Imagine WebAR Hand Tracker

Version:	1.0.1
WebGL Demos:	https://webar.imaginerealities.com.au/ht/demo
Contact Support:	https://imaginerealities.com.au/contact-support
Publisher's Website:	https://imaginerealities.com.au
Unity Forums Thread:	Coming soon
Discord Community:	https://discord.gg/ypNARJJEbB

WebGL Introduction

The web is one of the most promising platforms for augmented reality, because it allows users to experience AR without the need to download a standalone application. And most, if not all, WebAR plugins for Unity require developers to pay on a subscription and/or per view basis.

Imagine WebAR Image Tracker is an augmented reality plugin for Unity WebGL which allows developers to implement AR experiences for the web. This plugin also allows developers to host their own AR experiences like any other Unity WebGL build.

WebGL applications built using this plugin are able to run on both desktop and mobile browsers. And since AR is mostly experienced through mobile devices, we are giving mobile browsers a higher priority.

Render Pipelines supported are Built-In RP and URP. Though, some rendering features are not supported (see **Current Limitations**) or still experimental.

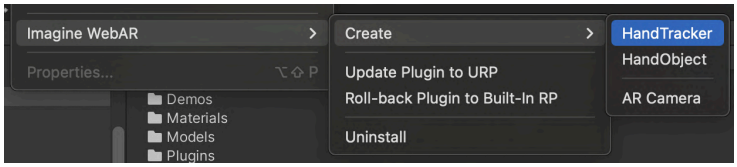
The plugin consists mainly of two modules: [1] a computer-vision (CV) module, written in Javascript, which uses the MediaPipe Hand Tracking solution. [2] A Unity Editor module which provides the tools necessary to create your Hand tracking AR Scene in Unity.

Setting up your AR Scene in Unity

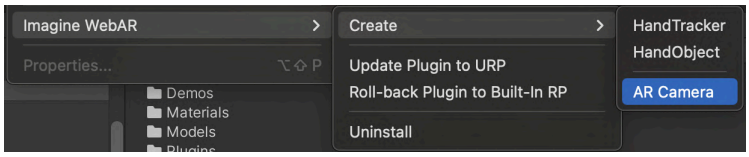
A simple AR scene can be set up in a few minutes. The HandTracker tutorial video can be found here: https://www.youtube.com/watch?v=bfEtM50xb7Q

To set up your first scene:

1.) Import the plugin and create a new Scene in Unity.
2.) Create a **Hand Tracker** from **Assets>Imagine WebAR>Create>Hand Tracker**



3.) Delete the Main Camera gameobject, and create an **ARCamera** from **Assets>Imagine WebAR>Create>ARCamera.** Drag this object into the Tracker Cam property of the Hand Tracker

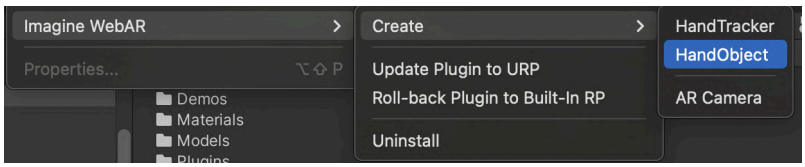


4.) Now we are ready to add **Hand Objects** to our tracker.

Adding Hand Objects to your Tracker

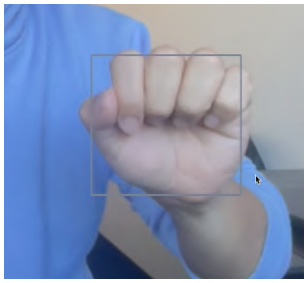
When the tracker detects a hand, it overlays a corresponding handObject in front of the camera. This can be an AR effect, virtual wearable, animated model, or a mix of everything. You can have at most 4 handObjects in your scene. Note that the more hands you track simultaneously, the more performance intensive your experience becomes.

To add a HandObject,from **Assets>Imagine WebAR>Create>HandObject**. HandObjects should have a unique **Hand Index** (0-3) for each of the 4 hands.



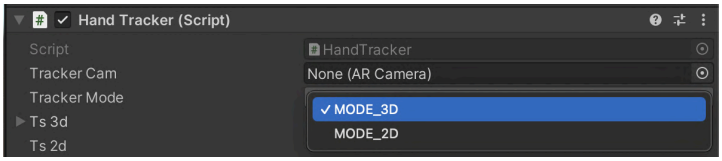
Hand Object - 2D and 3D Modes

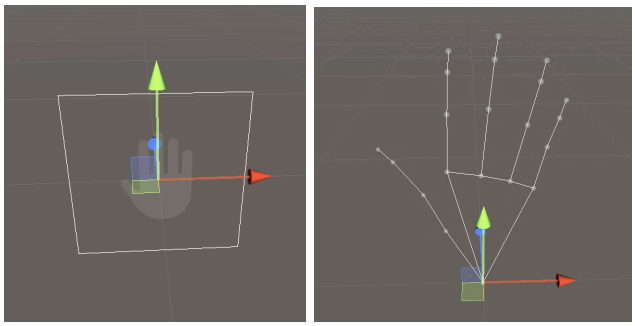
HandObjects can either be tracked in 2D or 3D mode. In 2D mode, the handObject will only move in the x and y axes, and will be locked at a fixed z (depth). This is best for 2D hand overlays or if your experience only needs to capture the hand position and motion. Aside from the xy-coordinate, you can also get the bounding box of the hand.



In 3D mode, you will be able to get the joint position and rotation of each hand landmark, which allows you to do more - such as anchoring virtual wearables, or triggering events based on finger positions. Note though - that while the screen x and y positions are mostly accurate, mediapipe just estimates the depth of each joint which introduces errors especially when some parts of the hand are occluded.

You can set the tracking mode in your **HandTracker>TrackerMode**

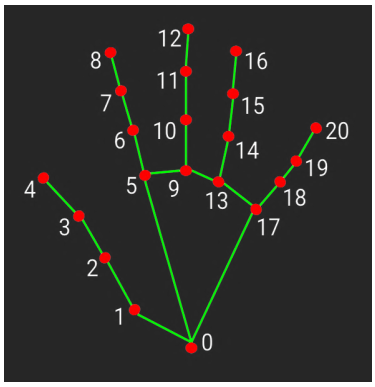




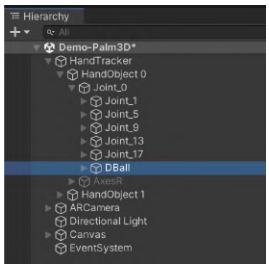
The left image shows a hand object in 2D mode, while the right image shows it in 3D mode.

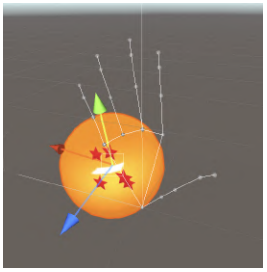
Hand Object - 3D Joint Positions

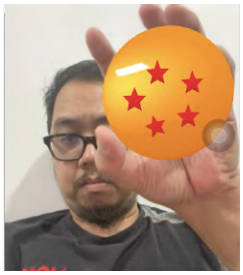
The joint diagram below shows the indices and positions of the hand joints (based on mediapipe).



And as an example, a virtual object can be anchored to the palm of the hand by dragging them inside **Joint_0** of your HandObject. And then further adjusting its position and scale accordingly.



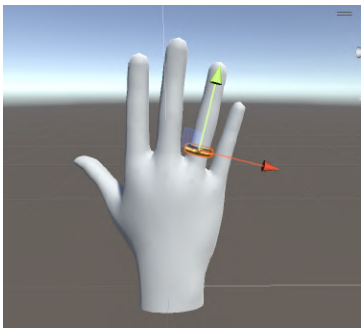


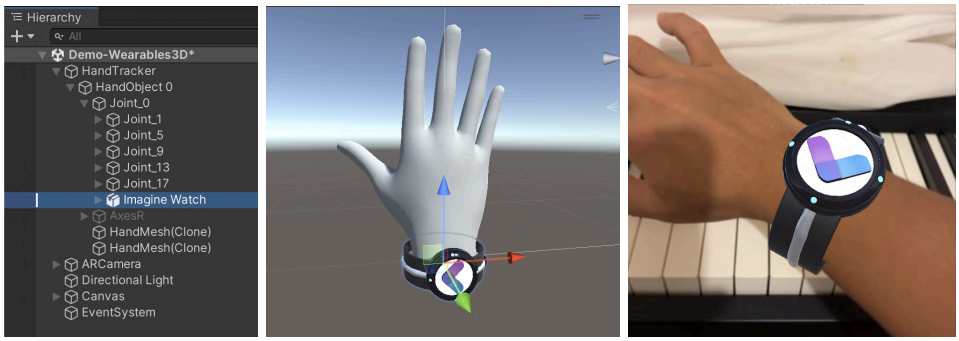


Hand Object - Wearable

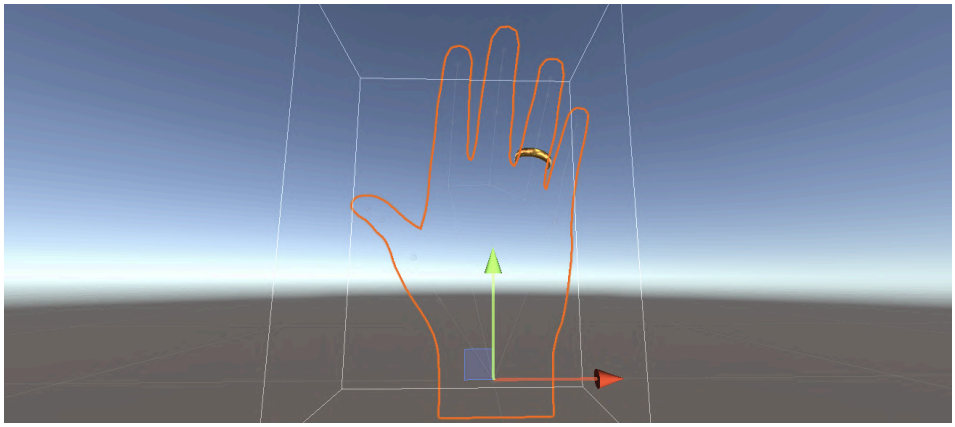
Virtual wearables and hand decorations are easy to implement because we are simply anchoring 3D models to the detected hand. Simply drag and position your gameObject then make sure to parent them in the appropriate joint in your Hand Object. Use the **joint diagram** above to find the correct joint. As an example, a ring can go in **Joint_13**, while a watch or bracelet can go to **Joint_0**.

You can enable **HandObject>Use Hand Mesh** to easily position your wearable gameObjects appropriately on a 3D hand.





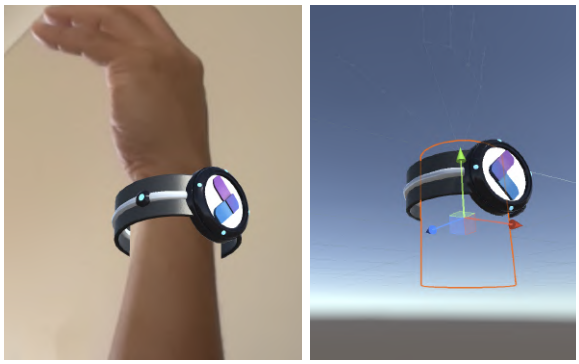
Hand Occluder



The hand occluder will mask out 3D model parts that are normally occluded by parts of the hand. You can use the handmesh as an occluder by simply replacing its material shader to **Imagine/DepthMask**.

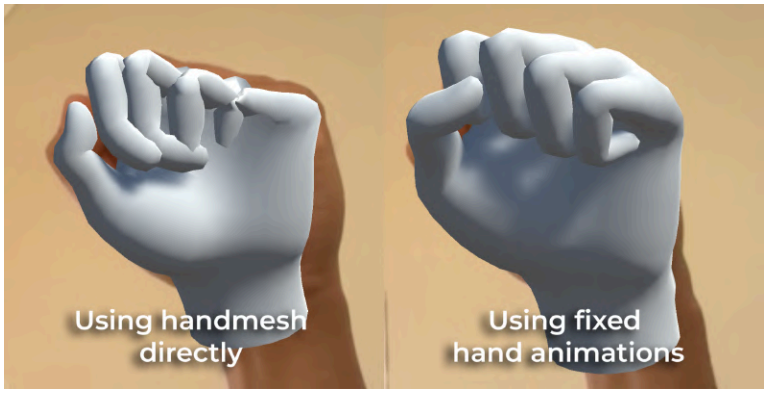
Note: To mask out transparent objects without any issues, you will need to use **ARCamera>VideoPlaneMode>TexturePtr** in tandem with the hand occluder.

This concept can be further applied to occlude other parts of the body as needed. The example below uses a default cylinder as an arm occluder in tandem with the hand occluder.

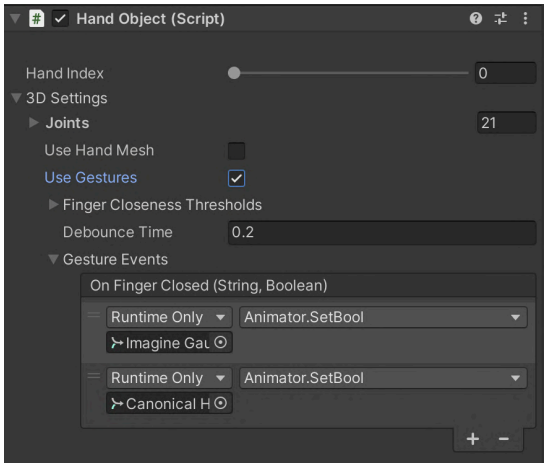


Hand Object - Finger Animation Events

While the hand mesh works great as a hand mask, rendering it directly is not ideal, as the fingers will occasionally show deformed. This is because the depth estimation of the mediapipe plugin is not very precise, especially when some regions of the hand are occluded. In these cases, we recommend using fixed hand animations based on “Finger Closeness”. This approach is similar to hand animations in virtual reality controllers. This allows you to display better-looking hands, at the expense of joint freedom and transition delays.

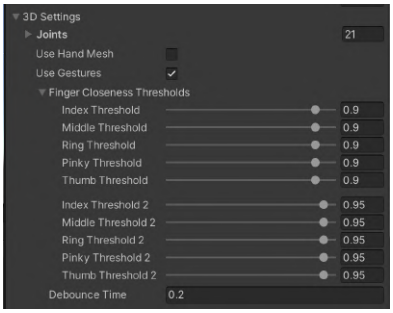


You can enable hand animation events by enabling **HandObject>3D Settings>Use Gestures.**

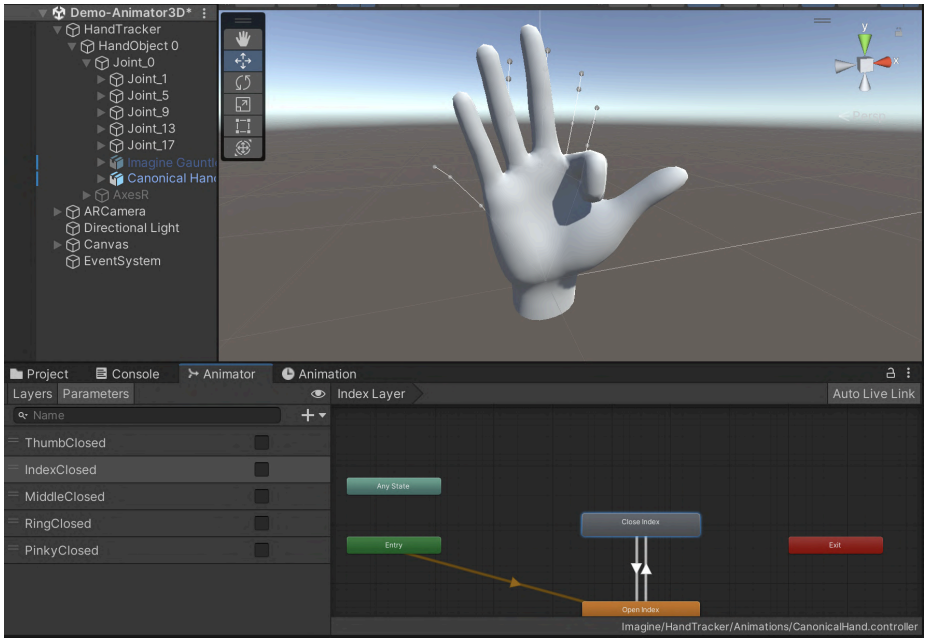


The HandObject tracks “Finger Closeness” and sends events for each finger. The state of each finger (open or closed) is determined based on the position and angle of each joint segment.

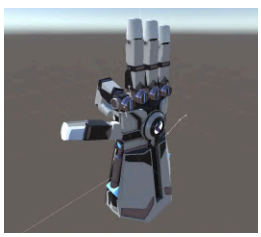
You can tweak the sensitivity of each finger in **HandObject>3D Settings>Finger Closeness Thresholds.**

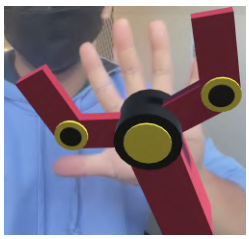


You can also use **HandObject>3D Settings>OnFingerClosed** to directly trigger your Animator Bool parameters for your fingers. You need to add the following bool parameters in your animator controller - **ThumbClosed, IndexClosed, MiddleClosed, RingClosed, and PinkyClosed** to trigger your corresponding animation state.
You may also choose to add different animation layers as needed.

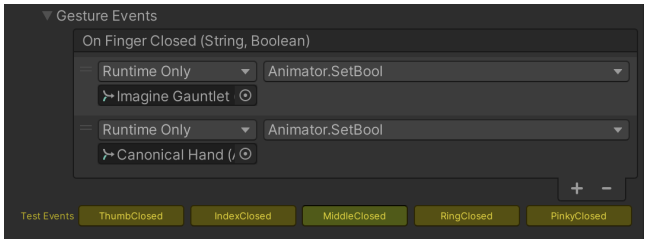


With this approach, you can also animate fantasy hands such as robotic gauntlets and monster claws. You don't need to use all 5 fingers in some of these cases.



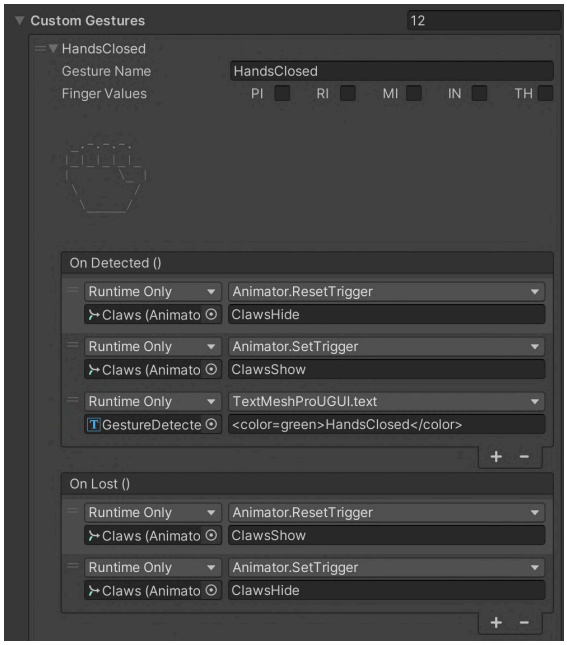



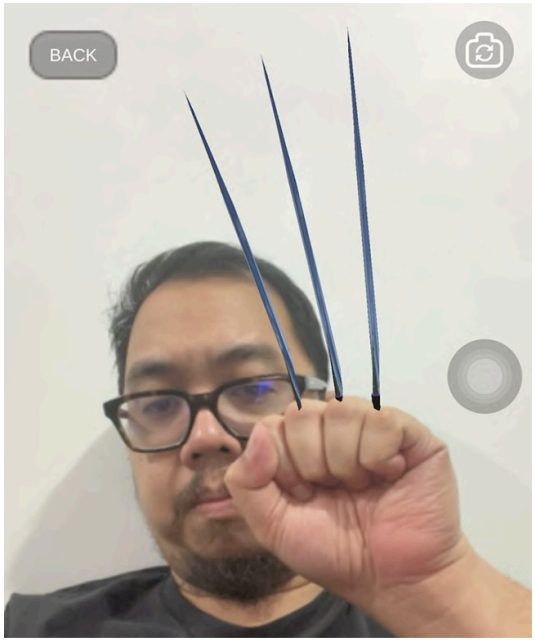
Test your finger events in Editor PlayMode using the TestEvents toggle button, when running your project in the Unity Editor.



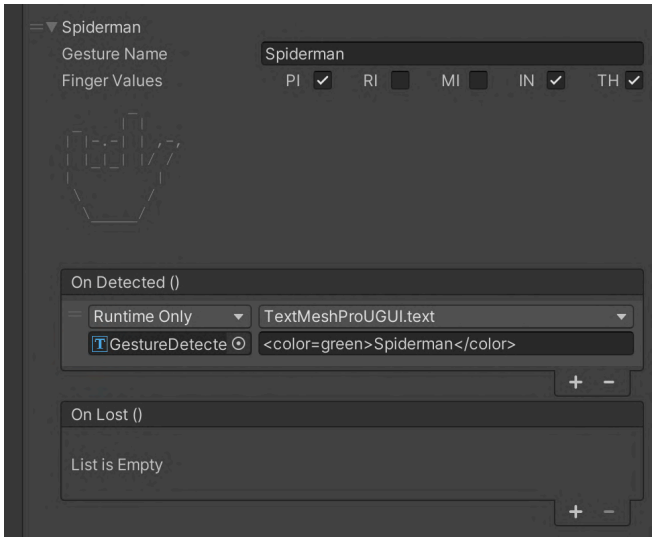
Hand Object - Gesture Detection (Experimental)

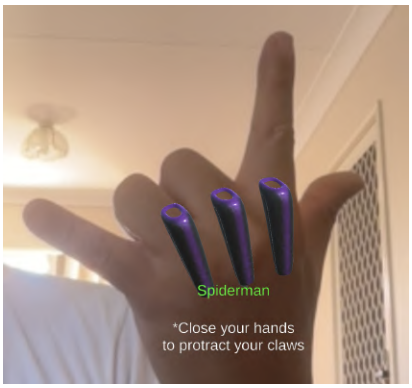
Further extending from the finger closeness functionality, comes gesture detection**.** These allow you to create **Custom Gestures** and raise events when such gestures are detected or lost. You can do this by enabling **HandObject>3D Settings>Use Gestures**, and then adding your gesture in **HandObject>3D Settings>Custom Gestures**



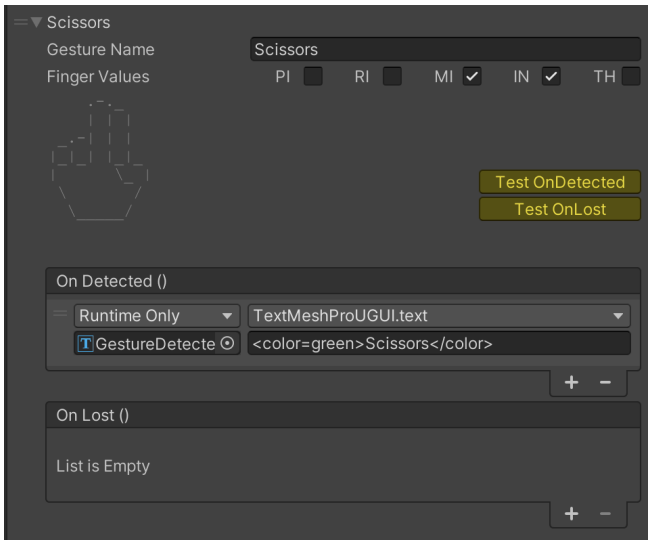


You can enable fingers using **Finger Values** checkboxes (**PI** for Pinky, **RI** for Ring, **MI** for Middle, **IN** for Index, and **TH** for Thumb) for your gestures. An ascii representation should help you visualize your gestures. And then you can hook up your functions in the **OnDetected** and **OnLost** events.





Test your custom gesture events in Editor PlayMode using the TestOnDetect and TestOnLost buttons, when running your project in the Unity Editor.

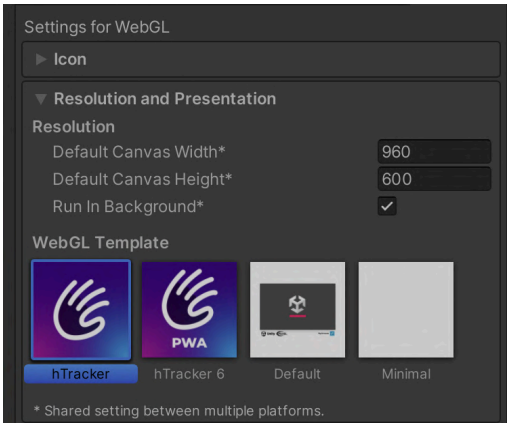


Important: The gesture recognition feature is still experimental and may still not be able to track hand gestures reliably, especially when some fingers are occluded. We are planning to implement a faster and more reliable approach in future versions of the HandTracker such as in this link - https://mediapipe-studio.webapps.google.com/studio/demo/gesture_recognizer

Building your AR Scene

After setting up all your handObjects, you can now build your AR scene.

1.) Go to **File>Build Settings,** switch to the **WebGL** platform if needed and add your scene in the build. Then select **hTracker** template in **Player Settings>Resolution and Presentation>WebGL Template**



Note: For better Unity 6 compatibility, you can also use the experimental **hTracker 6** template.

2.) **Build and Run** your project.

3.) Allow access to your device camera and stand in front of the webcam. You should see the hand objects anchored in your hands in AR.

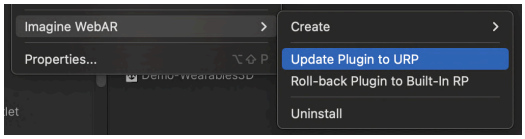
Updating the plugin to URP

Imagine WebAR also supports the Universal Render Pipeline for WebGL. To update the plugin to URP, follow the steps below:

1.) Create a new Unity project using the **3D(URP)** template.

2.) Import the plugin.

3.) Go to **Assets>Imagine WebAR>Update Plugin to URP**



4.) Wait for the reimport to finish.

Note: Failure to do this step when building in URP can cause black-screen or other camera issues.

WebAR Best Practices

Lightweight AR experiences and game optimizations

Another thing to consider is that WebAR runs in web browsers with very limited processing power compared to other platforms. So it is important to choose a lightweight experience and well-optimized to run even on low-tier mobile devices across a wide range of browsers and versions. Consider the following simplifications:

- Use low-res sprite sheets and/or low-poly models
- Use simple/unlit shaders whenever possible
- Avoid real-time image effects and post-processing
- Use simple animations instead of physics simulations
- Total memory consumed less than 300MB

FPS and Overheating

It is also very important to consider the impact of frame rate to the tendency of the device to heat-up. AR experiences, in general, consume more resources (such as camera and calculations) than standard games, thus putting more strain on device hardware and causing them to heat up, especially on high frame rates and longer play sessions. And In response to overheating, devices will cap the framerate to avoid any serious damage.

From our testing, mid-tier devices (eg. iPhone 8) is able to run a simple AR scene at 60 FPS for 3-5 minutes straight, before the frame rate starts to drop due to overheating.

Hence, it is best to keep your frame rate as low as possible (30 FPS or less is recommended for WebAR experiences in mobile). As well as designing your AR experience to be playable in small intervals (2-3 minute sessions) while keeping your game as optimized as possible.

HandTracker Settings

Tracker Cam

- Drag your scene's AR Camera in this field

Tracker Mode

- Use this to set either 2D or 3D tracking modes

Track Individual Joints

- Enable this if you want to track all 21 hand landmarks, otherwise, only tracks the wrist (Joint_0)

Hand Thickness

- Controls the thickness of the hand mesh

Default Canonical HandLength

- The default length (from wrist to tip of middle finger) of the canonical hand used by the hand tracker (in Unity units). This value is being calculated in runtime, so no need to edit in almost all cases.

Debug Hands

- Enable this if you want to draw line renderers to visualize the hand joints (in 3D mode) or the hand rectangular bounds (in 2D mode)

Debug Lines Prefab

- The prefab used to draw your debug line renderers

EditorDebugMode

- Since AR only runs in the WebGL build, and not in the Editor, use editor buttons to manually trigger handObject detections and the keyboard for moving the camera

HandObject Settings

Your HandObjects can be customized with several properties for your specific needs.

Hand Index

- The *unique* index of the player's hand being tracked. Up to 4 hands only

Handedness

- Tells us if the handObject is a right hand or a left hand
- Note - since multiple hands may get lost and found continuously, as well as some parts of the hand getting occluded in the process, it is expected that the handedness of a detected handObject may change

3D Settings > Use Hand Mesh

- Enable this flag to use a hand mesh in for your hand object
- Hand meshes are good for hand masks, but for visualizing actual hands, you can consider using a hand animator with finger animations

3D Settings > HandMesh

- The SkinnedMesh renderer of our hand mesh, rigged with hand joints.
- This is set automatically when you enable Use HandMesh. Not recommended to be edited except for advanced users

3D Settings > Use Gestures

- Enable this flag to use finger closeness and gesture detection events

3D Settings > Finger Closeness Thresholds

- Allows you to set the closeness thresholds of each finger
- Threshold (eg. Index Threshold) is the dot product of the root segment (eg. Joint0-5 Segment) and the primary segment (eg. Joint5-6 Segment) as vectors
- Threshold 2 (eg. Index Threshold) is the dot product of the primary segment (eg. Joint5-6 Segment) and the secondary segment (eg. Joint6-7 Segment) as vectors
- You can edit these thresholds to tweak the responsiveness of your finger closeness as well as to improve the detection of your custom gestures.

3D Settings > Debounce Time

- This is the time delay used to filter out noisy joint positions
- Lower values are more responsive but prone to positioning errors
- Higher values are less prone to positioning errors but are less responsive

3D Settings > OnFingerClosed

- Use this UnityEvent <string, bool> to catch finger closeness events
- The string parameter is the finger Id (eg. *ThumbClosed, IndexClosed, MiddleClosed, RingClosed, PinkyClosed*) and the boolean parameter is the closeness (true = closed, false = open).
- We recommend hook this directly to your Animator.SetBool function to animate your hand models.

3D Settings > Custom Gestures

- Use this to detect your custom gesture
- Use *GestureName* to name your gesture

- Use *FingerValues* checkboxes to set finger states for your gestures. Note: this is using openness eg. (Hands Open => (true, true, true, true, true), Hands Closed => (false, false, false, false, false))
- Use *OnDetected* to raise an event when your custom gesture is detected
- Use *OnLost* to raise an event when your custom gesture is lost

3D Settings > OnAllGesturesLost

- Used to raise an event when all your custom gesture are lost

3D Settings > DebugShowCameraRaycast

- Enable this to show a raycast from the camera to each of your hand joints

3D Settings > DebugShowAxes

- Enable this to show axis frames to each of your hand joints

2D Settings > Z Distance

- The fixed distance where the handObject is positioned when using 2D Tracking Mode

2D Settings > OnPosXChanged

- Use this event to get the horizontal screen space position (0 => left side of the screen, 1 = right side of the screen) of the handObject

2D Settings > OnPosYChanged

- Use this event to get the vertical screen space position (0 => bottom side of the screen, 1 = top side of the screen) of the handObject

2D Settings > OnPosChanged

- Use this event to get the screen space position (Vector2) of the handObject

ARCamera Settings

Video Plane Modes

- **NONE** - does not render a video plane inside Unity. Instead, the unity canvas is transparent and the camera image is rendered in javascript. This is the fastest video plane mode, but as a tradeoff, it can have rendering issues with transparency, and does not support Post Processing and HDR. Capturing screenshots while using this mode results in a black camera feed.
- **TEXTURE_PTR** - renders a video plane inside Unity to support Post Processing, HDR, and screenshot capture. This mode can have slightly performance impact. This mode is not available in the free version

- **DATAURLTEXTURE (DEPRECATED)** - renders a video plane inside Unity to support Post Processing, HDR, and screenshot capture. This mode uses dataURL textures to pass image data from javascript to Unity and can be very slow, especially on older devices.
- **WEBCAMTEXTURE (DEPRECATED)** - renders a video plane inside Unity to support Post Processing, HDR, and screenshot capture. This mode uses Unity's webcamtexture to get the image data. This method is unreliable and may have issues on some devices/browsers/OS versions.

Video Plane Mat
- The material to be used to render your video plane

Video Distance
- The distance between the ARCamera and the video plane

Unpause Pause On Enable Disable:
- If this is enabled, UnpauseCamera() and PauseCamera() are automatically called when ARCamera gameObject is activated or deactivated

Pause On Destroy:
- If this is enabled, PauseCamera() is automatically called when ARCamera gameObject or scene is destroyed

Pause On Application Lose Focus:
- If this is enabled, PauseCamera() is automatically called when the browser tab goes to the background. This is useful to keep the webcam alive when switching tabs/apps while the experience is running.

OnResized:
- Unity Event called when video camera dimensions are resized. This gets called when device orientation changes. Note that resizing the browser window does not necessarily trigger this event.

Resize Delay:
- The delay between resize events, where the ARCamera starts re-computing for the new size. Later versions of iOS (17) has a relatively long resize delay (bug) which hinders resize events in Unity. If so, try using a longer resize delay.

OnCameraImageFlipped:
- This gets called when the flipping/toggling the camera from front to back or vice versa is successful (using ARCamera.FlipCamera function).

OnCameraOrientationChanged:

- This gets called when the device is rotated from portrait to landscape, or vice versa.

Tracker - Scripting API

You can also use these API calls to control the WebGL tracker:

HandTracker.StartTracker

Manually start the tracker. This is useful when you need to restart the tracker after stopping it.

HandTracker.StopTracker

Manually stop your tracker. This is useful when you want to display non-AR related content in your scene.

ARCamera.PauseCamera

Temporarily pauses the camera.
Note: currently tracked objects will freeze in place if tracker is not stopped before calling this method.

ARCamera.UnpauseCamera

Resumes the camera.

ARCamera.FlipCamera

Toggle the camera from front to back, or vice versa.

Other Features:

ARShadowShader

Easily add shadow planes to your image target experiences

ChromaCutoutShader

Easily integrate green-screen/chroma videos to your AR experiences.

SyncVideoSound

Play Videoplayer and AudioSource simultaneously (because video with sound does not play in iOS Safari)

ScreenshotManager.GetScreenshot ()

Capture and display screenshots and allow users to save them to gallery or share to their social media apps

TextureDownloader.DownloadTexture (Texture2D texture)

Allows the users to download textures as a png or jpeg file. Useful for sharing image targets directly from the web browser for printing.
Note: Textures need to be **Uncompressed** and **Read/Write** enabled.

Current Limitations

Frame rate and Performance

Since WebGL supports a wide variety of devices and web browsers, the actual frame rate will be determined by the processing capability of the user device. During testing we have achieved 45-55 fps on newer iPhones (iPhone X, iPhone 14) while 12-24 fps on older devices (such as iPhone 8 and Samsung S8)

CV Source code

Source code of the CV module is not included by default, mainly because we cannot yet guarantee and provide support on its functionality and performance once customized by other developers.

Editor Simulation

Unfortunately, we do not yet have any means to test AR functionality in the Unity Editor. However, we plan to implement this feature in future versions.

FAQ and General Questions

Camera does not open when I host in my website

- Make sure you are hosting on your server with https enabled. Otherwise, access to the webcam will be blocked due to security reasons.

Unity loading bar is stuck at 90%

- This is usually caused by your WebGL compression. You can set **Player Settings>Publishing Settings>Compression Format** to **Disabled.** You can also compress your build but you have to ensure that gzip(.gz) or brotli(.br) is enabled in your hosting server.

Uncaught TypeError: Cannot read properties of undefined (reading 'FOV')

- Check your **Player Settings>Resolution and Presentation** and make sure that have selected the correct WebGL template.

Works in localhost, but webcam does not open when build is hosted

- Please make sure that you're hosting with SSL(using https).

Unity doesn't start - stuck in loading screen/white screen

- If you are seeing this error or something similar
  Uncaught SyntaxError: Invalid or unexpected token (at WebGL.framework.js.br:1:2)
  Try building without compression in PlayerSettings>Publishing Settings

Known Issues:

Visit the #bug-reports channel in our discord

Need Help?

Visit the #support-channel in our discord

Change Notes:

Version 1.0.1

• [ADDED] 2D Tracking Mode
• [ADDED] Hand Animator - Finger Closeness Events
• [ADDED] Hand Animator - Custom Gestures (Experimental)
• [ADDED] Demo scenes for 2D Tracking, Virtual Wearables, Hand Animator, and Hand Gestures
• [ADDED] Improved documentation
• [FIXED] Bug where hands with same handedness swap a lot
• [FIXED] Bug where hand is not properly rendered Issue when using the front camera
• Major refactoring for improved workflow and property drawers

Version 1.0.0

- First release