Capturing Raw Streams Tutorial
Using Unity* Software

Intel® RealSense™ SDK 2014

With the Intel® RealSense™ SDK, you have access to robust, natural human-computer interaction (HCI) algorithms such as face tracking, finger tracking, gesture recognition, speech recognition and synthesis, fully textured 3D scanning and enhanced depth augmented reality.

Using the SDK and Unity* software you can create Windows* applications and games that offer innovative user experiences.

In this tutorial, you'll learn how to use the SDK to capture color and depth images from your input device. An application can render image samples within a loop and output the video data streams to a screen or output file.

By the end of this tutorial you'll be ready to start using the hand tracking and other algorithm modules in Unity software with the C# programming language.

Contents

Overview

The Intel RealSense SDK supports two types of modules: input/output modules and algorithm modules. This tutorial shows you how to implement I/O modules, and later tutorials show you how to implement algorithm modules.

This tutorial shows how to capture aligned color and depth samples, but it is also possible to capture them individually (unaligned). Capturing unaligned samples may be useful if you require a high frame rate for streaming depth data.

You can use either procedural calls (used in this tutorial) or event callbacks to capture data, and code samples are provided for both (see Table 1). Using event callbacks is usually preferred when developing console applications; procedural calls are often used for GUI applications.

Table 1: Code Samples

Code Sample	**For more information, see:**
Capturing aligned or unaligned color and depth streams using procedural calls File: RawDataCapture.cs	This Tutorial. Also see Color and Depth Samples using the SenseManager sections in the SDK Reference Manual.
Capturing aligned or unaligned color and depth streams using event callbacks	Color and Depth Samples using the SenseManager Events sections in the SDK Reference Manual.

- The depth stream can be used to innovatively show the user what exactly the RealSense Camera sees in your application.

Creating a Session

The SDK core is represented by two interfaces:

- **PXCMSession:** manages all of the modules of the SDK
- **PXCMSenseManager:** organizes a pipeline by starting, stopping, and pausing the operations of its various modalities.

The first step when creating an application that uses the Intel RealSense SDK is to create a session. A session can be created explicitly by creating an instance of **PXCMSession**. Each session maintains its own pipeline that contains the I/O and algorithm modules.

Another way of creating a session is by creating an instance of the **PXCMSenseManager** using **CreateInstance**. The PXCMSenseManager implicitly creates a session internally. Do this in the Start function before calling the Update method.

```
 /// <summary>
 /// Use this for initialization
 /// Unity function called on the frame when a script is enabled
 /// just before the Update method is called the first time.
/// </summary>
void Start () {

    /* Initialize a PXCMSenseManager instance */
    psm = PXCMSenseManager.CreateInstance();
    if (psm == null){
        Debug.LogError("SenseManager Initialization Failed");
        return;
    }

}
```

Initializing the Pipeline

1. Add the color and depth streams to the pipeline using the **EnableStream** function as separate calls.
   a. Specify the stream types **STREAM_TYPE_COLOR** and **STREAM_TYPE_DEPTH** from **PXCCapture**.
   b. Specify the resolution (**width** and **height**) of the streams.
2. Initialize the pipeline with the **Init** function so that the requested stream samples can be processed.

- Color stream resolution can support up to 1920x1080 pixels; you can configure various frame rates as well.
- The SDK also gives you access to left, right, and IR camera feeds.

Note: If a stream is not available with the specified settings, the camera will not stream—to indicate the settings are incorrect. When the settings are correct, **Init** function will return **PXC_STATUS_NO_ERROR** status.

```
 /// <summary>
/// Use this for initialization
/// Unity function called on the frame when a script is enabled
/// just before any of the Update methods is called the first time
/// </summary>
void Start ()
{
                /* Initialize a PXCMSenseManager instance */
                psm = PXCMSenseManager.CreateInstance();
                if (psm == null){
                   Debug.LogError("SenseManager Initialization Failed");
                   return;
                 }

               /* Enable the depth stream of size 640x480 and color stream of size 640x480 */
                psm.EnableStream(PXCMCapture.StreamType.STREAM_TYPE_DEPTH, 640, 480);
                psm.EnableStream(PXCMCapture.StreamType.STREAM_TYPE_COLOR, 640, 480);

                /* Initialize the execution pipeline */
                sts = psm.Init();
                if (sts != pxcmStatus.PXCM_STATUS_NO_ERROR){
                   Debug.LogError("PXCMSenseManager.Init Failed");
                   OnDisable();           // Clean-up
                   return;
                }
}
```

Capturing Color and Depth Streams

1. Perform all processing in the **Update** function, which Unity software calls every frame.
2. In every **Update** (per frame), first use the **AcquireFrame** function:
   a. TRUE (aligned) to wait for both color and depth samples to be ready in a given frame; else
   b. FALSE (unaligned) to return whenever either of the two samples are ready.
3. Retrieve an instance of sample from **PXCMCapture.Sample** through the **QuerySample** function.
4. Retrieve and render the individual color and depth images from the sample as explained in the next section of this tutorial.
5. Release the frame for reading the next samples (color + depth) through the **ReleaseFrame** function.

```
 /// <summary>
/// Update is called every frame by Unity, if the MonoBehaviour is enabled.
/// </summary>
void Update () {

   /* Make sure PXCMSenseManager Instance is Initialized */
   if (psm == null) return;

   /* Wait until any frame data is available true(aligned) false(unaligned) */
   if (psm.AcquireFrame(true) != pxcmStatus.PXCM_STATUS_NO_ERROR) return;

   /* Retrieve a sample from the camera */
   PXCMCapture.Sample sample = psm.QuerySample();
   if (sample != null)
   {
           /*Retrieve and render the individual color and depth images */
   }

   /* Release the frame to process the next frame */
   psm.ReleaseFrame();


}
```

Rendering Depth and Color Streams

1. Retrieve a sample image instance using **sample.depth** or **sample.color**.
2. If a Texture2D is not allocated (for the first time):
    a. Retrieve the sample image's resolution using **image.info.width** and **image.info.height** along with the **TextureFormat** and allocate the Texture2D.
    b. Associate the Texture2D with a gameObject with a mesh to render the texture by setting **gameObject.renderer.material.mainTexture** to the allocated Texture2D.
3. Retrieve the image data by using **AcquireAccess** on the sample image with the appropriate access type and pixel format, in this case, **PXCMImage.Access** and **PXCMImage.PixelFormat**.
4. Convert the image data to a texture using **imageData.ToTexture2D**.
5. Release the Access on the sample image using **sampleImage.ReleaseAccess**.
6. Apply the updated texture onto the **gameObject** mesh using **Texture2D.Apply**.
7. Repeat the same approach for the color data.

```
  /// <summary>
  /// Update is called every frame by Unity, if the MonoBehaviour is enabled.
  /// </summary>
  {
       depthImage = sample.depth;
        if (depthImage != null)
        {
           if (depthTexture2D == null)
           {
              /* If not allocated, allocate a Texture2D */
              depthTexture2D = new Texture2D(depthImage.info.width, depthImage.info.height,
TextureFormat.ARGB32, false);

              /* Associate the Texture2D with a gameObject */
              depthPlane.renderer.material.mainTexture = depthTexture2D;
              depthPlane.renderer.material.mainTextureScale = new Vector2(-1f, 1f); // for a mirror effect
           }

           /* Retrieve the image data in Texture2D */
           PXCMImage.ImageData depthImageData;
            depthImage.AcquireAccess(PXCMImage.Access.ACCESS_READ,
            PXCMImage.PixelFormat.PIXEL_FORMAT_RGB32, out depthImageData);
          depthImageData.ToTexture2D(0, depthTexture2D);
          depthImage.ReleaseAccess(depthImageData);

           /* Apply the texture to the GameObject to display on */
           depthTexture2D.Apply();
        }
  }
```

Cleaning Up the Pipeline

After your application is done capturing and rendering samples, you must “clean up”. This is done in the **OnDisable** function, which Unity software calls right before the behavior is disabled.

1. Check to make sure that **PXCMSenseManager** is already released.
2. If not, close all the last opened streams and release any session and processing module instances using **Dispose()** on the **PXCMSenseManager** instance.

```
/// <summary>
/// Unity function that is called when the behaviour becomes disabled () or inactive.
/// Used for clean-up in the end
/// </summary>
void OnDisable()
{
    if (psm == null) return;
    psm.Dispose();
}
```

Now you have all the information to configure, capture, render, and display raw color and depth data from input streams using your device.

Running the Code Sample

You can run the Unity* tutorial code sample by running the RawDataCapture scene in Unity software.

Figures 1 shows the output when capturing and rendering aligned color and depth streams from the **RawDataCapture.cs** unity code sample.

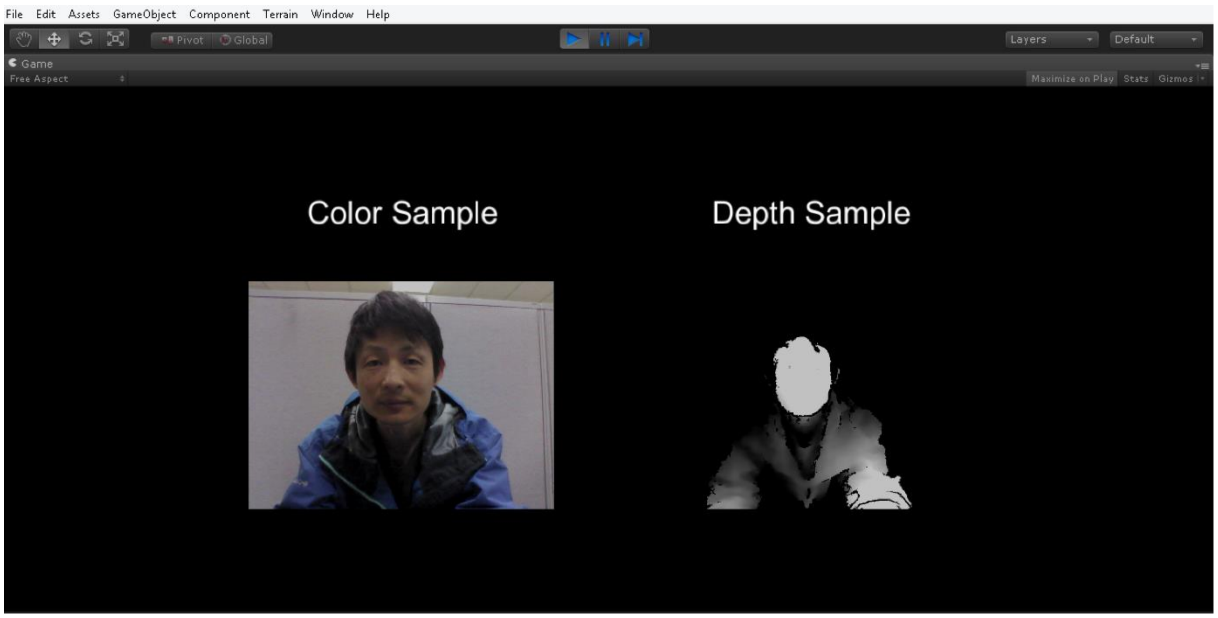

Figure 1. Rendered Color and Depth Streams

To learn more

- The SDK Reference Manual is your complete reference guide and contains API definitions, advanced programming techniques, frameworks, and other need-to-know topics.

- You can use PXC[M]CaptureManager to query a PXC[M]Capture device in order to manipulate camera behavior such as DepthConfidenceThreshold, IVCAMAccuracy, MirrorMode, , IVCAMMotionRangeTradeOff, etc. Refer to the Interface and Function Reference : Essential section in the SDK Reference Manual.

- You can extract z-depth data from the depth samples using **data.planes[0]**. Refer to the Access Image and Audio Data section in the SDK Reference Manual.