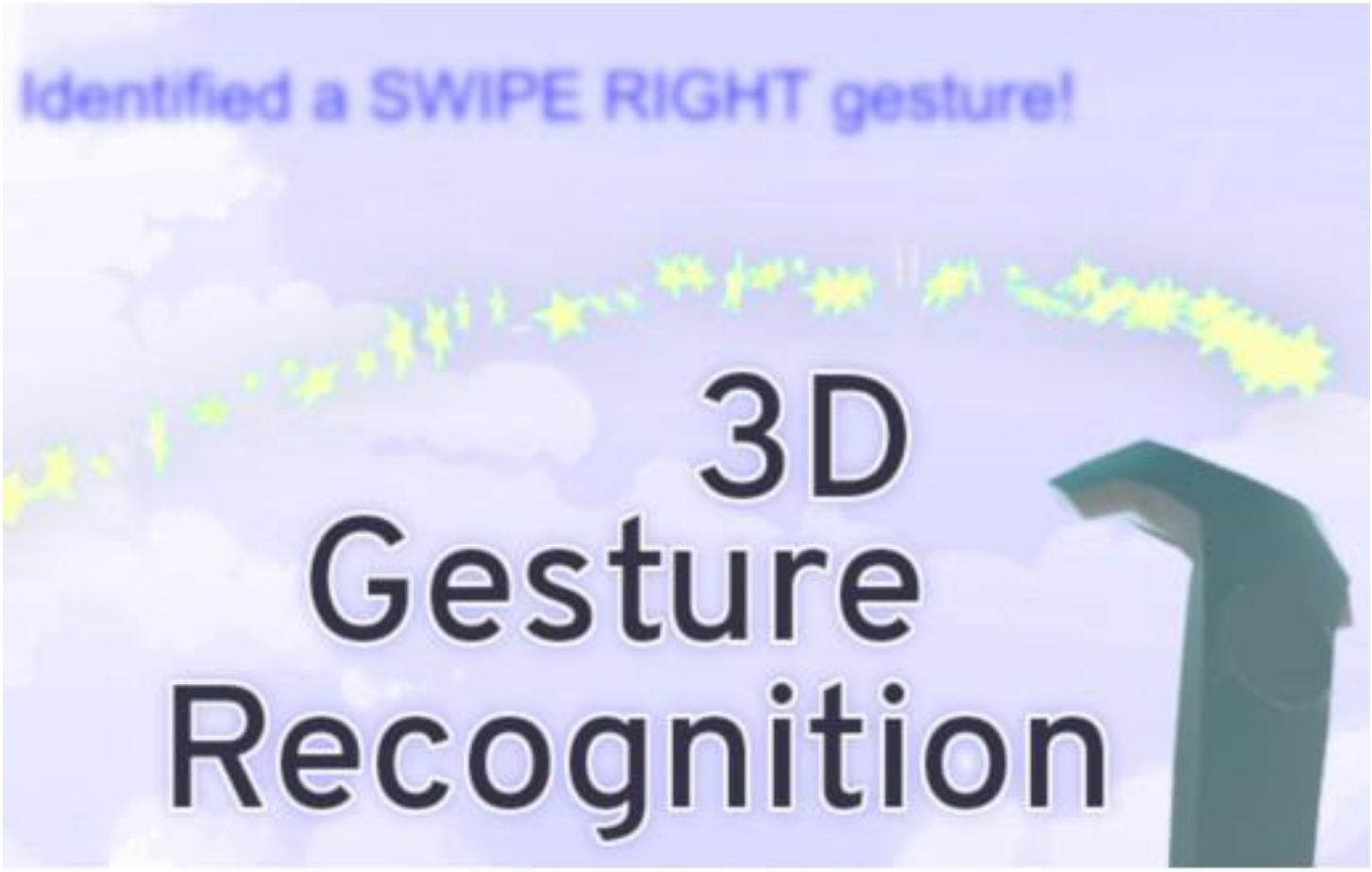
Identified a SWIPE RIGHT gesture!
3D
Gesture
Recognition

3D GESTURE RECOGNITION PLUG-IN FOR UNITY

[IMPORTANT!] This plug-in is free for non-commercial use. Commercial use is prohibited. See the license statement at the end of this document.

If you want to use this plug-in in a commercial application, please contact us at support@marui-plugin.com for a commercial license.

Check out our YouTube channel for tutorials, demos, and news updates:
https://www.youtube.com/playlist?list=PLYt4XosVlCmWtmDmx1lS8OVGU70tmQOfv

Content:

1: What is 3D Gesture Recognition?

Making good user interaction for VR is hard. The number of buttons often isn't enough and memorizing button combinations is challenging for users.
Gestures are a great solution! Allow your users to wave their 3D controllers like a magic want and have wonderful things happening. Draw an arrow to shoot a magic missile, make a spiral to summon a hurricane, shake your controller to reload your gun, or just swipe left and right to "undo" or "redo" previous operations.
MARUI has many years of experience of creating VR/AR/XR user interfaces for 3D design software. Now YOU can use its powerful gesture recognition module in Unity.
This is a highly advanced artificial intelligence that can learn to understand your 3D controller motions.
The gestures can be both direction specific ("swipe left" vs. "swipe right") or direction independent ("draw an arrow facing in any direction") - either way, you will receive the direction, position, and scale at which the user performed the gesture!
Draw a large 3d cube and there it will appear, with the appropriate scale and orientation.
Both one-handed and two-handed gestures are supported and even build combos of sequential gestures.

Key features:
- Real 3D gestures - like waving a magic wand in all three dimensions
- Support for multi-part gesture combinations such as two-handed gestures or sequential combinations of gestures
- Record your own gestures - simple and straightforward
- Easy to use - single C# class
- Can have multiple sets of gestures simultaneously (for example: different sets of gestures for different buttons)
- High recognition fidelity
- Outputs the position, scale, and orientation at which the gesture was performed
- High performance (back-end written in optimized C/C++)
- Includes a Unity sample project that explains how to use the plug-in
- Save gestures to file for later loading
- Support for Android-based devices (Oculus Quest, Cardboard, …)

2: Files included in this unity project:

- Plugins/x86_64/gesturerecognition.dll : The gesture recognition plug-in for Windows. Place this file in your Unity project under /Assets/Plugins/

- Plugins/Android/libgesturerecognition.so : The gesture recognition plug-in for Android. Place this file in your Unity project under /Assets/Plugins/

- GestureRecognition.cs : C# script for using the plugin. Include this file in your Unity project (for examples under /Assets/Scenes/Scripts/)

- GestureCombinations.cs : C# script for using the plugin for multi-part gesture combinations (eg. two-handed gestures or gesture combos). Include this file in your Unity project (for examples under /Assets/Scenes/Scripts/)

- GestureManager.unity : Unity scene that provides a convenient manager UI for creating and handling gesture files.

- Sample_OneHanded.unity / Sample_OneHanded.cs : Unity sample scene and script for one-handed gestures.

- Sample_OneHanded_Gestures.dat : Example gesture database file to one-handed gestures.

- Sample_TwoHanded.unity / Sample_TwoHanded.cs : Unity sample scene and script for two-handed gestures.

- Sample_TwoHanded_Gestures.dat : Example gesture database file to two-handed gestures.

- Sample_Military.unity / Sample_Military.cs : Unity sample scene and script for using military tactical gestures.

- Sample_Military_Gestures.dat : Example gesture database file for military gestures.

- all other files: assets used in the sample Unity projects.

3: Scripting tutorial for one-hand gestures:

(1) Place the Plugins/x86_64/gesturerecognition.dll (Windows) and/or Plugins/Android/libgesturerecognition.so (Android / MobileVR / Oculus Quest) files in the /Assets/Plugins/ folder in your unity project and add the GestureRecognition.cs file to your project scripts.

(2) Create a new Gesture recognition object and register the gestures that you want to identify later.

```
GestureRecognition gr = new GestureRecognition();
int myFirstGesture = gr.createGesture("my first gesture");
int mySecondGesture = gr.createGesture("my second gesture");
```

(3) Record a number of samples for each gesture by calling startStroke(), contdStroke() and endStroke() for your registered gestures, each time inputting the headset and controller transformation.

```
Vector3 hmd_p = Camera.main.gameObject.transform.position;
Quaternion hmd_q = Camera.main.gameObject.transform.rotation;
gr.startStroke(hmd_p, hmd_q, myFirstGesture);
[…]

// repeat the following while performing the gesture with your controller:
Vector3 p = OVRInput.GetLocalControllerPosition(OVRInput.Controller.RTouch);
Quaternion q = OVRInput.GetLocalControllerRotation(OVRInput.Controller.RTouch);
gr.contdStroke(p,q);
// ^ repead while performing the gesture with your controller.

[…]
gr.endStroke();
```

Repeat this multiple times for each gesture you want to identify.
We recommend recording at least 20 samples for each gesture.

(4) Start the training process by calling startTraining().
You can optionally register callback functions to receive updates on the learning progress by calling setTrainingUpdateCallback() and setTrainingFinishCallback().

```
gr.setMaxTrainingTime(10); // Set training time to 10 seconds.
gr.startTraining();
```

You can stop the training process by calling stopTraining().
After training, you can check the gesture identification performance by calling recognitionScore() (a value of 1 means 100% correct recognition).

(5) Now you can identify new gestures performed by the user in the same way as you were recording samples:

```
Vector3 hmd_p = Camera.main.gameObject.transform.position;
Quaternion hmd_q = Camera.main.gameObject.transform.rotation;
gr.startStroke(hmd_p, hmd_q);
[…]

// repeat the following while performing the gesture with your controller:
Vector3 p = OVRInput.GetLocalControllerPosition(OVRInput.Controller.RTouch);
Quaternion q = OVRInput.GetLocalControllerRotation(OVRInput.Controller.RTouch);
gr.contdStroke(p,q);
// ^ repeat while performing the gesture with your controller.
[…]

int identifiedGesture = gr.endStroke();
if (identifiedGesture == myFirstGesture) {
    // ...
}
```

(7) More than just getting the most likely candidate which gesture was performed, you can also get the similarity how much the performed motion resembles the identified gesture:

```
double similarity;
int identifiedGesture = gr.endStroke(similarity);
```

This returns a value between 0 and 1, where 0 indicates that the performed gesture is very much unlike the previously recorded gestures, and 1 indicates that performed gesture is the exact average of all previously recorded gestures and thus highly similar to the intended gesture.

(6) You can save and load your gestures to a gesture database file.

```
gr.saveToFile("C:/myGestures.dat");
// ...
gr.loadFromFile("C:/myGestures.dat");
```

4: Scripting tutorial for two-handed gestures or gesture combos:

(1) Place the Plugins/x86_64/gesturerecognition.dll (Windows) and/or Plugins/Android/libgesturerecognition.so (Android / MobileVR / Oculus Quest) files in the /Assets/Plugins/ folder in your unity project and add the GestureCombinations.cs file to your project scripts.

(2) Create a new Gesture recognition object and register the gestures that you want to identify later. (In this example, we use gesture part "0" to mean "left hand" and gesture part "1" to mean right hand, but it could also be two sequential gesture parts performed with the same hand.)

```
GestureCombinations gc = new GestureCombinations(2);
int myFirstCombo = gc.createGestureCombination("wave your hands");
int mySecondCombo = gc.createGesture("play air-guitar");
```

Also, create the individual gestures that each combo will consist.

```
int myFirstCombo_leftHandGesture = gc.createGesture(0, "Wave left hand");
int myFirstCombo_rightHandGesture = gc.createGesture(1, "Wave right hand");
int mySecondCombo_leftHandGesture = gc.createGesture(0, "Hold guitar neck");
int mySecondCombo_rightHandGesture = gc.createGesture(1, "Hit strings");
```

Then set the Gesture Combinations to be the connection of those gestures.

```
gc.setCombinationPartGesture(myFirstCombo, 0, myFirstCombo_leftHandGesture);
gc.setCombinationPartGesture(myFirstCombo, 1, myFirstCombo_rightHandGesture);
gc.setCombinationPartGesture(mySecondCombo, 0, mySecondCombo_leftHandGesture);
gc.setCombinationPartGesture(mySecondCombo, 1, mySecondCombo_rightHandGesture);
```

(3) Record a number of samples for each gesture by calling startStroke(), contdStroke() and endStroke() for your registered gestures, each time inputting the headset and controller transformation.

```
Vector3 hmd_p = Camera.main.gameObject.transform.position;
Quaternion hmd_q = Camera.main.gameObject.transform.rotation;
gc.startStroke(0, hmd_p, hmd_q, myFirstCombo_leftHandGesture);
gc.startStroke(1, hmd_p, hmd_q, myFirstCombo_rightHandGesture);
[…]
// repeat the following while performing the gesture with your controller:
Vector3 p_left = OVRInput.GetLocalControllerPosition(OVRInput.Controller.LTouch);
Quaternion q_left = OVRInput.GetLocalControllerRotation(OVRInput.Controller.LTouch);
gc.contdStroke(0, p_left, q_left);
Vector3 p_right = OVRInput.GetLocalControllerPosition(OVRInput.Controller.RTouch);
Quaternion q_right = OVRInput.GetLocalControllerRotation(OVRInput.Controller.RTouch);
gc.contdStroke(0, p_right, q_right);
// ^ repead while performing the gesture with your controller.
[…]
gc.endStroke(0);
gc.endStroke(1);
```

Repeat this multiple times for each gesture you want to identify.
We recommend recording at least 20 samples for each gesture, and have different people perform each gesture.

(4) Start the training process by calling startTraining().

You can optionally register callback functions to receive updates on the learning progress by calling setTrainingUpdateCallback() and setTrainingFinishCallback().

```
gc.setMaxTrainingTime(60); // Set training time to 60 seconds.
gc.startTraining();
```

You can stop the training process by calling stopTraining(). After training, you can check the gesture identification performance by calling recognitionScore() (a value of 1 means 100% correct recognition).

(5) Now you can identify new gestures performed by the user in the same way as you were recording samples:

```
Vector3 hmd_p = Camera.main.gameObject.transform.position;
Quaternion hmd_q = Camera.main.gameObject.transform.rotation;
gc.startStroke(0, hmd_p, hmd_q);
gc.startStroke(1, hmd_p, hmd_q);
[…]

// repeat the following while performing the gesture with your controller:
Vector3 p_left = OVRInput.GetLocalControllerPosition(OVRInput.Controller.LTouch);
Quaternion q_left = OVRInput.GetLocalControllerRotation(OVRInput.Controller.LTouch);
gc.contdStroke(0, p_left, q_left);
Vector3 p_right = OVRInput.GetLocalControllerPosition(OVRInput.Controller.RTouch);
Quaternion q_right = OVRInput.GetLocalControllerRotation(OVRInput.Controller.RTouch);
gc.contdStroke(1, p_right, q_right);
// ^ repead while performing the gesture with your controller.
[…]

gc.endStroke(0);
gc.endStroke(1);
int identifiedGestureCombo = gc.identifyGestureCombination();
if (identifiedGestureCombo == myFirstCombo) {
    // ...
}
```

(6) Now you can save and load the artificial intelligence.

```
gc.saveToFile("C:/myGestureCombos.dat");
// ...
gc.loadFromFile("C:/myGestureCombos.dat");
```

5: Build instructions for Windows

(1) Make sure plug-in file (Plugins/x86_64/gesturerecognition.dll) is in the “Plugins” folder of your project

(2) In your Unity editor, select the plug-in file and in the inspector make sure it is selected as a plug-in file for the Windows platform.

(3) Place your gesture database (“.DAT”) files in a folder called “StreamingAssets” in your Unity project.

(4) In your Unity script file, use `Application.streamingAssetsPath` as base folder when loading the gesture library instead of an absolute file path. You can use the UNITY_EDITOR preprocessor variable to make sure your game will find the gesture recognition database file both when playing in the Unity editor and when building / exporting as a stand-alone game:

```
#if UNITY_EDITOR
gr.loadFromFile("myProject/myGestureDatabaseFile.dat");
#else
gr.loadFromFile(Application.streamingAssetsPath + "/myGestureDatabaseFile.dat");
#endif
```

6: Build instructions for Android (Mobile VR, Oculus Quest, …)

(1) Make sure plug-in file (Plugins/Android/libgesturerecognition.so) is in the “Plugins” folder of your project

(2) In your Unity editor, select the plug-in file and in the inspector make sure it is selected as a plug-in file for the Android platform.

(3) Place your gesture database (“.DAT”) files in a folder called “StreamingAssets” in your Unity project.

(4) In your Unity script file, use `Application.streamingAssetsPath` as base folder when loading the gesture library instead of an absolute file path. You can use the UNITY_EDITOR preprocessor variable to make sure your game will find the gesture recognition database file both when playing in the Unity editor and when building / exporting as a stand-alone game:

```
#if UNITY_EDITOR
gr.loadFromFile("myProject/myGestureDatabaseFile.dat");
#else
gr.loadFromFile(Application.streamingAssetsPath + "/myGestureDatabaseFile.dat");
#endif
```

7 Software license statement (EULA):

This software is free to use for non-commercial purposes. You may use this software in part or in full for any project that does not pursue financial gain, including free software and projects completed for evaluation or educational purposes only. Any use for commercial purposes is prohibited.

You may not sell or rent any software that includes this software in part or in full, either in its original form or in altered form.

If you wish to use this software in a commercial application, please contact us at contact@marui-plugin.com to obtain a commercial license.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.