Tutoriel de Reconnaissance d'Objet
Table PixelSense SUR40

Table des matières

1.Introduction

1.Pré-requis

Logiciels :

- **SDK Microsoft Surface 2.0**
- **Microsoft Visual Studio 2010** (le SDK n'est pas supporté par Visual Studio 2012)

Matériels :

- **Table PixelSense SUR40**
- Code barre (ou tag) de reconnaissance PixelSense, imprimable depuis http://msdn.microsoft.com/en-us/library/ff727841.aspx

Document :

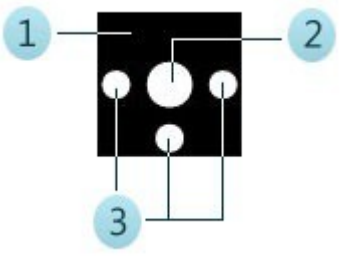
- **Development Tutorial on PixelSense**

2. Reconnaissance d'objets avec la table PixelSense SUR40

La table **PixelSense SUR40** offre la possibilité de reconnaître des tags.

Cette reconnaissance peut servir pour reconnaître des objets : une application peut reconnaître des tags pour reconnaître un objet ou les distinguer parmi une collection d'objets.

Les tags peuvent contenir un byte d'information.



Pour être reconnu, un tag nécessite 3 parties obligatoires. Une

surface non-réfléchissante noire (1), un cercle central réfléchissant (2) permettant de localiser le tag sur la surface de la table et trois cercles sur les côtés (3) permettant de détecter l'orientation de l'objet sur la table.

En plus de ces points obligatoires de reconnaissance, les tags comportent des cercles réfléchissants dans les coins représentant les bits. En exemple, voilà les tags pour les valeurs **0xC1**, **0xC2** et **0xC3**.

Le **SDK Microsoft Surface 2.0** comporte une **API** permettant d'exploiter ces tags.

Note sécurité : ces tags ne doivent pas être utilisés comme moyen d'authentification. Leur nombre restreint (256) cause des problèmes de fiabilité.

2. Présentation de l'API

Les tags de reconnaissance sont représentés au niveau du code par la structure **TagData** dans le **Core Layer** et dans le **Presentation Layer.** Cette structure comporte 4 éléments :

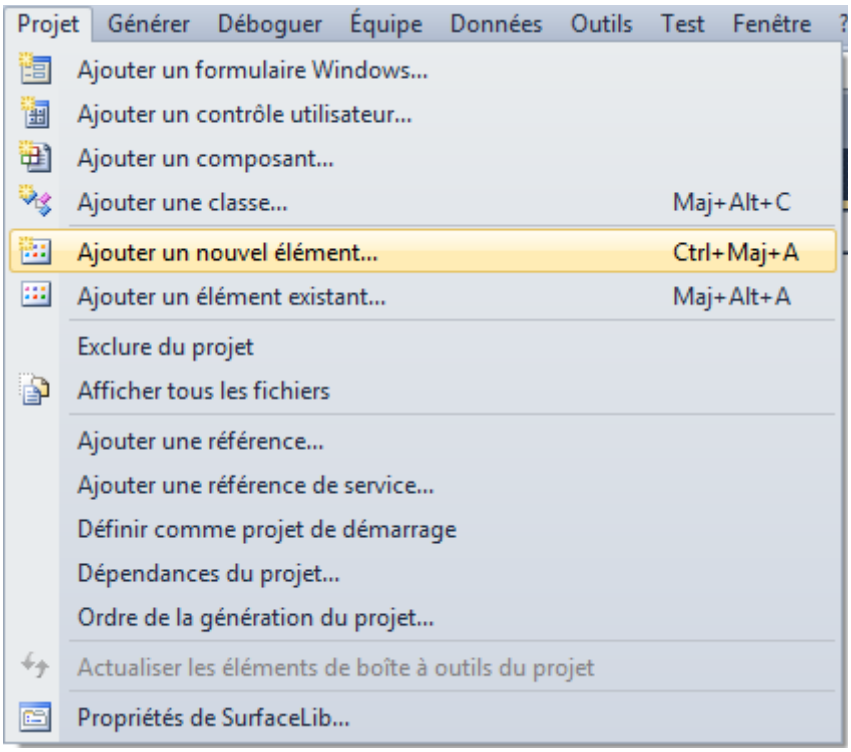
Value	Représente la valeur du Tag. Actuellement, est un entier compris entre 0 et 256
Series	Indique comment les valeurs Schema et ExtendedValue doivent être interprétées. Actuellement, la seule valeur supportée est 0
Schema	Le schema du Tag. Actuellement, la seule valeur supportée est 0
ExtendedValue	La valeur étendue du Tag. Actuellement, la seule valeur supportée est

3. Utilisation pratique de la reconnaissance de Tag

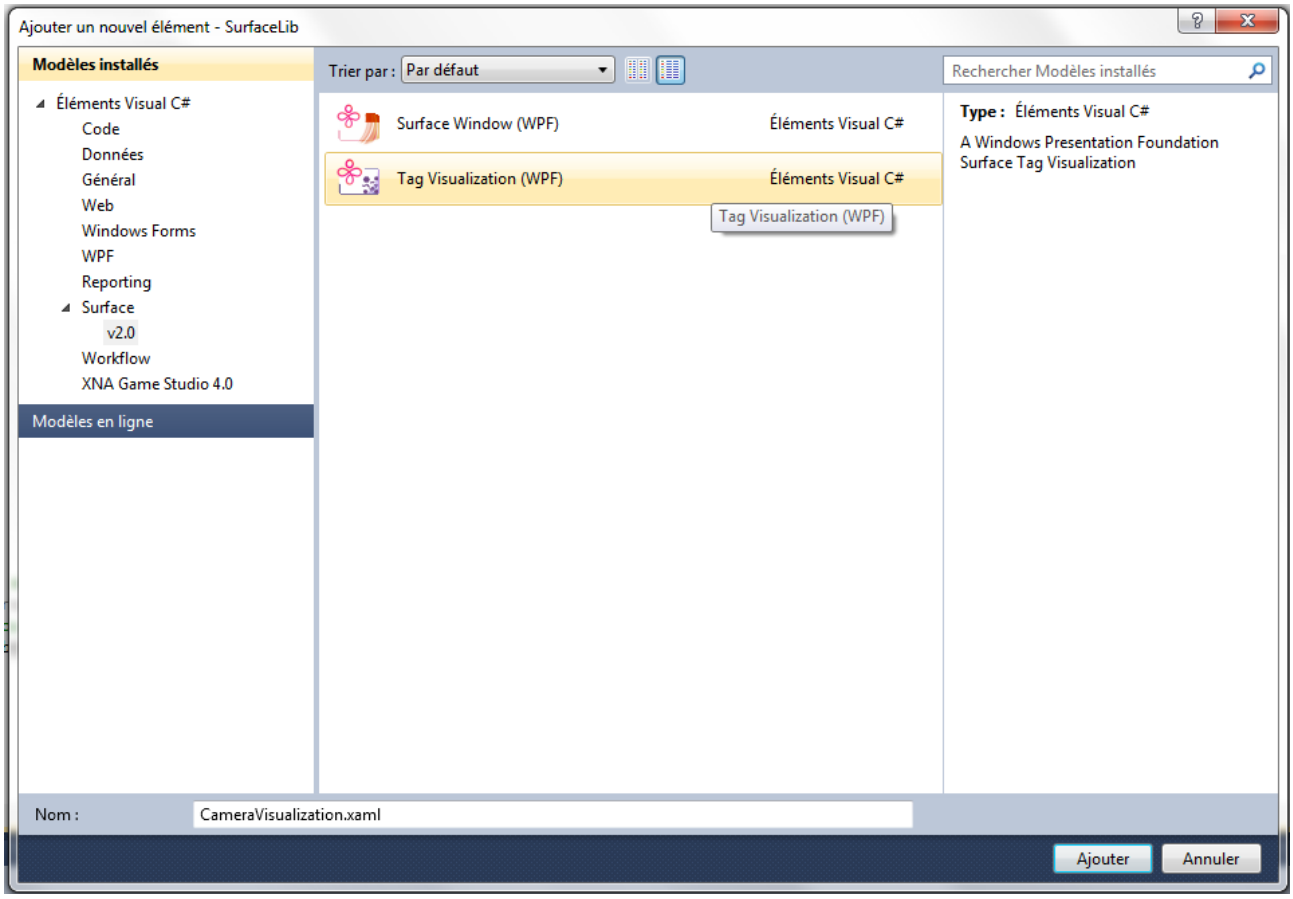
1. Création de la TagVisualization

La **TagVisualization** est la forme qui va apparaître lorsque le tag sera posé sur la table.

Pour créer cette **TagVisualization**, il faut se rendre dans le menu **Projet** et cliquer sur **Ajouter un nouvel élément**.



Dans la liste des catégories, choisir **Surface** puis **v2.0**. Dans la liste des templates choir **TagVisualization (WPF)**. Entrer **CameraVisualization.xaml** comme nom de la **TagVisualisation** et cliquer sur **Ajouter.**



Nous allons ensuite modifier son fichier **xaml** comme suit :

CameraVisualization.xaml

```
<s:TagVisualization x:Class="SurfaceApplication2.CameraVisualization"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:s="http://schemas.microsoft.com/surface/2008"
    Loaded="CameraVisualization_Loaded"
    Height="89" Width="89">
    <Canvas>
        <Rectangle Name="myRectangle"
                Height="89"
                Width="89"/>

        <Label Name="CameraModel"
               Content=" "
               FontSize="12"
               Canvas.Top="65"
               Canvas.Left="0"/>

    </Canvas>
</s:TagVisualization>
```

Ici, notre **TagVisualization** sera donc un rectangle de taille 89*89 avec un label en bas de celui-ci qui pour l'instant n'a pas de valeur.

La ligne `Loaded="CameraVisualization_Loaded"` définit la fonction à appeler dans le fichier **C#** associé pour customiser la **TagVisualization** programmatiquement.

Voyons maintenant à quoi ressemble la classe **CameraVisualization**.

CameraVisualization.xaml.cs

```
using System;using System.Collections.Generic;using System.Text;using System.Windows;
using System.Windows.Controls;using System.Windows.Data;using System.Windows.Documents;
using System.Windows.Input;using System.Windows.Media;using System.Windows.Media.Imaging;
using System.Windows.Navigation;using System.Windows.Shapes;using Microsoft.Surface;
using Microsoft.Surface.Presentation;using Microsoft.Surface.Presentation.Controls;
using Microsoft.Surface.Presentation.Input;

namespace SurfaceApplication2
{
    public partial class CameraVisualization : TagVisualization
    {
        public CameraVisualization()
        {
            InitializeComponent();
        }
        private void CameraVisualization_Loaded(object sender, RoutedEventArgs e)
        {
            //TODO: customize CameraVisualization's UI based on this.VisualizedTag here
        }
    }
}
```

2. Utilisation de la TagVisualization

```
<s:TagVisualizer
    Name="MyTagVisualizer"
    VerticalAlignment="Stretch"
    HorizontalAlignment="Stretch"
    Background="Transparent"
    Height="720" Width="720"
    VisualizationAdded="OnVisualizationAdded"
    VisualizationRemoved="OnVisualizationRemoved"
    >
    <s:TagVisualizer.Definitions>
       <s:TagVisualizationDefinition Value="0x1"
           Source="CameraVisualization.xaml"
           LostTagTimeout="100"
           MaxCount="8"
           OrientationOffsetFromTag="0"
           PhysicalCenterOffsetFromTag="0.01,0.01"
           TagRemovedBehavior="Fade"
           UsesTagOrientation="True"
           />
     </s:TagVisualizer.Definitions>
 </s:TagVisualizer>
```

Voici comment on définit une **TagVisualization** dans une application. Dans son fichier **xaml**, on va rajouter une balise `<s:TagVisualizer >`, c'est ici qu'on va définir la zone dans laquelle les tags seront reconnus. Ici cette zone fait 720*720 et elle définit deux fonctions : **OnVisualizationAdded** et **OnVisualizationRemoved**.

`VisualizationAdded="Fonction"` définit la fonction qui sera appelée lorsque qu'un tag sera posé sur la table.

`VisualizationRemoved="Fonction"` définit la fonction qui sera appelée lorsque qu'un tag est retiré de la table.

Il existe aussi `VisualizationMoved="Fonction"` qui définit la fonction qui sera appelée lorsqu'un tag est déplacé ou lorsqu'il change d'orientation.

Une fois la zone de reconnaissance créée, il va falloir définir quels tags le programme va reconnaître. Pour cela, on créé la balise `<s:TagVisualizer.Definitions/>` qui est un tableau de `<s:TagVisualizationDefinition/>`. Celle-ci contient beaucoup de caractéristiques :

- `Value` correspond a la valeur du tag (de **0x00** à **0xFF**).
- `Source` correspond au fichier **xaml** qui définit la **TagVisualization**.
- `LostTagTimeout` correspond au temps (en ms) après quoi on considère le tag retiré.
- `MaxCount` est le nombre maximum de tag correspondant à cette valeur de tag.
- `OrientationOffsetFromTag` rotation applicable à la **TagVisualization**(de 0° à 360°).
- `PhysicalCenterOffsetFromTag` offset en x et y de la **TagVisualization** par rapport au tag.
- `UsesTagOrientation` vrai si on souhaite que la **TagVisualization** tourne en même temps que le tag.

On peut également définir un tag programmatiquement, voici le code :

```
TagVisualizationDefinition tagDef=new TagVisualizationDefinition();
tagDef.Value = 1;
tagDef.Source = new Uri("CameraVisualization.xaml", UriKind.Relative);
tagDef.MaxCount = 4;
tagDef.LostTagTimeout = 100.0;
tagDef.OrientationOffsetFromTag = 0.0;
tagDef.PhysicalCenterOffsetFromTag = new Vector(0.01, 0.01);
tagDef.TagRemovedBehavior = TagRemovedBehavior.Fade;
tagDef.UsesTagOrientation = true;
MyTagVisualizer.Definitions.Add(tagDef);
```

Et enfin, les fonctions définies précèdemment par `VisualizationAdded="OnVisualizationAdded"` et `VisualizationAdded="OnVisualizationRemoved"`

```
private void OnVisualizationRemoved(object sender, TagVisualizerEventArgs e)
       {
           CameraVisualization camera = (CameraVisualization)e.TagVisualization;
           //code
        }
private void OnVisualizationAdded(object sender, TagVisualizerEventArgs e)
```

```
        {
            CameraVisualization camera = (CameraVisualization)e.TagVisualization;
            //code
        }
```

La première ligne dans chaque fonction permet de récupérer une **CameraVisualization**, avec laquelle on va pouvoir interagir via la **TagVisualization**. On peut par exemple changer le texte du label qui est contenu dans la **TagVisualization** ou en changeant la couleur de celle-ci par exemple.

```
            camera.CameraModel.Content = "Exemple_Content";
            camera.myRectangle.Fill = SurfaceColors.Accent4Brush;
```

4. Bonnes pratiques de l'utilisation des Tags

1. Bonnes pratiques matérielles

La table Surface doit être utilisée dans un milieu avec peu de lumière ou de la lumière indirecte afin de ne pas troubler le système de reconnaissance avec des ombres distinctes et des points lumineux.

Les Tags doivent avoir un noir qui soit réellement noir afin de bien absorber les infra-rouges. Le matériau utilisé pour le Tag ne doit pas être trop réfléchissant, afin d'éviter un effet d'éblouissement autour des points blancs.

Les Tags doivent être de bonne qualité, un tag imprimé s'use rapidement et le système de détection peut avoir des soucis avec des tags usés. Dans le cas de tags commandés pour usage avec une table Surface, précisez l'utilisation d'une table Surface 2.0.

Les Tags doivent être placés sur des objets physiques de couleur noir ou non-réfléchissante aux infra-rouges pour obtenir la meilleure détection des tags pour les petits objets, comme des pièces de jeu. Il est important de ne pas avoir d'éclat de couleur clair autour du tag. Sur des objets plus imposants, étendre la zone noir du tag permet d'avoir de meilleures performances.

Les Tags placés sur des objets de couleur clair ou blanche ne doivent pas être centrés. De plus, il vaut mieux étendre la zone noire autour du Tag.

Les Tags placés sur des zones non plates ou mal équilibrées peuvent créer des problèmes à la lecture sur la table.

2. Bonnes pratiques logicielles

Pour une reconnaissance optimale, il vaut mieux avoir une zone de reconnaissance des

objets de couleur sombre et sans composante rouge. Cela permet d'éviter, en cas de reflet, d'avoir une nouvelle détection.

Dans des conditions peu idéales, un tag peut apparaître et réapparaître brusquement de manière fréquente. Ceci s'explique par le fait que le système perd et regagne le contact avec un objet marqué en déplacement. Dans le cadre d'un objet cachant des informations, pour un jeu multijoueur par exemple, il est préférable que le tag disparaisse rapidement, si l'objet est renversé par exemple. Dans des conditions peu idéales, il peut toutefois être préférable d'avoir un délai avant de perdre la reconnaissance de l'objet, afin d'éviter le phénomène d'apparitions et de disparitions fréquentes.

Il est également important de repérer les objets marqués par la valeur de leur tag et non par l'identifiant de l'objet (propriété `Id`).

Ne pas utiliser deux fois la même valeur de tag, ou ne pas faire placer de manière proche deux tags avec la même valeur. En effet, le système pourrait intervertir les deux identifiants.

Références

Caractéristiques de la table :

http://www.samsung.com/au/business/business-products/smart-signage/specialised-solutions/LH40SFWTGC/XY

http://www.solatys.fr/upload/9f32e151-de3d-44ff-ad1b-a903e5639ffc.pdf

La documentation du SDK Microsoft Surface 2.0 en ligne est très bien fournie :

http://msdn.microsoft.com/en-us/library/ff727815.aspx

Reconnaissance des tags :

http://msdn.microsoft.com/en-us/library/ff727854.aspx

http://msdn.microsoft.com/en-us/library/ff727919.aspx

Pour imprimer les tags :

http://msdn.microsoft.com/en-us/library/ff727841.aspx