Kinect Interactions – allow developers to create interactive applications with a new, well-designed and ergonomically friendlier interaction language than before. The Kinect SDK can now detect a "push towards the screen" gesture to activate buttons, and this is the recommended "activate" gesture now instead of the old "hover, wait until the circle fills and its pressed" gesture. The latest K4W SDK solves this issue by differentiating between an open and a closed hand. When you close your hand, you "grip" whatever's underneath it (such as a scroll viewer), and then you can move your hand to scroll. If you release the scroller while your hand is still moving, the scrolling keeps happening – similarly to the inertia scroll you are already used to on touch devices. You can close your hand again at any time, and stop the inertia – again, as you expected.

Software's

- Kinect SDK 1.7
- Kinect for Windows Developer Toolkit

Initializing the Kinect Sensor with KinectSensorChooser

The control *KinectSensorChooserUI* from the *Microsoft.Kinect.Toolkit.Controls* namespace indicates the status of the Attached Kinect Sensor, and can inform the user if the sensor is unplugged, plugged into the wrong USB port, etc.

1. In the MainView.xaml, add the following namespace and add the KinectSensorChooserUI control to the main grid:

```
xmlns:k=http://schemas.microsoft.com/kinect/2013
<Grid>
   <k:KinectSensorChooserUI HorizontalAlignment="Center" VerticalAlignment="Top"
Name="sensorChooserUi" />
</Grid>
```

2. We also need to initialize the KinectSensorChooser from the code behind file
Add a private field to the MainWindow code behind file.

```
private KinectSensorChooser sensorChooser;
```

3. Set up an event handler for the OnLoaded event in the MainWindow constructor (or in XAML, as you wish):

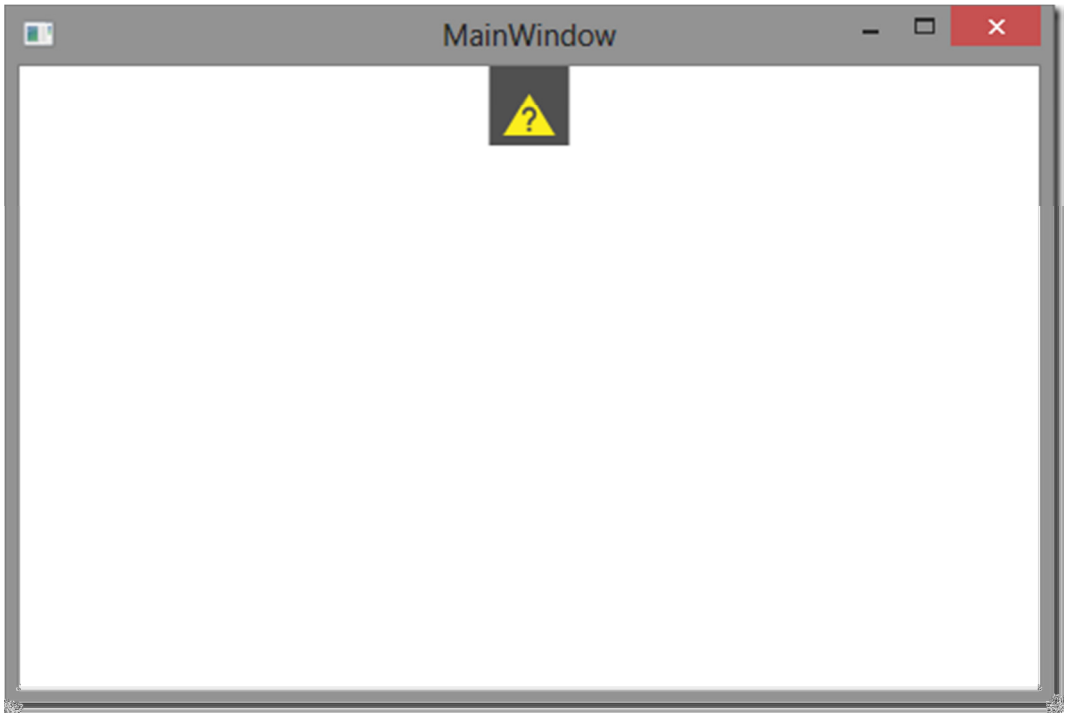
```
public MainWindow()
{
   InitializeComponent();
   Loaded += OnLoaded;
}
```

4. create the OnLoaded event handler:

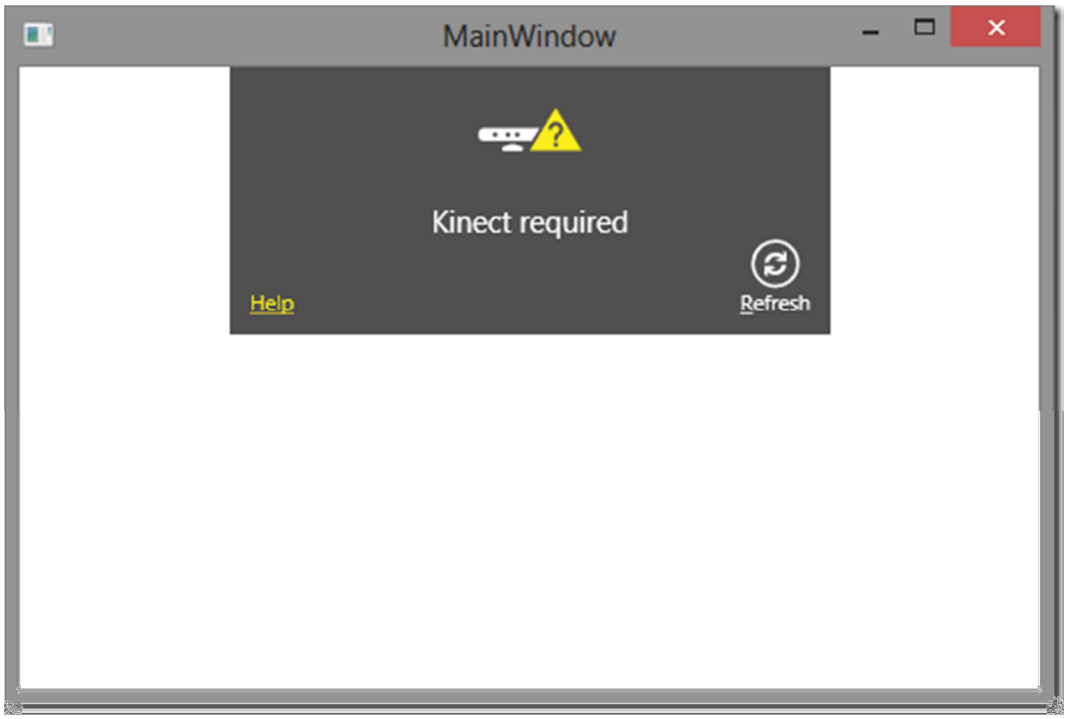
```
private void OnLoaded(object sender, RoutedEventArgs routedEventArgs)
{
    this.sensorChooser = new KinectSensorChooser();
    this.sensorChooser.KinectChanged += SensorChooserOnKinectChanged;
    this.sensorChooserUi.KinectSensorChooser = this.sensorChooser;
    this.sensorChooser.Start();
}
```

If a sensor is chosen or got initialized, the code above will invoke the SensorChooserOnKinectChanged event handler.

If there are no Kinects connected to the computer:



Kinect required



Once the Kinect is connected, it initializes it.

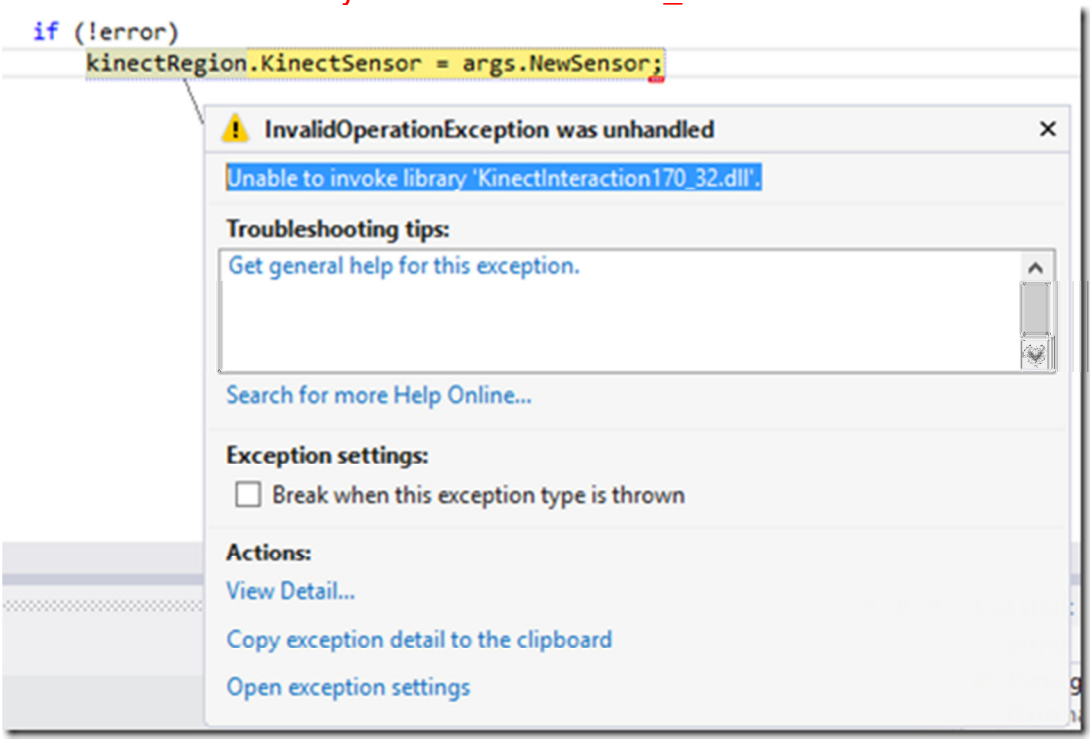
The Kinect Region

KinectRegion is the key element of Kinect Interactions on WPF: it is the screen area where interactive elements are placed and can be manipulated. KinectRegion is also responsible for displaying and moving the hand cursor. An application can have multiple KinectRegions, but they cannot be nested. Each KinectRegion can have its own respective Kinect sensor.

```
<k:KinectRegion Name="kinectRegion">
</k:KinectRegion>
```

Error message:

"Unable to invoke library 'KinectInteraction170_32.dll'."



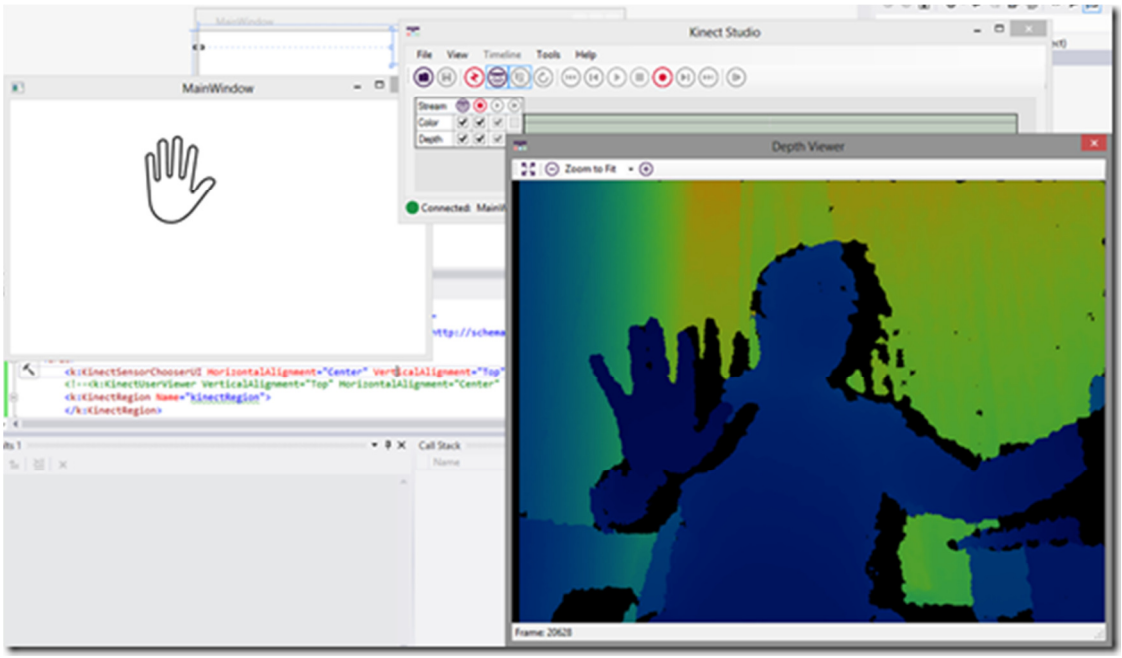
Solution:

This is a common problem with Kinect. The dll mentioned can be a bit tricky to find. The simplest way is to install the Controls Basics – WPF source code from the Developer Toolkit Browser (press the Install button) – or just download the sample project for this article.

In the AnyCPU.Debug folder of the sample's installation directory, copy the KinectInteraction170_32.dll and KinectInteraction170_64.dll files to the project's output directory.

Note: You may have to wait as much as 20 seconds for the Kinect Sensor to initialize. During this process, your application will not respond and will show a wait cursor if you move the mouse over it.

Better hand recognition using Kinect:



User Viewer

The KinectUserViewer displays the depth pixels that belong to an identified user or users. Use the KinectUserViewer control to make sure that the Kinect SDK "sees" you as a human and is able to separate you from the surroundings. If you see nothing in the KinectUserViewer (after the Kinect initialization has finished), it means that the Kinect is not tracking you, it hasn't identified you as a person yet. Try to move around, and remember that you need to be at least a meter (or even more) away from the sensor for it to detect you.

```
<k:KinectUserViewer VerticalAlignment="Top" HorizontalAlignment="Center"
k:KinectRegion.KinectRegion="{Binding ElementName=kinectRegion}" Height="100" />
```

Screenshots:

Interactive Controls

KinectTileButton:
The KinectTileButton is one of the simplest controls available. It resembles a Windows 8-style tile with a template-able label and a Content that is just like the content of the regular Button control.

In this project, we are using this to display the slides on the bottom of the screen.
Below is the source code to declare the tile buttons dynamically and bind it to the slide show.

```
       for (int i = 1; i < 10; i++)
            {
                var button = new KinectTileButton
                {
                    Content = i,
                    Height = 100,
                };
                int i1 = i;
                button.Click += OnClicked;

                scrollContent.Children.Add(button);
            }


       private void OnClicked(object sender, RoutedEventArgs
routedEventArgs)
        {

            {
                Slideshow.DataContext = "";
                BitmapImage bi3 = new BitmapImage();
                bi3.BeginInit();
                bi3.UriSource = new Uri("Images\\Slide"+
routedEventArgs.Source.ToString().Substring(52,1) +".bmp",
UriKind.Relative);
                bi3.CreateOptions =
BitmapCreateOptions.IgnoreColorProfile;
                bi3.EndInit();
                Picture.Stretch = Stretch.Fill;
                Picture.Source = bi3;
            };
        }
```

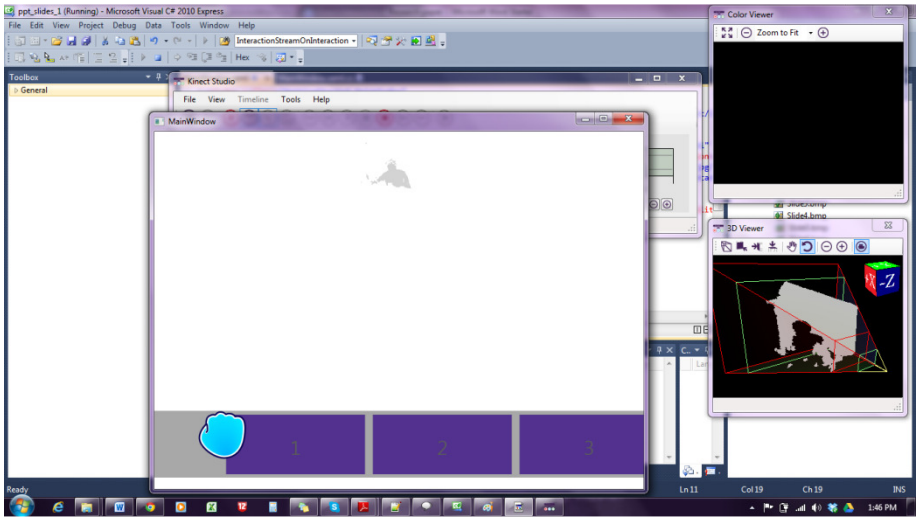
KinectScrollViewer:

This looks exactly like a standard WPF ScrollViewer. It even has a StackPanel called “scrollContent”, and these two controls are set up to scroll horizontally

```
<k:KinectScrollViewer VerticalScrollBarVisibility="Disabled"
HorizontalScrollBarVisibility="Auto" VerticalAlignment="Bottom">
   <StackPanel Orientation="Horizontal" Name="scrollContent" />
</k:KinectScrollViewer>
```

Now you can move your hand, and the KinectScrollViewer will “stick to it”, much like when you scroll a list on a touch device, such as a phone or a tablet. If you open your hand, you will release the KinectScrollViewer. If you push your hand forward, you can select easily one of the slides.

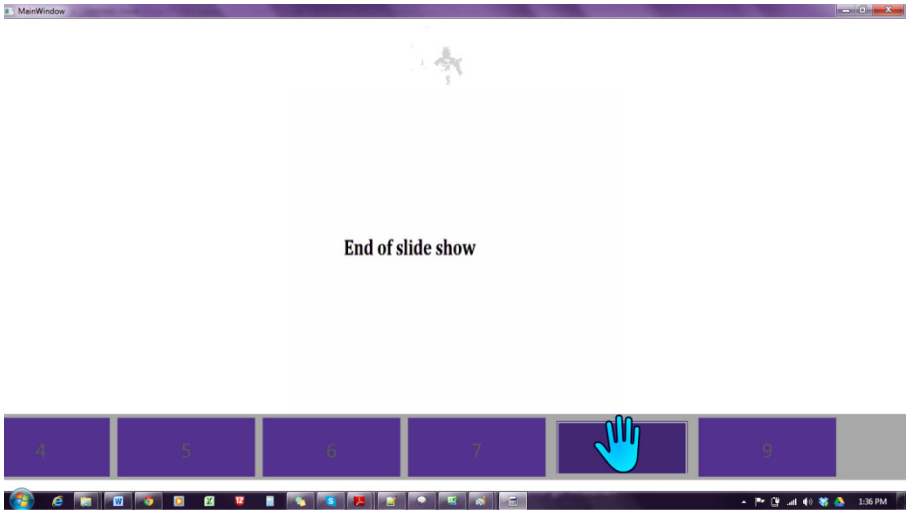
Screenshots:



Slide select using hand press action

Once the hand press is recognized the hand color changes

Slides corresponding to the Tile button appears after the click



References:

- Kinect SDK and developer for windows downloads: http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx
- http://en.wikipedia.org/wiki/Kinect
- Kinect SDK 1.7 features: http://www.microsoft.com/en-us/kinectforwindows/develop/new.aspx
- http://www.soulsolutions.com.au/Blog/tabid/73/EntryId/853/Kinect-For-Windows-Interactions-Gallery-ndash-KinectRegion.aspx