Imagine WebAR Image Tracker

Version:	1.4.2
WebGL Demos:	https://imagetracker.imaginerealities.com.au/demo https://imagetracker.imaginerealities.com.au/urp-game
Contact Support:	https://imaginerealities.com.au/contact-support
Publisher's Website:	https://imaginerealities.com.au
Unity Forums Thread:	https://forum.unity.com/threads/released-imagine-webar-image-tracker-for-unity-webgl.1382748/
Discord Community:	https://discord.gg/ypNARJJEbB

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.

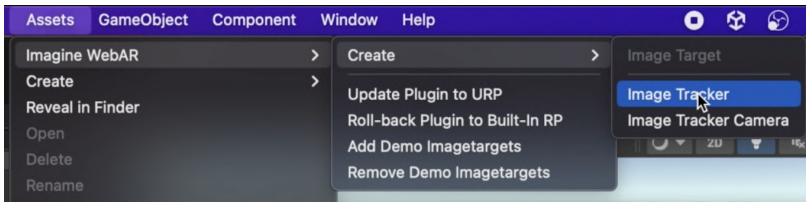
Render Pipelines supported are Built-In RP and URP. Though, some rendering features are not supported (see **Current Limitations**) or still experimental.

The plugin consists mainly of two modules: [1] a computer-vision (CV) module, written in Javascript, which uses Natural-Feature tracking. This method disregards the need of any markers, and allows developers to anchor 3D objects directly into any image (Given that this image has enough “features”. See **WebAR Best Practices**). [2] A Unity Editor module which provides the tools necessary to create “Image targets” and set up your AR Scene in Unity.

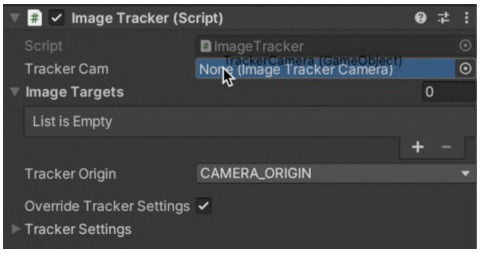
Setting up your AR Scene in Unity

A simple AR scene can be set up in a few minutes. Tutorial video can be found here.
To set up your first scene:

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



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



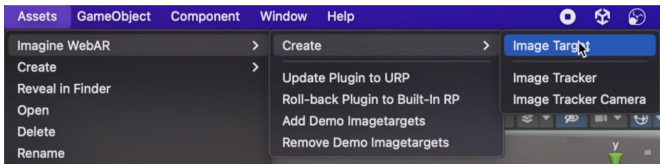
4.) Now we are ready to add **Image Targets** to our tracker.

Adding Image Targets to your Tracker (Paid Version)

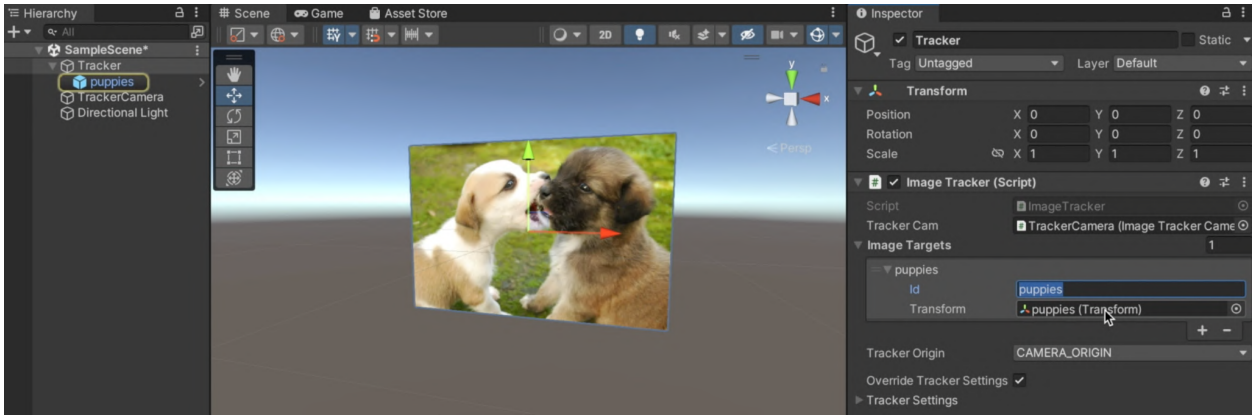
Image Targets are images, which are detected by the tracker, and overlaid with 3D objects. Any image can be used as an image target however, tracking performance will greatly depend on the amount of “features” in the image. Images with high pixel contrasts are strong targets, while images with smooth gradients perform poorly or not at all. Please see **WebAR Best Practices** for more information.

If you purchased a **Paid** version of the asset, you'll be able to create custom image targets. Follow the steps below to add Image Targets to your tracker (Paid Version):

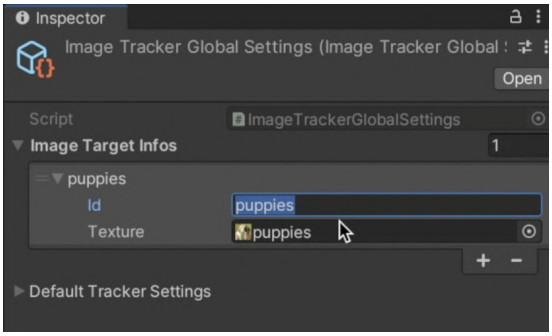
1.) Drag/import your image in Unity.
2.) Select this image in the Unity project window, and go to **Assets>Imagine WebAR>Create>Image Target**



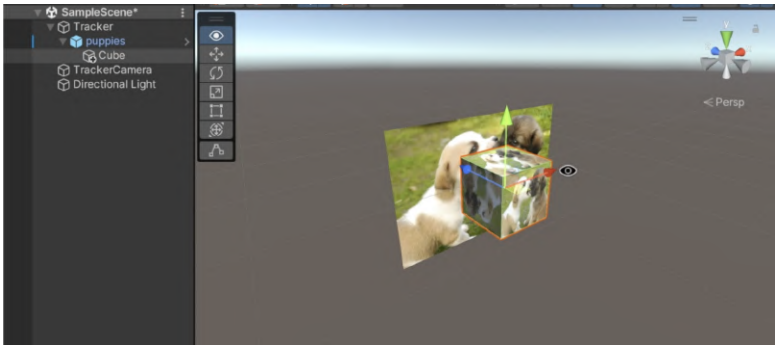
3.) Save your new Image Target prefab in your preferred directory.
4.) If you have your AR scene open, this Image Target prefab is automatically added to your hierarchy and to your Image Tracker. Otherwise, simply drag this prefab in your scene, select your Image Tracker - add a new element in the **Image Targets** List property with your Image Target's name and transform.



5.) <u>Important:</u> To check if your Image Target is set up correctly, go to **Assets/Imagine/ImageTracker/Resources/ImageTrackerGlobalSettings.asset** and you should see your new Image Target in **ImageTargetInfos**. And the Id should match with the one you have in your Image Target Tracker.



6.) Finally, drag in your game objects in your new Image Target object.



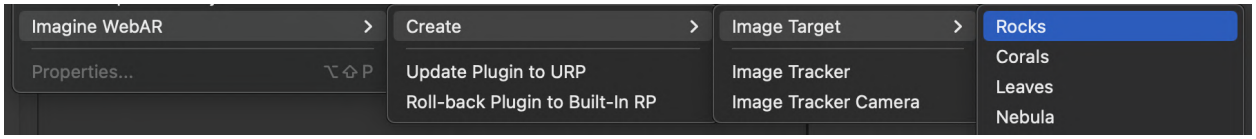
Adding Image Targets to your Tracker (Free Version)

The **Free** version does not allow you to create your own image targets, but it comes with several templates that you can use. There is also an allotted space at the center where you can place your logo/branding.

You can also submit your image target designs in our Discord. If it gets enough votes/likes, then it will have a chance to be included as a new template in the next asset release.

Follow the steps below to add Image Targets to your tracker (Free Version):

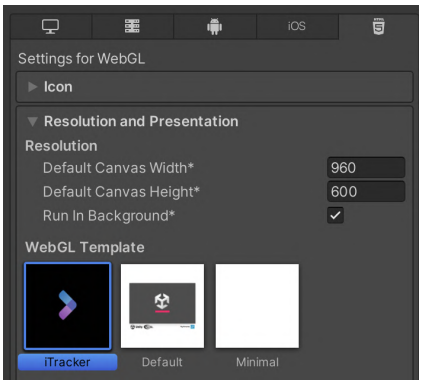
1.) Go to **Assets>Imagine WebAR>Create>ImageTarget** and choose your preferred template. The ImageTarget prefab should be automatically created in your scene.



2.) Drag in your game objects as a child of your new Image Target object.

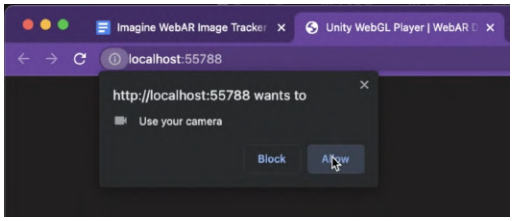
Building your AR Scene

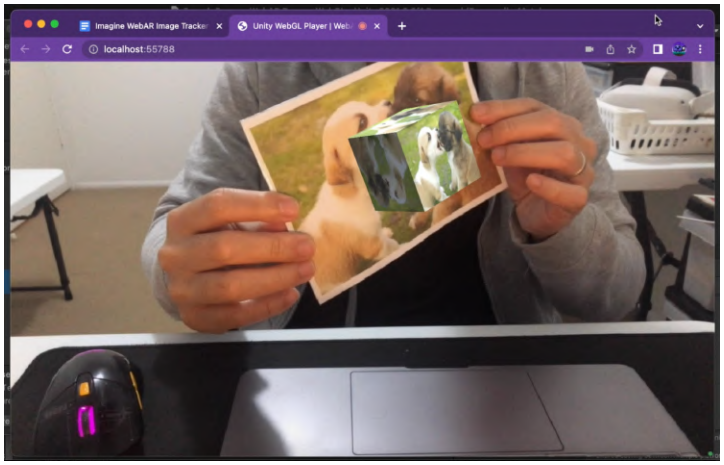
1.) Go to **File>Build Settings,** switch to the **WebGL** platform if needed and add your scene in the build. Then select **iTracker** template in **Player Settings>Resolution and Presentation>WebGL Template**



2.) Build and Run your project.

3.) Allow access to your device camera and scan your printed image target. You should see the game objects anchored in your image in AR.



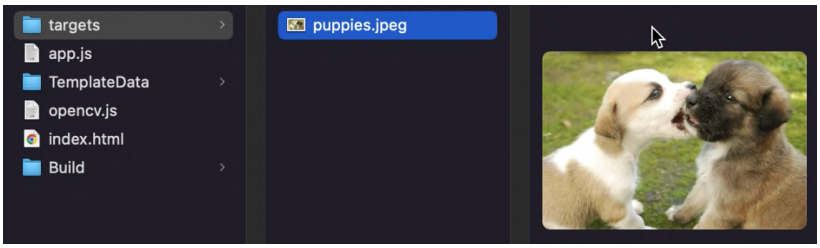


For custom Image targets (Paid Version), you can verify that your Image Targets are properly included in your build, by going to your build folder and opening **index.html** in Visual Studio. You should see the below html line:

```
<imagetarget id='YOUR_ID' src='targets/YOUR_FILE.png'></imagetarget>
```

```
<!--IMAGETARGETS-->
    <imagetarget id='puppies' src='targets/puppies.jpeg'></imagetarget>
```

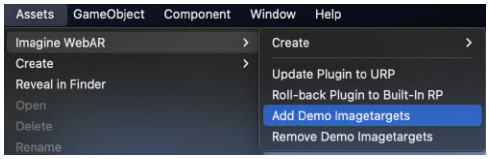
for each of your Image Targets. You can also see that all your images are copied from the Unity folder to a folder called **/targets** in your build folder.

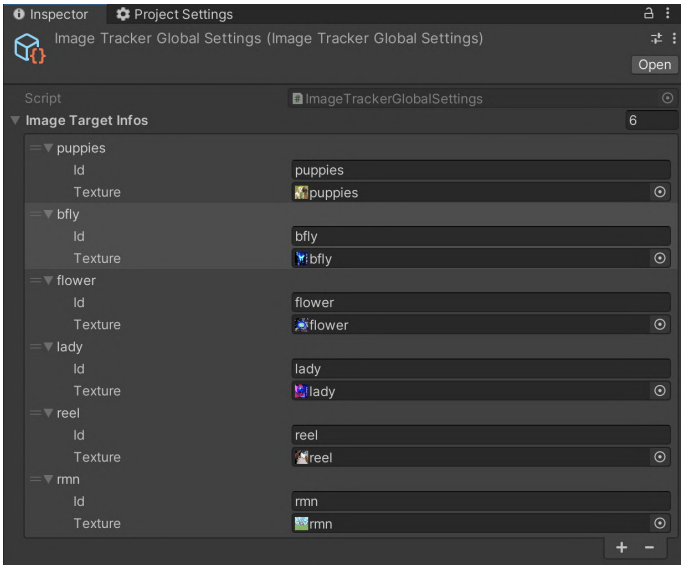


Building Demo Samples

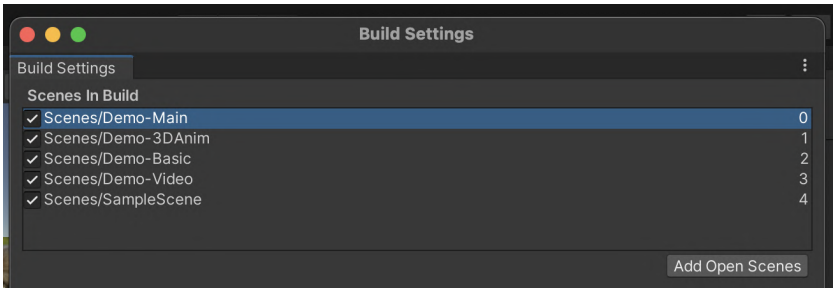
This plugin comes with demo samples which you can build and test. Follow the steps below to do this:

1.) Go to **Assets>Imagine WebAR>Add Demo Image Targets**. You should be able to see new Image Targets in your ImageTrackerGlobalSettings object. This step is not required in the **Free** version.





2.) Make sure to include the demo scenes (**Assets>Imagine>Scenes**) in your Build Settings, with **Demo-Main** as your first scene.

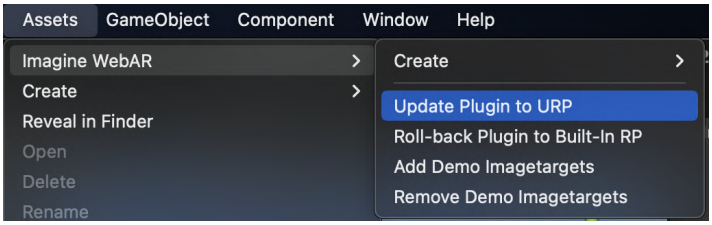


3.) Go to **Player Settings,** Select **WebAR** template in **Resolution and Presentation>WebGL Template**
4.) Build and Run your project.

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: The plugin does not fully support all URP features - Camera HDR and Post-Processing are disabled by default. This is because the plugin is rendering the video background in javascript while the 3D objects are rendered by Unity WebGL with a transparent background. Turning these features on breaks the transparency of the canvas and the background is rendered black.
This can be partially resolved by enabling **Use Webcam Texture (Experimental)** in your TrackerCamera.

Hosting your AR experience for free in AWS

Watch the tutorial video here: Video coming soon

Hosting your AR experience for free in Google Firebase

Watch the tutorial video here: Video coming soon

WebAR Best Practices

Selecting the right Image Target

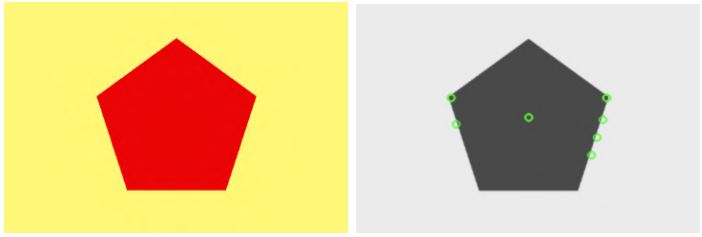
Several factors can affect the detection and tracking performance of your Image Target. An ideal Image Target is described by the following:

- Rich and well-distributed details across the entire image
- Excellent contrast between the bright and dark regions of the image
- No repetitive or symmetrical patterns

What are Features?

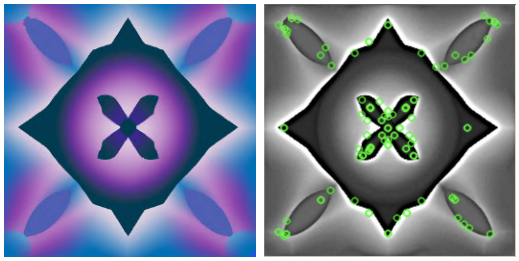
Features are sharp details such as corners and contrasting pixels in textures. Tracking images with more features are more robust to noise and jitter.

A great example of a feature-rich Image Target



A poor Image Target with very few trackable features

A poor Image Target due to lack of contrast



An Image Target with symmetrical or repeating patterns can “confuse” the tracker

Physical factors affecting tracker quality

Physical factors such as print media and lighting also play a vital role in the effectiveness of the tracker.

- Image Targets printed in glossy or reflective surfaces will have tracker quality issues on some angles. It is best to print your images in matte, whenever possible.
- Environment should be well lit. The lighting conditions can easily affect how the camera sees the image thus affecting tracker quality.
- Image Target printout should be rigid. Creased or bent targets degrade the quality or the tracker.

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 (including tracker)

FPS and Overheating

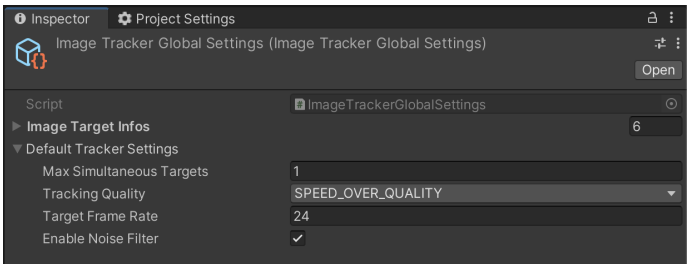
It is also very important to consider the impact of frame rate to the tendency of the device to heat-up. AR experiences, in general, consume more resources (such as camera and calculations) than standard games, thus putting more strain on device hardware and causing them to heat up, especially on high 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.

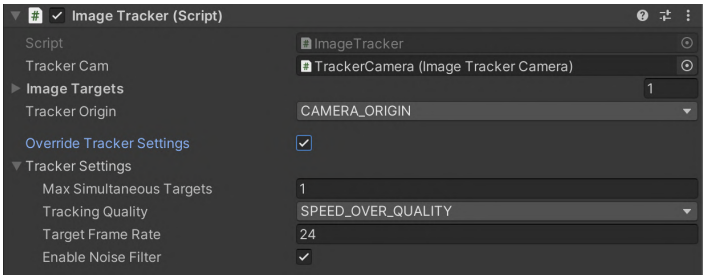
Hence, it is best to keep your frame rate as low as possible (30 FPS or less is recommended for WebAR experiences in mobile). As well as designing your AR experience to be playable in small intervals (2-3 minute sessions) while keeping your game as optimized as possible.

Tracker Settings

The tracker can be customized with a couple of properties. And these default settings can be found in **Assets>Imagine>Resources>Image Tracker Global Settings.asset.**



They can also be overridden on a per scene basis, by enabling the **Override Tracker Settings** in your Image Tracker object.



Tracker Origin

- Use **CAMERA_ORIGIN** to position your image targets relative to the camera at the origin.
- Use **FIRST_TARGET_ORIGIN** to position your camera relative to the first detected Image Target. This is useful for Image Target setups which are sensitive to orientation such as when using gravity, trail renderers or world particle systems.

Max Simultaneous Targets

- Use this setting to set the number of image targets that are allowed to be tracked simultaneously.
- Note: In theory, there is no limit in the number of Image Targets tracked simultaneously , but doing so can decrease the frame rate of your application.

Tracking Quality

- Use **SPEED_OVER_QUALITY**, to get the highest frame rate at the expense of longer detection times and introducing tracking noise.
- Use **BALANCED**, to accommodate for both speed and quality.
- Use **QUALITY_OVER_SPEED**, to get the most stable tracking at the expense of frame rate.
- Note: Tracker performance is still greatly determined by the end user device - meaning **QUALITY_OVER_SPEED** can run smoothly on newer devices, while **SPEED_OVER_QUALITY** can still run slow on older/low-tier devices.

Target Frame Rate

- Use this setting to let the tracker know your desired framerate (Normally this is between 30fps for mobile experiences).
- Note: Actual frame rate is still determined by the processing power of the end user device.

Advanced Settings > Max Frame Length

- Higher values will increase tracking quality and decrease jitters, but decreases frame rate

Advanced Settings > Max Frame Area

- Higher values will increase tracking quality and detectability, but decreases frame rate

Advanced Settings > Detect Interval

- Lower values will speed up detection time, but significantly decreases frame rate

Advanced Settings > Tracked Points

- Higher values will improve stability, but can decrease frame rate

Debug Mode:

- When this flag is enabled, you can press “I” on your keyboard to visualize the detected features in your image targets.

On Image Found:

- Subscribe to this UnityEvent to invoke callbacks when a specific image target is tracked.

On Image Lost:

- Subscribe to this UnityEvent to invoke callbacks when a specific image target is untracked.

Tracker - Scripting API

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

ImageTracker.StartTracker

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

ImageTracker.StopTracker

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

ImageTrackerCamera.PauseCamera

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

ImageTrackerCamera.UnpauseCamera

Resumes the camera.

ImageTracker.IsImageTargetTracked (string id)

Use this method to check if a specific image target is currently being tracked.

Other Features:

TextureDownloader.DownloadTexture (Texture2D texture)

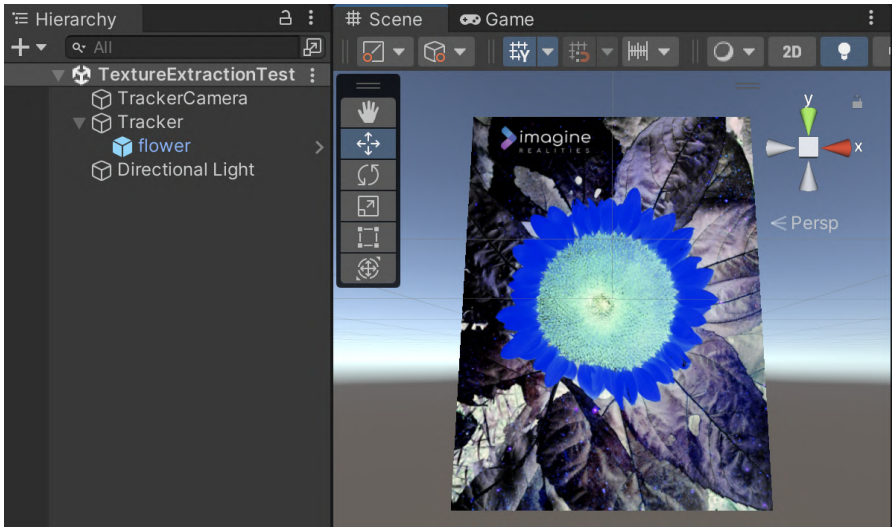
Allows the users to download textures as a png or jpeg file. Useful for sharing image targets directly from the web browser for printing.
Note: Textures need to be **Uncompressed** and **Read/Write** enabled.

Extracting Textures From the Camera

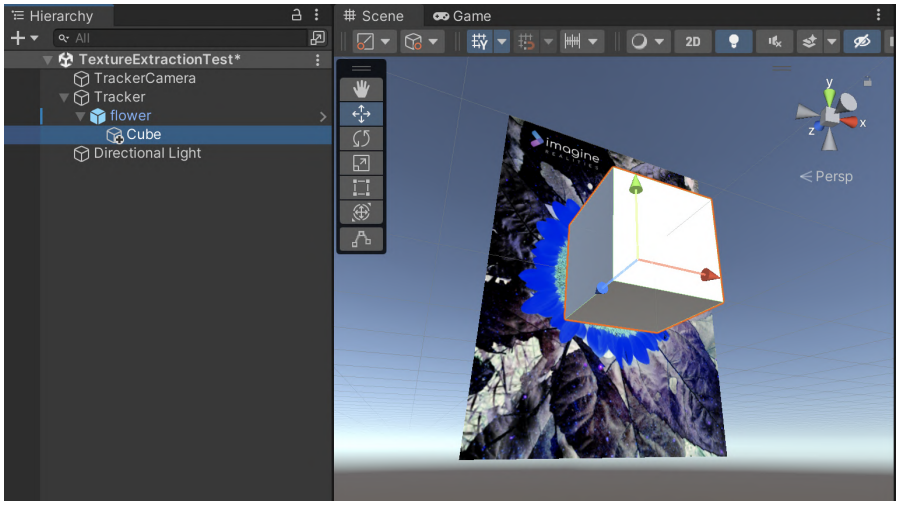
The texture extraction feature allows you to get the image target texture from the camera feed, remove perspective distortions, and use it as a texture.

Some AR experiences, such as interactive coloring books, require access to the camera texture in real-time. Since version 1.4.2, Warped Texture Extraction is now available in the plugin. Follow these steps to quickly setup Texture Extraction

1.) Setup your AR scene by creating an Image Tracker, and AR Camera, and your Image target

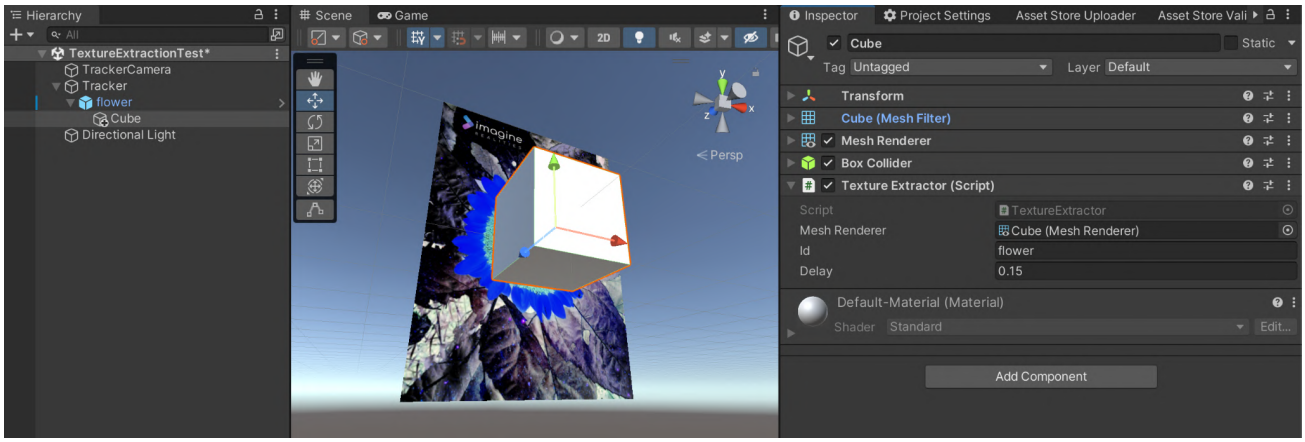


2.) Add a gameobject as a parent of your Image Target. In our case, let's use a Unity cube.



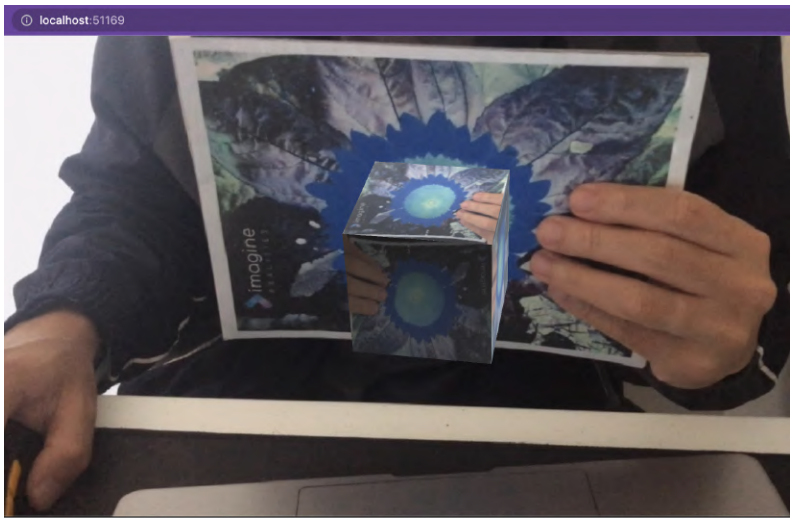
3.) Attach a **TextureExtractor** Component to your gameObject.

4.) Drag the cube to the **MeshRenderer** component and set the **Id** to match your image target's id



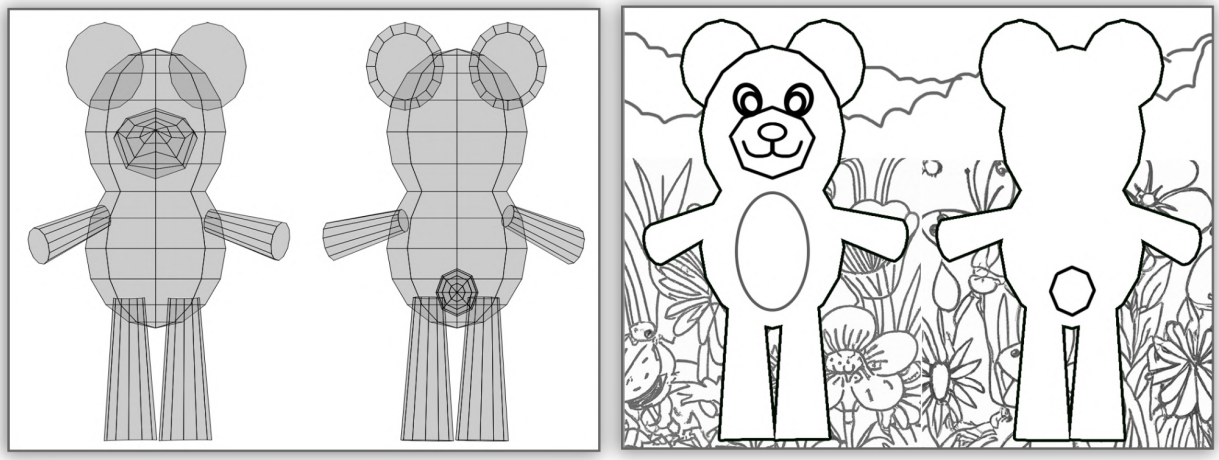
5.) You can also optionally set the **Delay** property. But note that extracting textures every frame can significantly decrease your frame rate.

6.) Then Build and Run your project to see how it works!



Interactive Coloring Book AR Experience

To create an interactive coloring book experience, you will need to design a special image target that exactly matches the UV map of your 3D character. Then use the TextureExtractor to map the camera texture to your model.



It is also important to add background elements other than the model outline as these will help with the tracking of your image target - since the user will alter some parts of the image.

Current Limitations

Transparency

Transparent pixels directly in front of the video background are culled by default. This is currently the biggest limitation of the tracker. This will be resolved once Unity properly supports transparent WebGL canvas rendering.

There is currently an experimental workaround by enabling **Use Webcam Texture** flag in ImageTrackerCamera.

Update: This issue seems to have been fixed as of testing on Unity 2021.3.0f1

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.

URP

As mentioned above, some URP features are disabled by default. Camera HDR and Post-Processing overrides the depth buffer thus disabling the transparency of our WebGL canvas.

There is currently an experimental workaround by enabling **Use Webcam Texture** flag in ImageTrackerCamera.

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 **WebAR** WebGL template.

Image not getting detected

- Double check if your build folder includes a folder called **/targets** and that your image files are included. Also check your **index.html** if your image target is included as

```
<imagetarget id='YOUR_ID' src='targets/YOUR_FILE.png'></imagetarget>
```

- Double check if the Image Target is registered in your **ImageTracker** as well as in **ImageTrackerGlobalSettings** and that their ids are matching.
- Check if your Image Target follows WebAR best practices - good amount of features, high contrast and non-symmetrical etc. (See **WebAR Best Practices** for more information)
- Make sure your target images does not have any transparent pixels

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

Can I still use the plugin with irregular/circular/non-rectangular image targets/stickers?

- Yes, you can simply replace all your transparent pixels with plain black or white. Just make sure it still follows **WebAR best practices** above

Known Issues:

Visit the #bug-reports channel in our discord

Change Notes:

Version 1.4.2

• [ADDED] Warped Texture Extraction Feature - Image targets can now be extracted from the camera frame and loaded as a Texture2D in Unity
• [FIXED] Fixed tracker error when device is flipped while tracking a target
• [FIXED] Fixed CORS error when imagetargets are loaded from a different domain
• [FIXED] Fixed issue where screen flashes black when resizing html elements
• [FIXED] Debug Image Target feature points not working properly
• Tracker is now more robust to occlusion
• Tracker is now more robust to skewing and drift caused by high-motion
• Added **Detect Interval, MaxFrameLength, MaxFrameArea** properties in Image Tracker Advanced Settings

Version 1.4.1

• [FIXED] Fixed javascript error introduced in 1.4.0

Version 1.4.0

• [FIXED] Fixed a very rare issue where camera feed goes very dark
• Added console logs when image target is lost/found
• Minor optimizations to decrease plugin size by 30%

Version 1.3.3

• [FIXED] Fixed UI cropping issues for some mobile browsers such as iOS Safari
• [FIXED] Fixed an intermittent issue in where targets stop being detected after getting detected once (usually occurs in scenes with multiple image targets)
• Added some basic Editor debugging features - Found and Lost events, as well as keyboard controls for camera movement
• *Breaking changes to your custom index.html (To upgrade, see index.html.132-133.diff)*

Version 1.3.2

• [FIXED] Fix compile error for URP
• Added AR Shadow shader (for URP)

Version 1.3.1

• [FIXED] Resolved an issue where tracker stops working if image target is detected upon camera initialization
• Added Experimental DataUrlTextures (as an alternative to WebcamTextures) for displaying the camera feed inside Unity

• Added GetWebGLCameraFrame API for extracting the camera feed as a Texture2D

Version 1.3.0
• Overall improvement of our tracker algorithm which includes -
• Significant noise and jitter reduction in both near and far distances
• Significant boost in frames per second (reaching 60fps in high to mid tier devices, readching 30fps to low tier devices)
• Significant quality improvement for simultaneous target tracking
• Deprecated Noise Filtering and Stability settings
• Added back default Unity loading screen
• Added AR Shadow shader (for Built-In RP)

Version 1.2.3
• [FIXED] Resolved an issue where the camera is not properly initialized if the device have multiple back cameras
• Fixed a bug where the editor framerate setting is not properly being set
• [URP] Minor improvements, bug fixes and error handling in Universal Render Pipeline

Version 1.2.2
• [FIXED] Resolved a device language issue causing javascript errors for users in specific regions

Version 1.2.1
• [WeChat] Added fallback button when the webcam video failed to play automatically due to browser restrictions
• Fixed a race-condition bug where the camera's field of view gets initialized to zero

Version 1.2.0
• Improved general tracking quality. More resistant to camera motion especially at very close distances. Self-correction when image reappears after partly being obscured.
• Quality and Optimization Improvement in simultaneous tracked images
• Performance boosts and Optimizations
• Added new **Advanced>Detectability** property slider in ImageTracker Unity inspector
• Added new **Advanced>Tracked Points** property
• Replaced **EnableNoiseFilter** flag with **Advanced>Noise Filtering** settings
• Added new **Advanced>Stability** property
• Removed **ImproveMatches** flag. This is now done by default.
• Minor changes to the plugin's folder structure

Version 1.1.0
• Significantly reduced noise and jitter, especially when image target is steady (at a slight expense of maximum tracking distance and motion)
• Minor optimisations to boost the frame rate. (reaching up to 60FPS on mid/low tier devices)
• Minor UI improvements on the ImageTracker Unity inspector
• Added **OnImageFound** and **OnImageLost** UnityEvents

• Added **IsImageTracked** API to check if an image is currently being tracked

Version 1.0.5
• First release