How to Use Azure-Kinect Examples for Unity

Despite its name, 'Azure-Kinect Examples' can work with several depth sensors – Azure-Kinect, RealSense and Kinect-v2. The installation depends on what sensor you have at your disposal.

1. (Azure Kinect) Download and install Azure-Kinect Sensor SDK, as described in 'Installation of Azure-Kinect SDKs' below.
2. (Azure Kinect) Download and install Azure-Kinect Body Tracking SDK, as described in 'Installation of Azure-Kinect SDKs' below.
3. (Kinect-v2) Download and install Kinect SDK 2.0, as described in 'Installation of Kinect-v2 SDK' below.
4. (RealSense) Download and install RealSense SDK 2.0, as described in 'Installation of RealSense SDK' below.
5. Import this package into new Unity project.
6. Open 'File / Build settings' and switch to 'PC, Mac & Linux Standalone', Target platform: 'Windows' and Architecture: 'x86_64'.
7. Make sure that Direct3D11 is the first option in the 'Auto Graphics API for Windows'-list setting, in 'Player Settings / Other Settings / Rendering'.
8. Open and run a demo scene of your choice from a subfolder of the 'AzureKinectExamples/KinectDemos'-folder. Short descriptions of the available demo scenes will be published soon.

Installation of Azure-Kinect SDKs

1. Download the latest version of 'Azure Kinect Sensor SDK'. Here is the download page: https://docs.microsoft.com/en-us/azure/Kinect-dk/sensor-sdk-download
2. Run the installer. The installation of 'Azure Kinect Sensor SDK' is straightforward.
3. Connect the Azure Kinect sensor.
4. Run 'Azure Kinect Viewer' to check, if the sensor and SDK work as expected.
5. Download the latest version of 'Azure Kinect Body Tracking SDK'. Here is the download page: https://docs.microsoft.com/en-us/azure/Kinect-dk/body-sdk-download
6. Follow the instructions here, to install the body tracking SDK: https://docs.microsoft.com/en-us/azure/Kinect-dk/body-sdk-setup
7. Run 'Azure Kinect Body Tracking Viewer' and move in front of the sensor to check, if the body tracking SDK works as expected.

Installation of Kinect-v2 SDK

1. Download the Kinect for Windows SDK 2.0. Here is the download page: http://www.microsoft.com/en-us/download/details.aspx?id=44561
2. Run the installer. Installation of Kinect SDK 2.0 is straightforward.
3. Connect the Kinect-v2 sensor. The needed drivers will be installed automatically.
4. Run 'SDK Browser 2.0' and select 'Kinect Configuration Verifier' to check, if the sensor and SDK work as expected.

Installation of RealSense SDK

1. Download the latest version of RealSense SDK 2.0. Here is the download page: https://github.com/IntelRealSense/librealsense/releases
2. Run the installer. Installation of RealSense SDK 2.0 is straightforward.
3. Connect the RealSense sensor.
4. Run 'Intel RealSense Viewer' to check, if the sensor and SDK work as expected.

Short Descriptions of the Available Demo Scenes

Short descriptions of the available demo scenes can be found in the online documentation.

How to run the VfxPointCloudDemo-scene

There are special preparations needed for the VfxPointCloudDemo-scene. Please do as follows:

1. Run the Unity package manager (menu Window / Package Manager). Select to view 'All packages' and enable the 'Show preview packages'-option.
2. Install the 'High Definition RP'-package.
3. Install the 'Visual Effect Graph'-package, as well.
4. Open the project settings (menu Edit / Project Settings) and select 'Graphics'. Then drag the HDRP-asset from the KinectDemos/PointCloudDemo/HDRP-folder of the project to the 'Scriptable Render Pipeline Settings' in the Graphics-section.
5. Select the 'Player'-section on the left. Change the color space from 'Gamma' to 'Linear'.
6. Open and run the VfxPointCloudDemo-scene from KinectDemos/PointCloudDemo-folder.
7. If you still can't see the point cloud on screen, select the VfxPointCloud-object and then press 'Edit' in the user interface of the 'Visual Effect'-component.
8. You can also select other visual effect as 'Asset template' of the VfxPointCloud-object, from the PointCloudDemo/VFX-folder.

The KinectManager and Sensor Interfaces

All sensor-related scenes require a KinectManager-component. In all demo scenes it resides in the KinectController-game object. This is the most basic component that manages the sensors and data. Its public API is utilized by all other sensor-related components. You can also utilize the public API of KinectManager in your scripts. Please note, the KinectManager persists across the scenes. In case of multi-scene setup, it should be created only once – in the startup scene. All other scenes can always reference it by calling the static 'KinectManager.Instance'-property.

Here are the public settings of the KinectManager-component:

Setting:	*Description:*
Use Multi Cam Config	Whether to create the sensor interfaces according to the multi-camera config, if one is available. Otherwise use the sensor interfaces, as configured in the child objects of this object.
Get Depth Frames	Whether to get depth frames from the sensor(s).
Get Color Frames	Whether to get color frames from the sensor(s).
Get Infrared Frames	Whether to get infrared frames from the sensor(s).
Get Pose Frames	Whether to get pose frames from the sensor(s).
Get Body Frames	Whether to get body frames from the body tracker.
Sync Depth and Color	Whether to synchronize depth and color frames.
Sync Body and Depth	Whether to synchronize body and depth frames.
Min User Distance	Minimum distance to user, in order to be considered for body processing. Value of 0 means no minimum distance limitation.
Max User Distance	Maximum distance to user, in order to be considered for body processing. Value of 0 means no maximum distance limitation.
Max Left Right Distance	Maximum left or right distance to user, in order to be considered for body processing. Value of 0 means no left/right distance limitation.
Max Tracked Users	Maximum number of users, who may be tracked simultaneously. Value of 0 means no limitation.
Show Allowed Users Only	Whether to display only the users within the allowed distances, or all users.
User Detection Order	How to assign users to player indices - by order of appearance, distance or left-to-right.
Ignore Inferred Joints	Whether to ignore the inferred joints, or consider them as tracked joints.
Ignore Z-Coordinates	Whether to ignore the Z-coordinates of the joints (for 2D-scenes) or not.
Bone Orientation Constraints	Whether to apply the bone orientation constraints.
Player Calibration Pose	Calibration pose required, to start tracking the respective user.
User Manager	User manager, used to track the users. KM creates one, if not set.
Gesture Manager	Gesture manager, used to detect user gestures. KM creates one, if not set.
Display Images	List of images to display on the screen.
Display Image Width	Single image width, as percent of the screen width.
Status Info Text	UI-Text to display status messages.

The K4A-package supports multiple sensor setups, as well as several types of sensors by design (Azure Kinect, Kinect-v2 & RealSense D400). In all demo scenes the available sensor interfaces reside on child objects of the KinectController-game object. At start up, the KinectManager tries to find the sensor interfaces on these child objects. If nothing is found, it creates a single Kinect4Azure interface on the same game object. Feel free to duplicate, create or remove sensor-interface objects, according to your specific setup. You can also change the sensor-interface settings, to match your setup needs. Please note, the transform-component of each sensor-interface object represents the sensor's pose in the scene.

Here are the common settings of all sensor-interface components:

Setting:	*Description:*
Device Streaming Mode	Device streaming mode, in means of connected sensor, recording or disabled.
Device Index	Index of the depth sensor in the list of currently connected sensors.
Recording File	Path to the recording file, if the streaming mode is PlayRecording.
Min Distance	Minimum distance in meters, used for creating the depth-related images.
Max Distance	Maximum distance in meters, used for creating the depth-related images.
Point Cloud Resolution	Resolution of the generated point-cloud textures.
Point Cloud Vertex Texture	Render texture, used for point-cloud vertex mapping. The texture resolution should match the depth or color image resolution.

Point Cloud Color Texture	Render texture, used for point-cloud color mapping. The texture resolution should match the depth or color image resolution.

Short Descriptions of the Components in KinectScripts-Folder

Short descriptions of the general-purpose components located in KinectScripts-folder can be found in the online documentation.

How to Reuse the Kinect-related Scripts and Components in Your Unity Project

1. Copy folder 'KinectScripts' from the AzureKinectExamples-folder of this package to your project. This folder contains all needed components, scripts, filters and interfaces.
2. Copy folder 'Resources' from the AzureKinectExamples-folder of this package to your project. This folder contains some needed libraries and resources.
3. Copy the sensor-SDK specific sub-folders (Kinect4AzureSDK, KinectSDK2.0 & RealSenseSDK2.0) from the AzureKinectExamples/SDK-folder of this package to your project. It contains the plugins and wrapper classes for the respective sensor types.
4. Wait until Unity detects, imports and compiles the newly detected resources and scripts.
5. Please do not share the KinectDemos-folder in source form or as part of public repositories.

More Information, Support and Feedback

Web: https://rfilkov.com/2019/07/24/azure-kinect-examples-for-unity/
Docs: https://github.com/rfilkov/k4adocs/wiki
Tips & tricks: https://rfilkov.com/2019/08/26/azure-kinect-tips-tricks/

Contact: http://rfilkov.com/about/#contact (please mention your invoice number)
Twitter: https://twitter.com/roumenf
LinkedIn: https://www.linkedin.com/in/rfilkov/