestureSource = new VisualGestureBuilderFrameSource(KinectSensor.GetDefault(), 0);
```

Ici, l'ID du joueur est le 0, c'est-à-dire qu'aucun joueur n'est associé à cette source à son instanciation. Mais si une nouvelle personne apparaît devant Kinect, il est possible de modifier l'ID du joueur *tracké* de la source comme ceci :

```
gestureSource.TrackingId = 4566221;
```

*Information : chaque body contient un ID, comme on le verra un peu plus loin dans ce chapitre. C'est cet ID qu'il faut indiquer dans la *VisualGestureBuilderFrameSource* instanciée.*

Pour aider au développement de la détection de mouvement dans ce projet, la classe *GestureDetector*, fournie, a été codée. Elle correspond à un détecteur de mouvement pour un certain joueur. Elle contient les évènements *GestureDetected* et *GestureFirstDetected* qui sont déclenchés lorsque un



mouvement a été détecté, pour la première fois ou non. Cette classe est utilisable de suite et son utilisation sera vue plus loin (p. 39).

## 5.5 Principe de la détection de mouvement dans le projet

### Récupérer la liste des *body* présents

Pour ce projet, la classe *BodyFrameSource*, vue au chapitre 3.1, sera utilisée, ainsi que la *BodyFrame* correspondante. Pour rappel, la source *BodyFrameSource* permet de récupérer une liste de tous les *body*<sup>1</sup> visibles par Kinect. L'utilisation de celle-ci est donnée en exemple dans le chapitre 0, en page 17. La figure ci-dessous le reprend dans le cadre du projet *Kinect PowerPoint* :

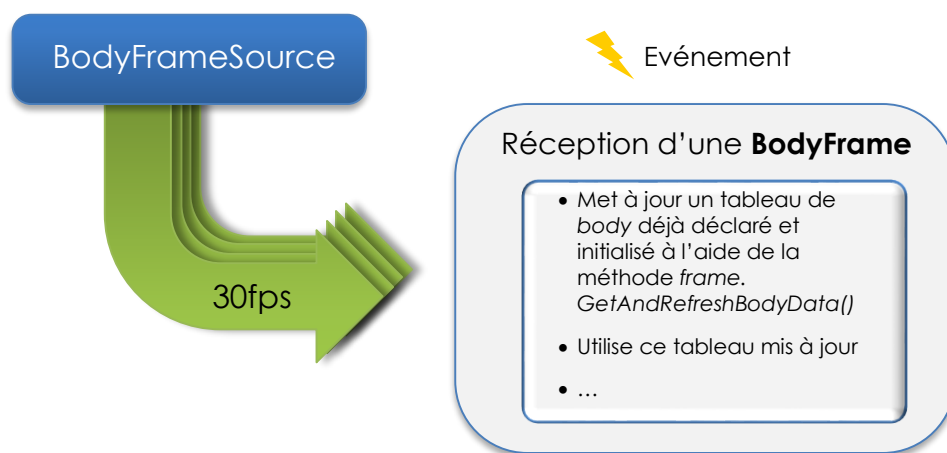


Figure 5.11 : Une *BodyFrame* et son tableau de joueur

La *BodyFrame* contient la méthode *GetAndRefreshBodyData*. Elle prend en paramètre une liste de *body* et se chargera d'en mettre à jour chaque élément.

*Information : il sera nécessaire de créer cette liste avant de l'utiliser dans la méthode *GetAndRefreshBodyData*.*

<sup>1</sup> Les corps, soit les joueurs qui sont visibles par Kinect.



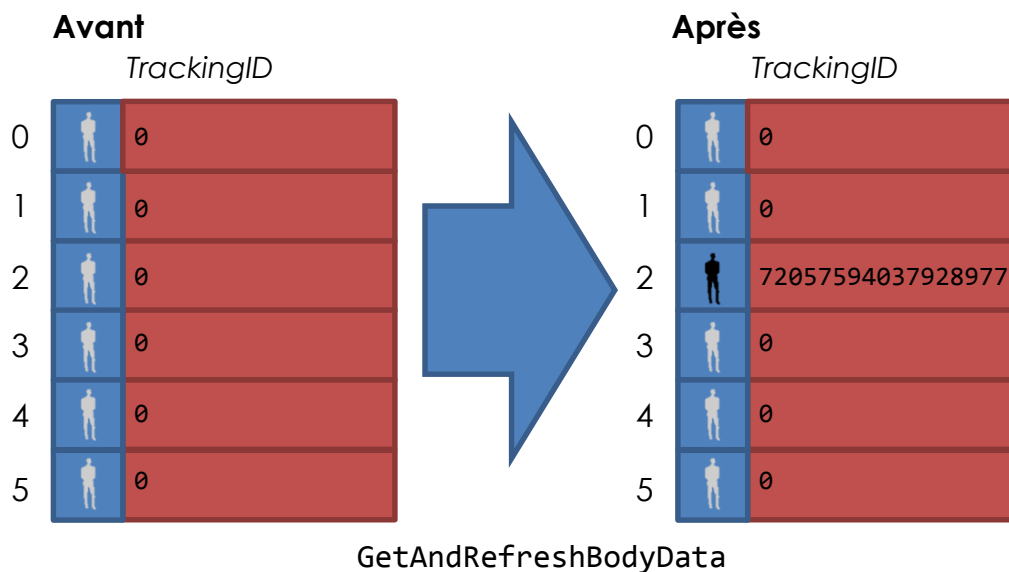


Figure 5.12 : L'effet de la méthode `GetAndRefreshBodyData`

Dans l'exemple ci-dessus, on compare le tableau de *body* avant et après l'application de la méthode `GetAndRefreshBodyData`.

Chaque emplacement du tableau contient un *body*. Cependant, avant que la méthode ne soit appliquée, les *TrackingID* de chaque *body* étaient égaux à 0 : aucun joueur n'était détecté par Kinect.

Après l'utilisation de la méthode `GetAndRefreshBodyData` sur le tableau de *body*, le tableau a été mis à jour : un joueur a été détecté par Kinect durant les 33 dernières millisecondes. On a donc un *body* *tracké*, avec un certain *TrackingID*. On trouve aussi beaucoup d'autres informations concernant ce joueur, tel que sa position dans l'espace, l'état de ces mains, l'orientation de sa tête, etc.

**Information importante :** on remarque que le tableau ne se remplit pas « dans l'ordre » : le premier *body* qui a été détecté n'est pas nécessairement en première position dans le tableau. Toutefois, tant qu'un *body* est suivi par Kinect, il restera toujours associé à la même case du tableau. Ex. : le joueur se situant dans la deuxième case du tableau restera dans celle-ci tant qu'il se trouve dans le champ de vision du capteur Kinect.

### La classe `GestureDetector` fournie

Pour aider au développement de la détection de mouvement dans ce projet, la classe `GestureDetector`, fournie, a été développée. Elle correspond à un détecteur de mouvement et possède des événements utiles qu'on peut suivre.

La classe `GestureDetector` s'instancie ainsi :

```
GestureDetector handler = new GestureDetector(this.originalPath+"\\SwitchKinect.gbd");
```

Ce constructeur crée un `GestureDetector` dont le *TrackingID* n'est pas spécifié.

Par la suite, on peut modifier l'ID à l'aide de l'attribut *TrackingID* de *GestureDetector* :

```
handler.TrackingID = 72057594037928977;
```

Où 72057594037928977 est l'ID d'un *body* tracké. Le *GestureDetector* détectera les mouvements du *body* correspondant.

Il faudra donc dans ce projet créer 6 *GestureDetector*, et modifier leur *TrackingID* en fonction des *body* détectés (à l'aide de *bodyframe*), comme on le voit sur la Figure 5.13 ci-dessous. Le *trackingID* des *GestureDetector* seront donc mis à jour environ 30 fois par seconde.

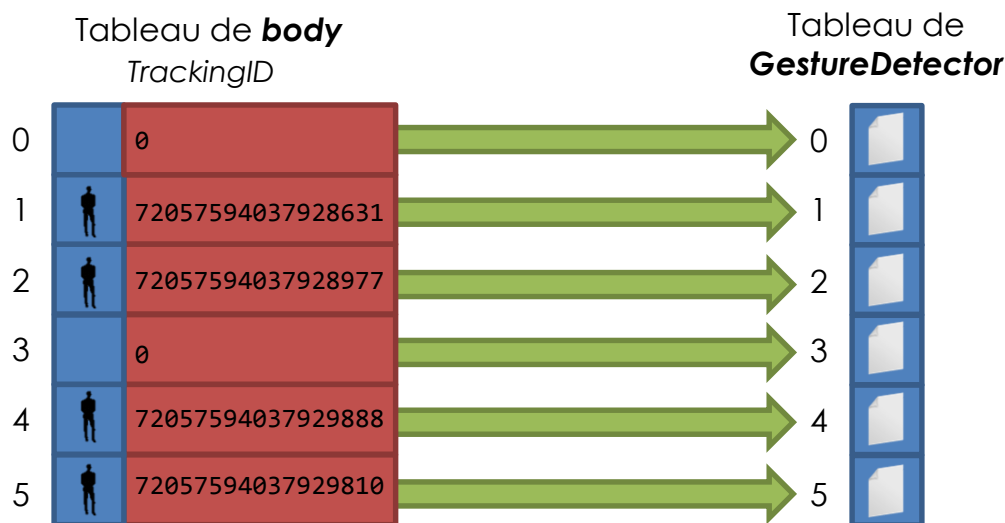


Figure 5.13 : Le tableau de *body* et ses instances de *GestureDetector*

Chaque *GestureDetector* contient les événements suivants :

- *GestureDetected* : cet événement est appelé lorsqu'un mouvement est actuellement détecté. En paramètre, on trouve, entre autre, le *Gesture* détecté
- *GestureFirstDetected* : de la même manière, cet événement est appelé lorsqu'un mouvement est détecté. Cependant, il n'est appelé qu'une seule fois par mouvement, tel qu'on l'a vu précédemment avec la Figure 5.9. C'est cet événement qu'il faudra suivre

Pour suivre un événement, il faudra procéder de la même manière que pour le projet *Kinect Draw* au chapitre 4.5, en page 23. Voici comment débiter la détection des mouvements d'un *GestureDetector* :

```
handler.GestureFirstDetected += handler_GestureFirstDetected;
```

## 5.6 Introduction au développement de cet add-in PowerPoint

Ouvrir le projet *Kinect PowerPoint* si ce n'est pas déjà fait

A l'aide de Visual studio, il est possible de créer des add-in qui seront intégrés à PowerPoint.

La figure ci-dessous représente la structure de fichier du projet :

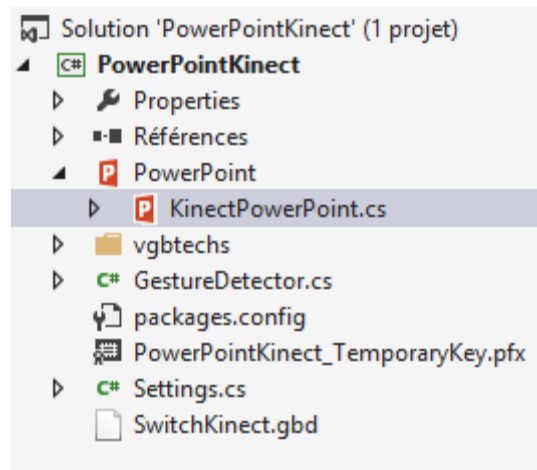


Figure 5.14 : La structure de fichier de Kinect PowerPoint

Le fichier *KinectPowerPoint.cs* contient la classe *KinectPowerPoint*. Elle contient le nécessaire pour faire fonctionner l'add-in. C'est ici que le code devra être ajouté.

Le dossier *vgbtechs* contient les deux fichiers *AdaBoostTech.dll* et *RFRProgressTech.dll*, qui correspondent aux deux technologies de détection de mouvement utilisées par *Visual Gesture Builder*, telles que vues au chapitre 5.4, en page 34. Ce dossier et son contenu ne doit pas être ni renommé, ni modifié. Dans le cas contraire, l'API de *Visual Gesture Builder* ne sera pas capable de détecter les mouvements.

Le fichier *GestureDetector.cs* contient la classe correspondante telle que décrite au chapitre 5.5, en page 39.

Le fichier *SwitchKinect.gbd* est une base de données de mouvement telle que décrite au chapitre 5.5. Il contient les gestes que l'API de *Kinect* devra détecter. Ce fichier est fourni dans le projet de base, mais il pourrait être créé.

## Le déploiement d'add-in Office

Ci-dessous, les principales étapes de l'ajout d'un add-in à un logiciel Office sont vulgairement décrits :

1. **Compilation de l'add-in** : on le trouve maintenant sous forme de fichiers *.dll*.
2. **Ajout d'une clé de registre** : une clé indiquant le chemin de l'add-in Office est ajoutée. Par exemple, dans ce projet, on trouve la clé : *HKEY\_CURRENT\_USER\Software\Microsoft\Office\PowerPoint\Addins\PowerPointKinect*, dont la valeur *Manifest* pointe vers les fichiers compilés.
3. **Chargement de l'add-in** : au démarrage de *PowerPoint*, cette clé est lue et charge l'add-in. Pour ce faire, chaque fichier *dll* est copié dans des sous-dossiers, qu'on peut trouver dans *%localappdata%\assembly\dl3* et charge ceux-ci en mémoire.

Les fichiers *.dll* dans le dossier *dl3* sont isolés les uns des autres, ce qui pose problème pour l'implémentation de *Kinect* dans *PowerPoint*. En effet, le dossier *vbtechs* en devient inaccessible.

Ouvrir le fichier *KinectPowerPoint.cs*

Observer la méthode *initProjectFiles()*

Cette méthode permet de résoudre cette difficulté. Elle est appelée au démarrage de l'application et copie les fichiers nécessaires au bon emplacement pour que l'add-in puisse détecter des mouvements. Elle doit être laissée telle quelle.

Elle définit aussi la chaîne de caractère *this.originalPath*, qui détermine le chemin du dossier contenant les fichiers originaux, soit l'emplacement des fichiers non situé dans *%localappdata%*. On y trouve donc tous les fichiers qui sont inclus dans le projet.

*Au vue du niveau de difficulté, cette méthode n'est pas décrite d'avantage ici. Pour plus d'information sur son fonctionnement, se référer au code source commenté fourni.*

---

<sup>1</sup> *%localappdata%* correspond, la plupart du temps, à *C:\Users\[username]\AppData\Local*

## 5.7 Développement et conseils

Ouvrir le projet si ce n'est pas déjà fait

---

Dans ce sous-chapitre, il sera décrit comment débiter le projet, quelques points supplémentaires qui aideront au développement, ainsi que quelques pistes d'algorithme.

### Commencer le projet

Ouvrir la page *KinectPowerPoint.cs*

---

C'est dans cette classe uniquement qu'il faudra ajouter des lignes pour intégrer la détection de mouvement.

Observer les initialisations de variable en début de fichier

---

On y trouve, notamment, les lignes suivantes :

```
//-----Reconnaissance des corps-----  
private BodyFrameReader bodyReader { get; set; }  
private IList<Body> bodies;  
private IList<GestureDetector> bodiesDetectors;
```

Le *reader bodyReader* permet de lire les données de la source *BodyFrameSource*. Son instantiation sera abordée un peu plus loin dans ce chapitre.

La liste *bodies* correspond au tableau de 6 *body* tel que vu au chapitre 5.5. Le code l'instanciant a déjà été ajouté dans la méthode *initKinect()*.

La liste *bodiesDetectors* correspond à un tableau de 6 *GestureDetector*, tel que vu au chapitre 5.5 en page 39. Il sera nécessaire d'ajouter les instructions ajoutant des nouveaux *GestureDetector* à ce tableau.

Atteindre la méthode *initKinect()*

---

C'est dans cette méthode qu'il faudra initialiser tout ce qui concerne Kinect. Elle contient déjà quelques instructions dont certaines ont déjà été vues dans ce tutoriel.

### Récupérer les frames des *datasources*

Dans cette partie, il sera présenté comment utiliser les *data sources* fournies par Kinect.

Atteindre la méthode *initKinect()*

---

Compléter la section «*récupérer les frames de body*» de cette méthode

---

Il faudra remplacer la section suivante dans le code :

```
//***** A COMPLETER *****/  
//***** récupérer les frames de body *****/
```

La première étape consiste à récupérer un *Reader* de la source que l'on souhaite utiliser. Dans ce projet, il sera question d'une *BodyFrameSource*.

Chaque source provenant de Kinect a un *Reader* associé. Par exemple, les

*Reader ColorFrameReader* et *InfraredFrameReader* correspondent aux sources *ColorFrameSource* et *InfraredFrameSource*.

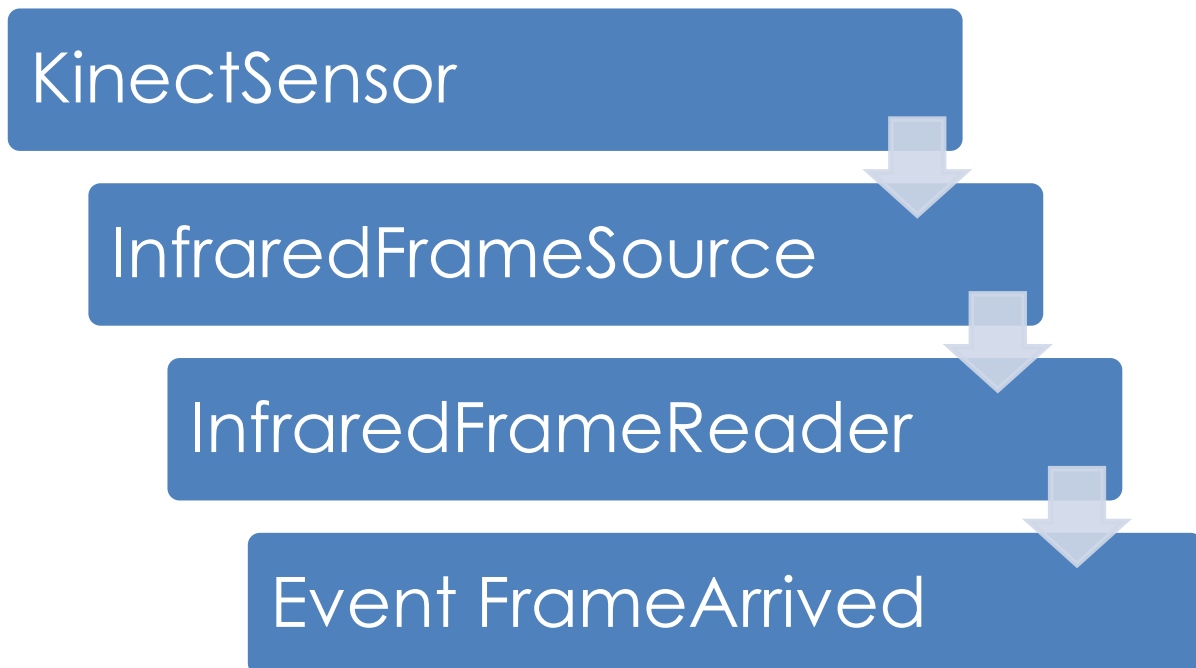


Figure 5.15 : L'évènement *FrameArrived* de la source IR

C'est ce *Reader* qui permet de récupérer les *frames* qui arrivent du capteur à l'aide de l'évènement *FrameArrived*.

Par exemple, voici l'instruction permettant de retourner un *ColorFrameReader* à partir de Kinect :

```
kinectSensor.ColorFrameSource.OpenReader();
```

Il faudra faire de même avec la source de *body* et la stocker localement sous le nom de *bodyReader* (ce *Reader* est déjà déclaré)

Deuxièmement, il faudra suivre l'évènement *FrameArrived* de ce *Reader*, de la même manière que cela a été fait dans le projet *Kinect Draw* et l'évènement *PointerMoved* (chapitre 4.5, page 23).

Une méthode telle que la suivante devrait apparaître dans le code. C'est ici qu'il faudra traiter les *frames* qui arrivent du capteur :

```
void bodyReader_FrameArrived(object sender, BodyFrameArrivedEventArgs e)
{
    throw new NotImplementedException();
}
```

Voici le code minimal qui doit apparaître dans la méthode *bodyReader* :

```
void bodyReader_FrameArrived(object sender, BodyFrameArrivedEventArgs e)
{
    using (BodyFrame frame = e.FrameReference.AcquireFrame())
    {
        if (frame != null)
        {
            //On met à jour ce tableau
            frame.GetAndRefreshBodyData(this.bodies);

            //...
            //Utiliser ici le tableau de body (this.bodies)

        } //FIN - traitement de la frame
    } //FIN - using frame
}
```

Il faut premièrement récupérer la *frame* courante depuis l'argument de la méthode. L'utilisation de la déclaration *using* est ici nécessaire. Elle permet qu'à l'accolade fermante correspondante (*FIN – using frame*), la *frame* soit libérée automatiquement<sup>1</sup>.

Dans un deuxième temps, une fois que la *frame* a été récupéré, il faut vérifier que celle-ci ne soit pas nulle. En effet, cela peut arriver, par exemple, lorsque Kinect démarre.

Dans un 3<sup>ème</sup> temps, on peut mettre à jour le tableau de *body* à l'aide de la méthode *GetAndRefreshBodyData*, tel qu'on l'a vu dans le chapitre 5.5, en page 38.

## Implémenter la détection de mouvement

Atteindre la méthode *initKinect()*

Compléter la section « création d'un *GestureDetector* » de cette méthode

Il faudra remplacer la section ci-dessous dans le code :

```
//***** A COMPLETER *****/
//***** création d'un GestureDetector par personne détectable *****/
//***** gérer les événements du GestureDetector précédemment créé *****/
```

Cette partie consiste à créer un *GestureDetector* par personne détectable, comme vu précédemment dans ce tutoriel, au chapitre 5.5, en page 39. Chaque *GestureDetector* instancié sera ajouté à la liste *this.bodiesDetectors*. Le *body* à l'emplacement 2 de la liste *this.bodies* correspond au *GestureDetector* à l'emplacement 2 de la liste *this.bodiesDetectors*, comme l'indique la Figure 5.13 en page 40.

Lors de l'instanciation d'un *GestureDetector*, on doit indiquer le chemin du fichier de base de données de mouvement *.gbd*. Le fichier *SwitchKinect.gbd* est fourni dans le projet, mais il est tout à fait possible d'en créer un à l'aide des logiciels *KinectStudio* et *Visual Gesture Builder*.

<sup>1</sup> Pour plus d'information sur la déclaration *using*, voir le site de Microsoft concernant celle-ci et les objets *IDisposable* : <https://msdn.microsoft.com/en-us/library/yh598w02.aspx>

Voici un exemple d'instanciation d'un `GestureDetector` :

```
| new GestureDetector(this.originalPath+"\\SwitchKinect.gbd");
```

**Information importante** : l'utilisation du chemin original de l'add-in est ici nécessaire. Ceci est dû au fait que ce projet soit un add-in Office.

Il sera aussi nécessaire de modifier la méthode `FrameArrived` du `bodyReader` pour que lorsque le `TrackingID` d'un `body` change, le `TrackingID` du `GestureDetector` associé soit mis à jour.

### Détecter les mouvements

Chaque `GestureDetector` contient l'évènement `GestureFirstDetected`, qu'on doit suivre et gérer. Lorsqu'un geste est détecté, il convient de :

Vérifier si le PowerPoint est en mode présentateur

On peut utiliser, pour ce faire, la ligne suivante. Elle est vraie si on est en mode présentation :

```
| this.Application.SlideShowWindows.Count > 0
```

Vérifier le nom de la `Gesture`

Le mouvement détecté est accessible via les arguments de l'évènement. Il convient de tester son nom. Par exemple, s'il est égal au nom `OutIn_Right`, effectuer le changement de slide.

Changer de slide

Les méthodes `Next()` et `Previous()` suivantes permettent de changer de slide :

```
| this.Application.SlideShowWindows[1].View.Next();  
| this.Application.SlideShowWindows[1].View.Previous();
```

Il est important de vérifier que le mode présentation est activé avant l'appel de ces méthodes. Dans le cas contraire, une erreur peut survenir.

## 5.8 Correction

Un code est disponible dans l'archive fournie, dans le dossier *KinectPowerPoint-Final*. Il est possible de s'en inspirer pour certains algorithmes. Cependant, ce code n'est pas présenté comme l'unique solution, mais comme proposition. Il est d'ailleurs nettement perfectible.

## 5.9 Le concept de l'ajout du pointeur

Ce chapitre présente comment un pointeur pourrait être ajouté. L'ajout de ce pointeur ne sera pas précisément expliqué, mais une vue d'ensemble du concept de cet ajout sera présente.

Il est possible d'ajouter un pointeur en utilisant le même principe que vu dans le projet *Kinect Draw* : utiliser `KinectCoreWindow`, et suivre l'évènement `PointerMoved`.



## Collaborer

Apportez des modifications simultanément avec d'autres personnes à partir de plusieurs PC et entamez des conversations dotées d'une fonctionnalité d'ajout de commentaires améliorée.

Partager en ligne n'a jamais été aussi simple : même si votre public ne dispose pas de PowerPoint, vous pouvez réaliser votre présentation en ligne, via un navigateur, grâce à la fonctionnalité Présenter en ligne.

Collaborez en même temps à partir d'emplacements différents, que vous utilisiez PowerPoint sur votre ordinateur de bureau ou PowerPoint Web App.

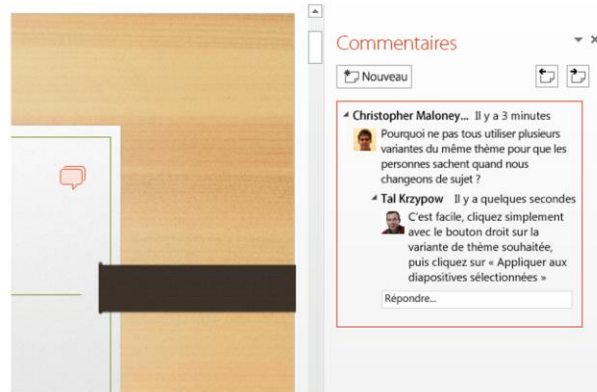


Figure 5.16 : Un pointeur (en jaune) peut être ajouté à ce projet

Plusieurs moyens pourraient être utilisés pour activer le pointeur à l'aide de Kinect. Par exemple, il serait possible que lorsque le joueur tend le bras, le pointeur s'affiche, et que lorsque le bras est ramené vers le joueur, le curseur n'est plus visible. Cette façon de faire a été testée, en utilisant la valeur de *HandReachExtent* d'un *KinectPointerPoint*, tiré de la méthode *PointerMoved* (cf. projet *Kinect Draw*, chapitre 4.5).

Le pointeur déjà implémenté dans PowerPoint ne peut pas être utilisé avec un add-in. L'idée testée qui a été trouvée est la suivante : au lancement du mode présentation et à chaque changement de slide, ajouter une ellipse non visible transparente représentant le pointeur. La déplacer selon un *KinectPointerPoint*, et l'afficher lorsque le bras est tendu.

*Information : des problèmes de threading<sup>1</sup> peuvent se poser. L'usage de l'instruction lock() est peut-être nécessaire, c'est pourquoi, pour implémenter cette fonctionnalité ainsi, il est conseillé d'avoir des notions en développement parallèle. C'est aussi pourquoi l'ajout du pointeur n'est pas explicitement expliqué dans ce tutoriel, le développement pouvant devenir fastidieux.*

<sup>1</sup> Les *threads* permettent d'exécuter plusieurs tâches en parallèles

## 6 Pour aller plus loin...

Ce dernier chapitre du tutoriel présente certains sujets qui permettront d'en apprendre plus sur l'utilisation de l'API Kinect.

Le site *Kinect for Windows* sur le site de Microsoft contient notamment toute la documentation technique de l'API, le SDK, etc. C'est le point d'entrée du développement sur Kinect :

<http://www.microsoft.com/en-us/kinectforwindows/>

On y trouve notamment des vidéos qui présentent l'API de Kinect V2 dans son ensemble :

<http://www.microsoft.com/en-us/kinectforwindows/develop/how-to-videos.aspx>

Certains points de l'API de Kinect n'ont pas été traités dans ce tutoriel. Voici quelques pistes qui permettront d'en découvrir plus :

- Le système d'engagement
- La reconnaissance vocale
- Le scan 3D à l'aide de *Fusion*
- Diverses *datasources* n'ont pas été utilisées, telle que *AudioSource*, *ColorFrameSource*, etc.
- Les logiciels *KinectStudio* et *Visual Gesture Builder*
- L'API de Kinect avec d'autre Framework, tel que Unity
- Les API *face* de Kinect, qui permettent de scanner un visage
- La reconnaissance d'expression des visages
- La région Kinect (*KinectRegion*)
- ...

### 6.1 Ajouter des fonctionnalités aux projets

Il est possible d'ajouter des fonctionnalités aux projets *Kinect Draw* et *Kinect PowerPoint* ; des idées sont décrites dans ce sous-chapitre.

Kinect Draw

Ci-contre, la Figure 6.1 montre certaines fonctionnalités qui pourraient être implémentée :

- Lorsque plusieurs personnes sont présentes, il est toujours possible de dessiner
- On peut choisir la couleur que l'on souhaite en fermant la main
- Il est possible de dessiner de la main droite et de la main gauche

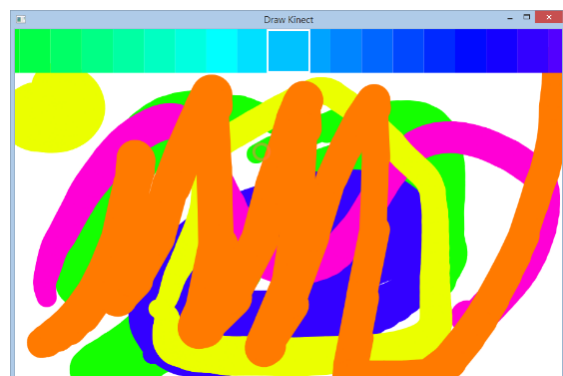


Figure 6.1 : Un « Kinect Draw » amélioré

On peut aussi penser à des fonctionnalités telles que le dessin simultané par plusieurs personnes, pouvoir effacer la feuille à l'aide d'un geste de la main, enregistrer le dessin au format jpeg, etc.

#### Kinect PowerPoint

Il serait possible d'ajouter un ruban à ce projet, indiquant, par exemple, le nombre de personnes détectées, et proposant certaines options à cocher

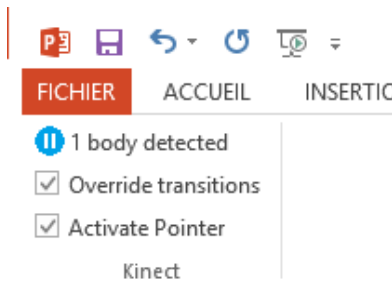


Figure 6.2 : Ajout d'un ruban

Une fonctionnalité pourrait permettre de modifier les transitions de la présentation courante pour mieux correspondre au geste. La transition « poussée » de PowerPoint pourrait être utilisée.

On pourrait aussi imaginer une fonctionnalité qui permettrait de dessiner sur le slide en cours, ou de contrôler l'activation du pointeur à l'aide de la voix.

## 7 Sources

### Sources d'informations utilisées d'internet

<http://www.guinnessworldrecords.com/world-records/fastest-selling-gaming-peripheral>

[http://fr.wikipedia.org/wiki/Interactions\\_homme-machine](http://fr.wikipedia.org/wiki/Interactions_homme-machine)

<http://fr.wikipedia.org/wiki/Kinect>

[http://fr.wikipedia.org/wiki/Interface\\_de\\_programmation](http://fr.wikipedia.org/wiki/Interface_de_programmation)

### Figures

Les figures non citées ci-dessous sont des images sous licence Creative Common, tirées de capture d'écran, des fonctions de dessins et schémas présentes dans Word, ou leur source est déjà mentionnée.

Figure 1.1

<http://www.independent.co.uk/incoming/article8787892.ece/alternates/w460/AN26939219People%20play%20on%20Ki.jpg>

Figure 1.3

<http://smeenk.com/wp-content/uploads/2014/03/kinectheader.png>

Figure 2.2

Tiré du guide d'installation rapide, dans la boîte de l'adaptateur Kinect

Figure 3.1

<http://re.buildinsider.net/small/kinectv2cpp/01/02.gif>

Figure 3.2

Capture d'écran tirée de *Kinect Evolution*, disponible dans *SDK Browser*

Figure 3.3

Capture d'écran tirée de *Kinect Evolution*, disponible dans *SDK Browser*

Figure 3.5

Capture d'écran tirée de *Body Index Basics*, disponible dans *SDK Browser*

Figure 3.6

Capture d'écran tirée de *Kinect Studio*

Figure 3.7

Capture d'écran tirée de *Kinect Studio*

Figure 3.8

Capture d'écran tirée de *Infrared Basics*, disponible dans *SDK Browser*

Figure 4.7

Capture d'écran tirée de *Controls Basics*, disponible dans *SDK Browser*

Figure 4.8

Tiré du PDF *KinectHIG2.0*, disponible sur :

<http://go.microsoft.com/fwlink/?LinkId=403900>

Figure 5.1

Capture d'écran tirée de *PowerPoint*

Figure 5.2

Capture d'écran tirée de *Kinect Studio*

Figure 5.4

<http://www.veryicon.com/icon/png/System/Radium/File.png>

## Remerciements

Je tiens à remercier tout naturellement M. Patrick Chenaux, mon chef de projet, pour le suivi effectué tout au long du TPI, ainsi que pour les conseils et l'aide apportés.

Je souhaite aussi remercier les experts M. Deniz Mutlu et M. Serge Wenger pour leur suivi, les recommandations sur le déroulement du TPI et les indications sur les attentes du rapport de projet.