Kinect Development Workshop – Basic Tutorial

How to create a basic Kinect application to control arrow keys using gestures

This tutorial runs through how to make a very basic Kinect application using the Microsoft Kinect SDK. The application lets you control the left and right arrows keys using hand gestures. It could be used to navigate a powerpoint presentation. It was created for a workshop run on the 14th of October, 2011.

Note: The final version in the video looks a bit dodgy, the end result is actually much better.

Jump to:

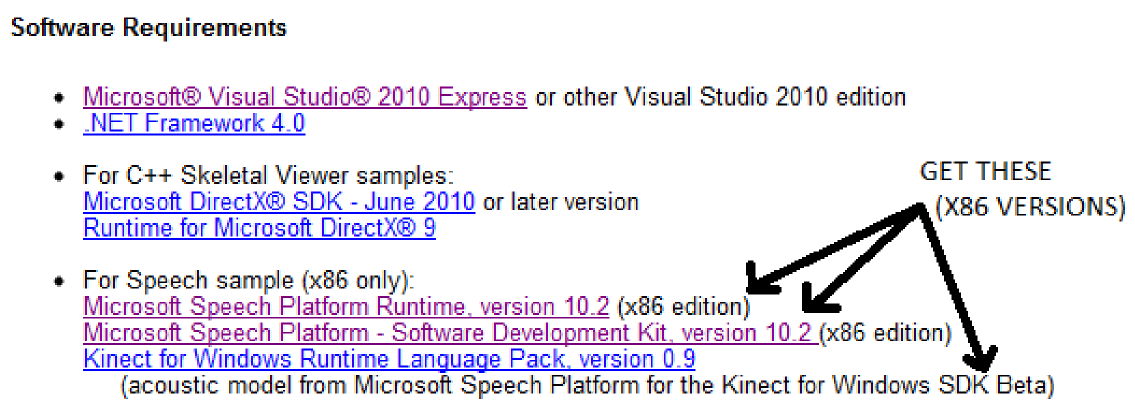
Get the final source code here.

Video Tutorial

Set up/Resources required:

1. Visual Studio 2010 C# Express - free from here: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/express-iso
   OR if you are a student you can get Visual Studio 2010 Ultimate from www.dreamspark.com

2. Kinect SDK: http://research.microsoft.com/en-us/um/redmond/projects/kinectsdk/download.aspx
3. Coding4Fun: http://c4fkinect.codeplex.com/
   Note: This is not required for normal kinect stuff, but is used in our tutorial for simplicity.
4. For speech, go to the readme, and under system requirements, get the3 speech samples.
   Note: Speech is not used in our tutorial, but is handy to have for future projects
   http://research.microsoft.com/en-us/um/redmond/projects/kinectsdk/docs/readme.htm



Software Requirements

- Microsoft® Visual Studio® 2010 Express or other Visual Studio 2010 edition
- .NET Framework 4.0

- For C++ Skeletal Viewer samples:
  Microsoft DirectX® SDK - June 2010 or later version
  Runtime for Microsoft DirectX® 9

- For Speech sample (x86 only):
  Microsoft Speech Platform Runtime, version 10.2 (x86 edition)
  Microsoft Speech Platform - Software Development Kit, version 10.2 (x86 edition)
  Kinect for Windows Runtime Language Pack, version 0.9
  (acoustic model from Microsoft Speech Platform for the Kinect for Windows SDK Beta)

Instructions

Setting up the Development environment

Step 1 - Creation

Create a new Windows Presentation Foundation (WPF) application. (Optional: Give it a name)

Step 2 - References

Add references to the following:

Microsoft.Research.Kinect (Found under .NET)

Coding4Fun.Kinect.Wpf (Download here: http://c4fkinect.codeplex.com/, navigate to using 'browse')

System.Windows.Forms (Found under .NET)

Step 3 - Display

Bring up the toolbox (ctrl + alt + x) and drag a canvas on to the screen.

Inside the canvas, add an image. *(Optional: set width and height to 640 x 480 in the properties)*

Inside the canvas, add 3 ellipses. Give them each a name e.g. headCircle, leftCircle, rightCircle. *(Optional: make them look like circles by setting width/height both to 40, and set fill so it stands out)*

Make sure the image and the ellipses are inside the canvas.

e.g. Your XAML code should look like this :

```
    <Canvas Name="canvas1">
        <Image Canvas.Left="0" Canvas.Top="0" Height="480" Name="image1"
Stretch="Fill" Width="640" />
        <Ellipse Canvas.Left="320" Canvas.Top="138" Height="40" Name="headCircle"
Stroke="Black" Width="40" Fill="Red" />
```

```
        <Ellipse Canvas.Left="10" Canvas.Top="10" Fill="Red" Height="40"
Name="rightCircle" Stroke="Black" Width="40" />
        <Ellipse Canvas.Left="71" Canvas.Top="68" Fill="Red" Height="40"
Name="leftCircle" Stroke="Black" Width="40" />

    </Canvas>
```

Step 4. References (again)

Inside the .cs now, you want to add:

```
using Microsoft.Research.Kinect.Nui;
using Coding4Fun.Kinect.Wpf;
```

Step 5. Loaded and Closed events

In your main window, go to properties -> Events tab -> Double click on ‘loaded’ and on ‘closed’ to add two events to your code.

Now create a Runtime environment outside of these events. In our example we call it ‘nui’.

```
Runtime nui = new Runtime();
```

Within the window_loaded event, we want to initialise the runtime environment and since we want camera and skeletal tracking we give it the following options.

```
nui.Initialize(RuntimeOptions.UseColor | RuntimeOptions.UseSkeletalTracking);
```

Under window_closed, we need to uninitialise the environment:

```
nui.Uninitialize();
```

Getting the RGB camera to display

Step 6. Create an event for the camera

Inside the window_loaded :

```
nui.VideoFrameReady += new
EventHandler<ImageFrameReadyEventArgs>(nui_VideoFrameReady);
```

Inside the event:

```
void nui_VideoFrameReady(object sender, ImageFrameReadyEventArgs e)
        {
       //This uses the Coding4Fun library
            image1.Source = e.ImageFrame.ToBitmapSource();
        }
```

Step 7. Open the video stream

Inside the window_loaded :

```
    nui.VideoStream.Open(ImageStreamType.Video, 2, ImageResolution.Resolution640x480,
ImageType.Color);
```

If you run the program, it should now display a video stream from your kinect.

Display the skeleton tracking

Step 8. Creating skeleton event

In the window_loaded, get it to fire off an event for the skeletons:

```
       nui.SkeletonFrameReady += new
EventHandler<SkeletonFrameReadyEventArgs>(nui_SkeletonFrameReady);

       void nui_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e) { }
```

Step 9. Get the first skeleton

The skeletonframe has all the skeletons, the kinect can see up to 4, but is only tracking 2 skeletons at a time. We want to get the first of these.

```
       SkeletonFrame allSkeletons = e.SkeletonFrame;

       //Get the first skeleton
       SkeletonData skeleton = (from s in allSkeletons.Skeletons
                                   where s.TrackingState ==
SkeletonTrackingState.Tracked
                                   select s).FirstOrDefault();
```

Step 10. Move the circles

We identify joints, using JointIDs within the skeletons array of Joints.

Here we call the method setCirclePosition which we need to define.

```
            if (skeleton != null)
            {

             setCirclePosition(headCircle, skeleton.Joints[JointID.Head]);
             setCirclePosition(leftCircle, skeleton.Joints[JointID.HandLeft]);
             setCirclePosition(rightCircle, skeleton.Joints[JointID.HandRight]);
            }
```

Step 11. setCirclePosition

The kinects x and y positions need to be scaled to fit our image and then the circles moved to the right position. Each joint has a position vector <x,y ,z >

```
        void setCirclePosition(FrameworkElement circle, Joint joint)
        {
            var scaledJoint = joint.ScaleTo(640,480,0.5,0.5);
            Canvas.SetLeft(circle, scaledJoint.Position.X);
            Canvas.SetTop(circle, scaledJoint.Position.Y);
        }
```

If you run the program, it should now display a video stream from your kinect and the circles should now be tracking your hand and head.

Controlling actions

Now we want to get program to do something when we move our hands around.

Step 12. Process Gestures

We create a method to process these gestures, since we are only tracking the hand and two hands in this case, we only need them as input:

```
        void processGestures(Joint head, Joint left, Joint right){ }
```

Inside the skeletonframeready method from step 8, we need to call this method (after we called the setCircle methods)

```
       processGestures(skeleton.Joints[JointID.Head],
       skeleton.Joints[JointID.HandLeft], skeleton.Joints[JointID.HandRight]);
```

Step 13. Processing Gestures

The way we process the gestures in our simple application is by looking at the relative distance between each hand and the head. If the hands reach further away from the head, then we will assume that is the gesture. We then send the arrow key commands to the system.

```
      void processGestures(Joint head, Joint left, Joint right){
            var minDistance = 0.5; //how far away we need to reach

            //if the left hand is more than a certain distance to the left of the head
            if (head.Position.X > (left.Position.X + minDistance))
            {
                    System.Windows.Forms.SendKeys.SendWait("{Left}");
            }
            //if the right hand is more than a certain distance to the right of the
head
            if (head.Position.X < (right.Position.X - minDistance))
            {
                    System.Windows.Forms.SendKeys.SendWait("{Right}");
            }
        }
```

Optional Extras

Controlling the kinect motor

The kinect motor isn't meant to be used often, but sometimes when you first set it up it may be facing the wrong way. You can control its elevation angle using the following function (its range goes from -27 to 27 degrees):

```
            nui.NuiCamera.ElevationAngle = 0;
```

Smoothing the skeletons:

To smooth the skeletons, this can be added to Window_Loaded.

```
            #region TransformSmooth
            nui.SkeletonEngine.TransformSmooth = true;
            var parameters = new TransformSmoothParameters
            {
                Smoothing = 0.75f,
                Correction = 0.0f,
                Prediction = 0.0f,
                JitterRadius = 0.0f,
                MaxDeviationRadius = 0.04f
            };
            nui.SkeletonEngine.SmoothParameters = parameters;
            #endregion
```

Better image tracking:

You can get better hand tracking using a depth image.

```
    void setCirclePosition(FrameworkElement circle, Joint joint)
    {
        float x, y;
        nui.SkeletonEngine.SkeletonToDepthImage(joint.Position, out x, out y);

        Canvas.SetLeft(circle, x * 640 - circle.ActualWidth / 2);
        Canvas.SetTop(circle, y * 480 - circle.ActualHeight / 2);

    }
```

Troubleshooting

I want to control power points, but it sends too many arrow keys.

Our basic implementation sends arrow keys to the system as long as your hand is a certain distance from the head. However, if you want to control powerpoints, you only want it to send the key once per gesture. This can be fixed using a Boolean check. Change processGestures to:

```
    bool LeftActivated = false;
    bool RightActivated = false;
    void processGestures(Joint head, Joint left, Joint right){
        var minDistance = 0.5; //how far away we need to reach

        //if the left hand is more than a certain distance to the left of the head
        if (head.Position.X > (left.Position.X + minDistance))
        {
            if (!LeftActivated)
            {
                System.Windows.Forms.SendKeys.SendWait("{Left}");
                LeftActivated = true;
            }
        }
        else
        {
            LeftActivated = false;
        }
        //if the right hand is more than a certain distance to the right of the
head
        if (head.Position.X < (right.Position.X - minDistance))
        {
            if (!RightActivated)
            {
                System.Windows.Forms.SendKeys.SendWait("{Right}");
                RightActivated = true;
            }
        }
        else
        {
                RightActivated = false;
            }
}
```

HELP I get a XAMLParseException the first time I try to run my program.

Try restart your computer. (It actually works!)

Helpful Links

Video tutorials/quick start guide: http://channel9.msdn.com/Series/KinectSDKQuickstarts
Better version of kinect-powered powerpoint: http://kinectpowerpoint.codeplex.com/
Walkthroughs: http://research.microsoft.com/en-us/um/redmond/projects/kinectsdk/guides.aspx