Module III – Programming with Kinect

Game Programming & Robotics

Unit # 13

Getting started with Kinect

- Installing Kinect
  http://www.codeproject.com/Articles/148251/How-to-Successfully-Install-Kinect-on-Windows-Open
- Kinect for Windows Quickstart Series
  http://channel9.msdn.com/Series/KinectQuickstart

Kinect for Windows SDK

Kinect for Windows v 1

- Kinect for windows SDK v1 was released for Windows 7 on June 16, 2011 in 12 countries.[] The SDK includes Windows 7 compatible PC drivers for Kinect device. It provides Kinect capabilities to developers to build applications with C++, C#, or Visual Basic by using Microsoft Visual Studio 2010 and includes following features:
  - Raw sensor streams: Access to low-level streams from the depth sensor, color camera sensor, and four-element microphone array.
  - Skeletal tracking: The capability to track the skeleton image of one or two people moving within the Kinect field of view for gesture-driven applications.
  - Advanced audio capabilities: Audio processing capabilities include sophisticated acoustic noise suppression and echo cancellation, beam formation to identify the current sound source, and integration with the Windows speech recognition API.
  - Sample code and Documentation.

Kinect for Windows 1.5

- In March 2012, Microsoft announced that next version of the Kinect for Windows will be available in May 2012. Kinect for Windows 1.5 will add new features, support for many new languages and would debut in 19 more countries.
- The Kinect for Windows 1.5 SDK woud include
  - 'Kinect Studio' a new app that allows developers to record, playback, and debug clips of users interacting with applications.
  - Support for new "seated" or "10-joint" skeletal system that will let apps track the head, neck, and arms of a Kinect user - whether they're sitting down or standing; which would work in default and near mode.
  - Support for four new languages for speech recognition – French, Spanish, Italian, and Japanese. Additionally it would add support for regional dialects of these languages along with English.
- It would be available in Hong Kong, South Korea, and Taiwan in May and Austria, Belgium, Brazil, Denmark, Finland, India, the Netherlands, Norway, Portugal, Russia, Saudi Arabia, Singapore, South Africa, Sweden, Switzerland and the United Arab Emirates in June.

Programming with Kinect

- Once you have Kinect SDK installed, add the reference of kinect.dll to your application and start programming.
- Kinect.dll, by default, is found in
  C:\Program Files\Microsoft SDKs\Kinect\v1.0\Assemblies\Microsoft.Kinect.dll)
- *The namespace to be added is Microsoft.Kinect*;
  Using Microsost.Kinect;

Enumerate Kinect sensors

```
private KinectSensor sensor;

foreach (var potentialSensor in KinectSensor.KinectSensors)
{
  if (potentialSensor.Status == KinectStatus.Connected)
  {
    this.sensor = potentialSensor;
    break;
  }
}
```


Start the Kinect

```
if (this.sensor != null)
{
  this.sensor.Start();
}
```


Enable Data Streaming

- There are several types of data that can be streamed out:
  - Color
  - Depth
  - Skeleton
- This example demonstrates how to enable each of the data streams.

```
if (this.sensor != null)
{
  this.sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
  this.sensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);
  this.sensor.SkeletonStream.Enable();
  this.sensor.ColorStream.Enable(ColorImageFormat.InfraredResolution640x480Fps30);
}
```


Handling events

- An event allows a class (or other object) to send notifications to other classes (or objects) that something has occurred. In simple terms an event is the outcome of a specific action.
- C# code uses the kinect event model by hooking any of the following events to the appropriate event handler.
  - Kinect.KinectSensor.ColorFrameReady,
  - KinectSensor.DepthFrameReady
  - KinectSensor.SkeletonFrameReady

Event Handling

- Register the event

```
if (this.sensor != null)
{
  this.sensor.ColorFrameReady += this.SensorColorFrameReady;
}
```

- Implement the event handler

```
private void SensorColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
{
  using (ColorImageFrame colorFrame = e.OpenColorImageFrame())
  {
    if (colorFrame != null)
    {
      colorFrame.CopyPixelDataTo(this.colorPixels);
      ...
    }
  }
}
```


Initializing Kinect Sensor

```
public HelloKinect()
{
    kinectsensor = KinectSensor.KinectSensors.FirstOrDefault();
    if (kinectsensor == null || kinectsensor.Status != KinectStatus.Connected)
    {
        Console.WriteLine("Unable to connect to Kinect device");
    }
    else
    {
        kinectsensor.SkeletonStream.Enable();
        kinectsensor.SkeletonFrameReady +=
            new EventHandler<SkeletonFrameReadyEventArgs>(kinect_SkeletonFrameReady);
        kinectsensor.Start();
    }

}
```


Retrieving data

- When a new frame of data is ready, the event is signaled and the handler runs and calls the following methods to get the frame.

```
SkeletonFrame skeletonFrame = e.OpenSkeletonFrame();
```

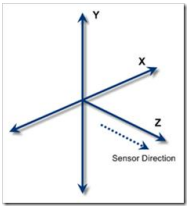

Skeletal Tracking

Retrieving joint positions

```
using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
{
    if (skeletonFrame != null)
    {

        Skeleton[] skeletons = new Skeleton[skeletonFrame.SkeletonArrayLength];
        skeletonFrame.CopySkeletonDataTo(skeletons);

        Skeleton trackedSkeleton = null;
        trackedSkeleton = skeletons.Where(s => s.TrackingState == SkeletonTrackingState.Tracked).First()

        foreach (Skeleton skeleton in skeletons)
        {
            if (skeleton.TrackingState == SkeletonTrackingState.Tracked)
            {
                foreach (Joint joint in skeleton.Joints)
                {
                    Console.WriteLine("{1} : {2} : {3}", skeleton, joint.JointType, joint.Position);
                }
            }
        }
    }
```


Joints

Each joint has a Position property that is defined by a Vector4: (x, y, z, w). The first three attributes define the position in camera space. The last attribute (w) gives the **quality** level (between 0 and 1) of the position. This allows you to filter and take only the data that the library is almost certain.



Smoothing

- *Smoothing*
  - Smoothing parameter. Increasing the smoothing parameter value leads to more highly-smoothed skeleton position values being returned.
  - It is the nature of smoothing that, as the smoothing value is increased, responsiveness to the raw data decreases.
  - Thus, increased smoothing leads to increased latency in the returned skeleton values.
  - Values must be in the range 0 through 1.0. Passing 0 causes the raw data to be returned.
- *Correction*
  - Correction parameter. Lower values are slower to correct towards the raw data and appear smoother, while higher values will correct toward the raw data more quickly.
  - Values must be in the range 0 through 1.0.
- *Prediction*
  - The number of frames to predict into the future.
  - Values must be greater than or equal to zero.
  - Values greater than 0.5 will likely lead to overshooting when moving quickly. This effect can be damped by using small values of fMaxDeviationRadius.
- *JitterRadius*
  - The radius in meters for jitter reduction.
  - Any jitter beyond this radius is clamped to the radius.
- *MaxDeviationRadius*
  - The maximum radius in meters that filtered positions are allowed to deviate from raw data.
  - Filtered values that would be more than this radius from the raw data are clamped at this distance, in the direction of the filtered value.

Filtering and Smoothing

```
TransformSmoothParameters smoothingParams = new TransformSmoothParameters();
smoothingParams.Smoothing = 0.75f;
smoothingParams.Correction = 0.0f;
smoothingParams.Prediction = 0.0f;
smoothingParams.JitterRadius = 0.05f;
smoothingParams.MaxDeviationRadius = 0.04f;
kinectsensor.SkeletonStream.Enable(smoothingParams);
```

Task I

- Write a kinect-enabled application that lets you navigate power point slides via hand gestures.

Last but not the least……

- You can use the following method to programmatically send a key stroke to an active application
  - System.Windows.Forms.SendKeys.SendWait("{Left}");
- This can be used to send <Right>/<Left> key inputs to navigate the power point slides to the next/previous slide.
  - System.Windows.Forms.SendKeys.SendWait("{Right}");
  - System.Windows.Forms.SendKeys.SendWait("{Left}");