MiVRy – 3D Gesture Recognition AI

v1.6

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 files (Plugins/Android/arm64-v8a/libgesturerecognition.so and Plugins/Android/armeabi-v7a/libgesturerecognition.so) are in the “Plugins” folder of your project.

(2) In your Unity editor, select the plug-in files and in the inspector make sure it is selected as a plug-in file for the Android platform for ARM64 and ARMv7 respectively.

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

(4) In your Unity script file, use Unity’s Android Java API to get the location of the cache folder and use a UnityWebRequest to extract the gesture database file from the .apk to the cache folder and load it from there. This is necessary, because on Android all project files are packed inside the .apk file and cannot be accessed directly. You can use the UNITY_ANDROID 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 Android app:

```
LoadGesturesFile = "myGestures.dat";
// Find the location for the gesture database (.dat) file
#if UNITY_EDITOR
// When running the scene inside the Unity editor,
// we can just load the file from the Assets/ folder:
string gesture_file_path = "Assets/GestureRecognition";
#elif UNITY_ANDROID
// On android, the file is in the .apk,
// so we need to first "download" it to the apps' cache folder.
AndroidJavaClass unityPlayer = new AndroidJavaClass("com.unity3d.player.UnityPlayer");
AndroidJavaObject activity = unityPlayer.GetStatic<AndroidJavaObject>("currentActivity");
string gesture_file_path = activity.Call
                          <AndroidJavaObject>("getCacheDir").Call<string>("getCanonicalPath");
UnityWebRequest request = UnityWebRequest.Get(Application.streamingAssetsPath
                         + "/" + LoadGesturesFile);
request.SendWebRequest();
while (!request.isDone) {
    // wait for file extraction to finish
}
if (request.isNetworkError)
{
    // Failed to extract sample gesture database file from apk
    return;
}
File.WriteAllBytes(gesture_file_path + "/" + LoadGesturesFile, request.downloadHandler.data);
#else
// This will be the case when exporting a stand-alone PC app.
// In this case, we can load the gesture database file from the streamingAssets folder.
string gesture_file_path = Application.streamingAssetsPath;
#endif
if (gr.loadFromFile(gesture_file_path + "/" + LoadGesturesFile) == false)
{
    // Failed to load sample gesture database file
    return;
}
```

(5) In your project settings, make sure that under "*Other Settings*" you have selected the proper **API Level** and **Target Architecture** for the Android device that you want to use. For Oculus Quest, the API level should be "Android 8.0 Oreo" and the Target Architecture should be ARM64 (ARMv7 may also work). Also, under "XR Settings" make sure that you have enabled "**Virtual Reality Support**" and include the required SDKs for your device.

7 Troubleshooting and Frequently Asked Questions (FAQ):

(1) Where in my own program do I have to create the GestureRecognition or GestureCombination object?

You can create the gesture recognition object anywhere in your project. There are no special requirements where to do it. Commonly, it is created in the XR rig or Oculus/HTC Vive VR framework where the controller input is processed, but this is just one option.

(2) How can I get the position of VR controllers (Oculus Touch, HTC Vive Controllers, Valve Knuckles controller etc)?

As you can see in the Sample_OneHanded.unity scene, you can use the generic Unity XR rig with two objects "Left Hand" and "Right Hand" which are set to be Generic XR Controllers. So they work for any supported VR device.
Then, in the C# script you can just use

```
GameObject left_hand = GameObject.Find("Left Hand");
gc.contdStroke(Side_Left, left_hand.transform.position, left_hand.transform.rotation);
```

or

```
GameObject right_hand = GameObject.Find("Right Hand");
gc.contdStroke(Side_Right, right_hand.transform.position, right_hand.transform.rotation);
```

If in your project you cannot use the XR rig, please check out the Unity documentation for which commands will relate to your device:
https://docs.unity3d.com/Manual/OpenVRControllers.html

(3) How can I save my own recorded gestures to use them the next time I start Unity?

In your own script, you can save your recorded gestures with
`gr.saveToFile("C:/where/you/want/your/myGestureCombos.dat");`
This file you can then load next time you start the program.
If you used the GestureRecognitionSample_OneHanded Unity file, then your gestures will be saved in your asset folder in
`GestureRecognition\Sample_TwoHanded_MyRecordedGestures.dat`
Please see the GestureRecognitionSample_OneHanded.cs script on line 429 to see how it works.

(4) How can I open and edit gesture database (.DAT) files?

Please use the "GestureManager" scene in the Unity sample to open and edit.DAT gesture database files.

(5) The Gesture Recognition library does not detect if a gesture is different from all recorded gestures. I want to know if the user makes the gesture I recorded or not.

The gesture recognition plug-in will always return the number of which other (known) gesture is most similar to the one you just performed.
If you want to check if the gesture you made is different from all the recorded gestures, use the following code instead of the normal "endStroke()" function:

```
double similarity;
int identified_gesture = endStroke(ref similarity);
```

Then the similarity variable will give you a measurement of how similar the performed gesture was to the detected gesture. A value of one will indicate perfect similarity, a low value close to zero indicate great differences between the performed gesture and the recorded gesture. You can use this value to judge if the performed gesture is sufficiently similar to the recorded one.

(6) I want to use Gesture Recognition in my commercial project. What commercial licensing options do you provide?

We offer both single-payment licenses for one project or profit-sharing licenses where we receive a part of the sales price on each unit sold.
Pricing is dependent on the size of your project.
Please contact us at support@marui-plugin.com for details.

(7) Do I have to call "startTraining()" every time I start my game? Does it have to keep running in the background while my app is running?

No, you only need to call startTraining() after you have recorded new gesture data (samples) and want these new recordings to be used by the AI. However, you need to save the AI after training to a database file (.DAT) and load this file in your game before using the other gesture recognition functions.
While the training is running, you cannot use any of the other functions, so you cannot let training run in the background. You must start (and stop) training in between using the AI.

(8) How long should I let the training run to achieve optimal recognition performance?

Usually, the AI will reach its peak performance within one minute of training, but if you're using a large number of gestures and samples, it may take longer. You can check the current recognition performance from the training callback functions and see if the performance still keeps increasing. If not, feel free to stop the training.

8 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.