UI Toolkit 3D Object Image - Manual

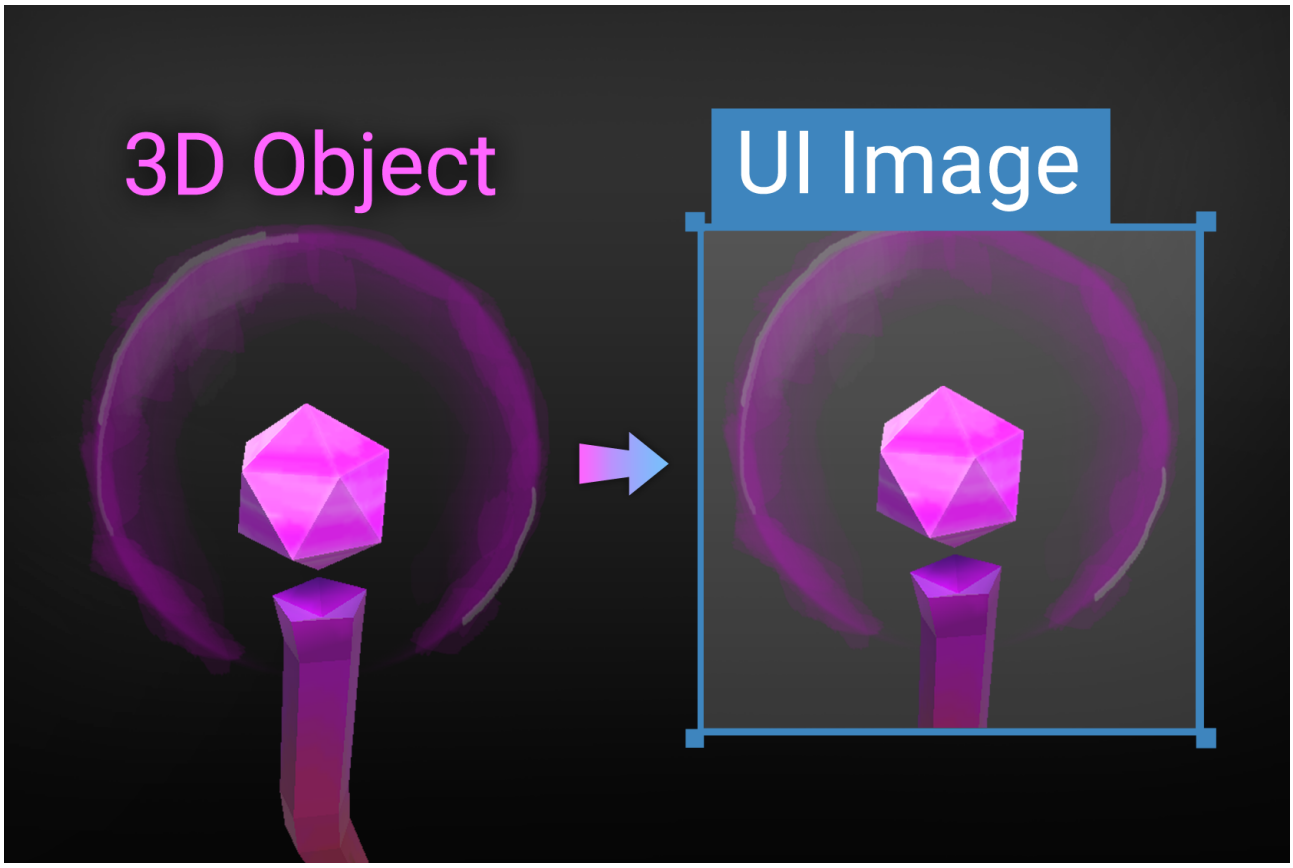


Table of contents

Requirements & Setup

Requirements

Unity 2021.3 or higher is required since that is when Unity added the UI Toolkit Module for runtime use. If you can, please upgrade to the highest LTS version of Unity. The newer the version the less „glitches" the UI Toolkit has.

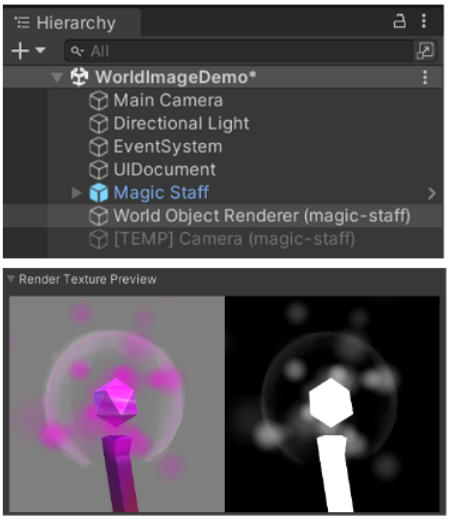
Please keep in mind that UI Toolkit as a whole is still a work in progress and not quite ready for prime time. Unity itself still recommends using UGUI instead of UI Toolkit for runtime applications (source).

How to create a World Image

Before we start you have to understand that the World Image requires a scene object called „World Object Renderer“ to work.

World Image (in UI Builder) + **World ObjectRenderer** (in Scene, renders the 3D object into a render texture that can be used in the UI Toolkit):

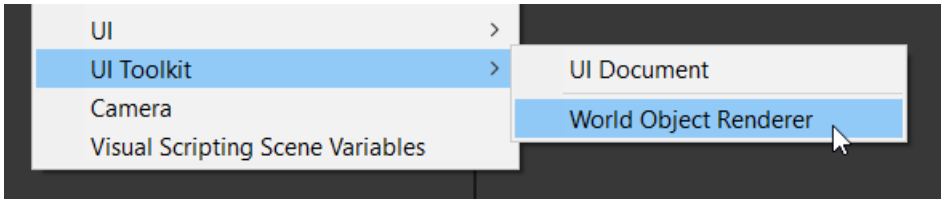
A 3D Object .. → is rendered into a texture .. → which is used in the UI.

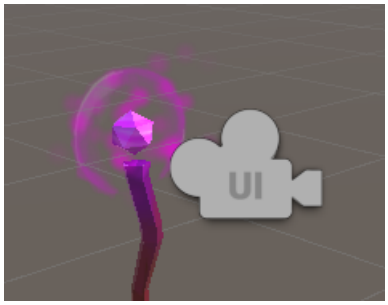


Let's do this step by step.

Step #1:

Create the „World Object Renderer“ in the scene via **Right-Click > UI > World Image (uGUI)**:

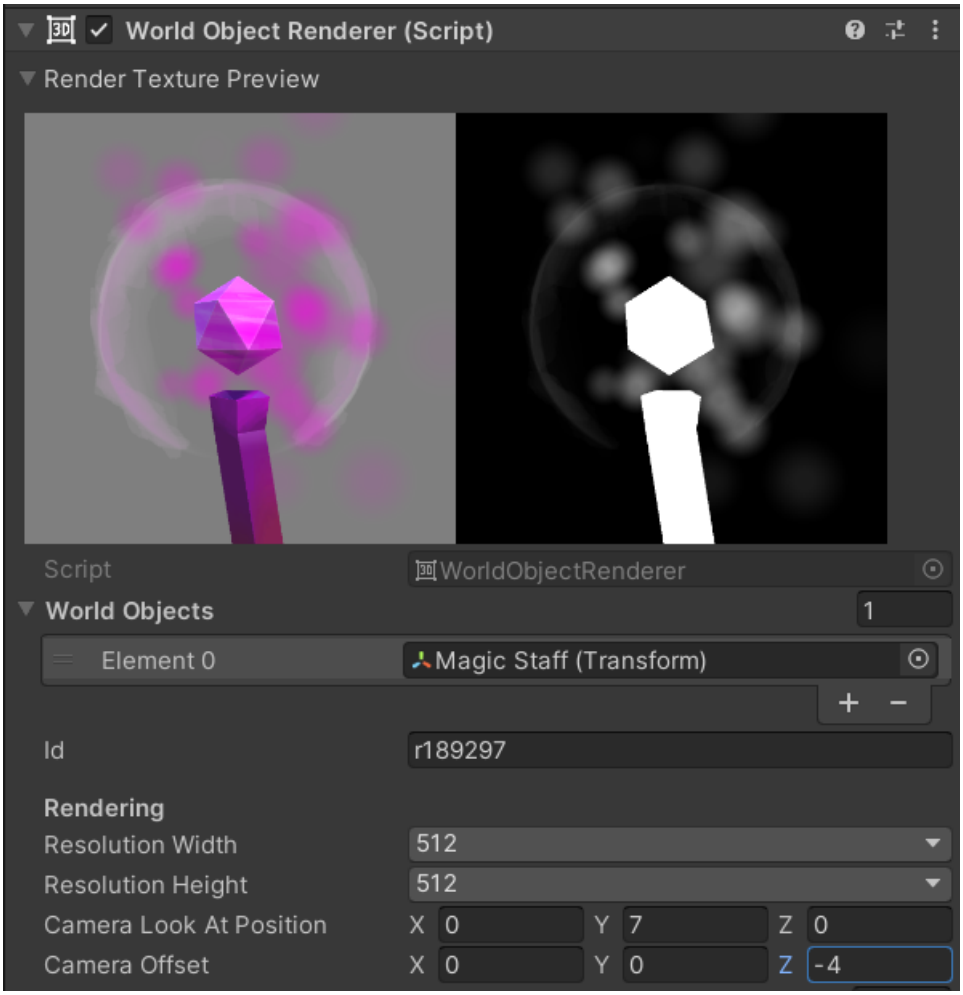




You may have noticed that the World Object Renderer also generates a temporary camera. It is used to render in to a render texture which is then used as the source for the UI image.

Step #2:

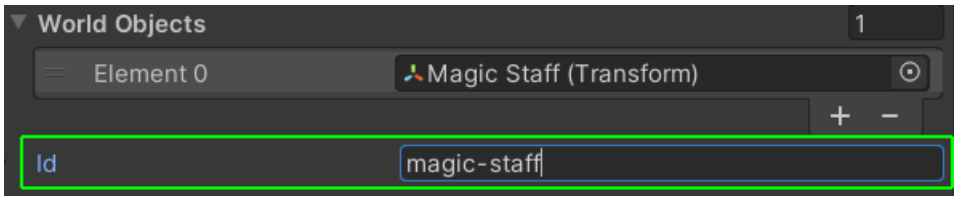
Once you have done that then you can drag in your object(s) into the „WorldObject“ list and tweak the options. Use the Render Preview to teak the camera position.



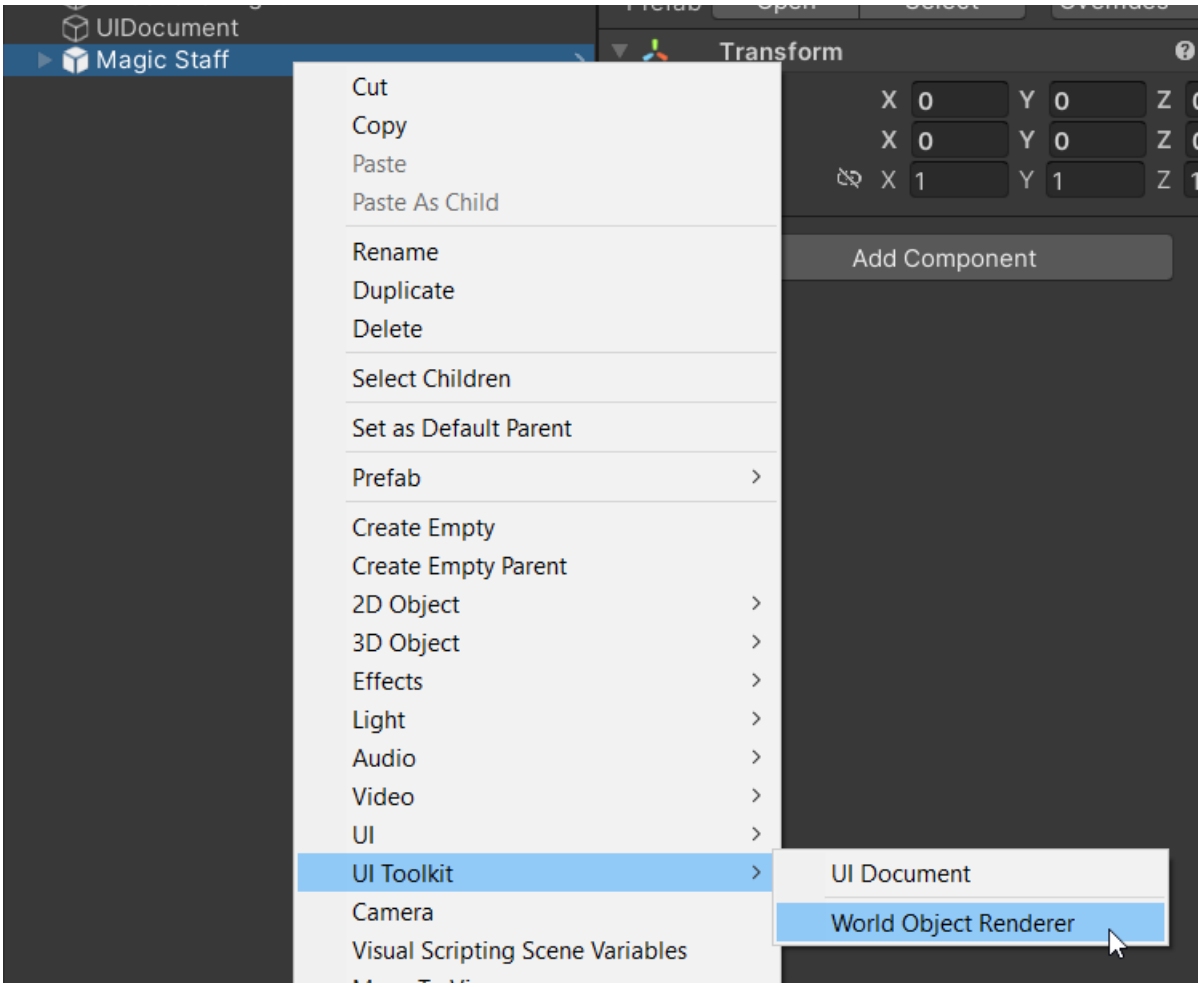
HINT: If you do not add any world object then the image will simply act as a camera into your 3D world. Use „CameraLookAtOffset“ to position the camera in the world.

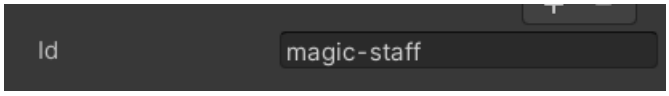
Step #3:

Choose an ID for your object renderer (by default a random ID is chosen).



HINT: If you right-click on a game object while creating the renderer then the id will be auto-generated based on the name of that objet. For example „Magic Staff“ becomes „magic-staff“:

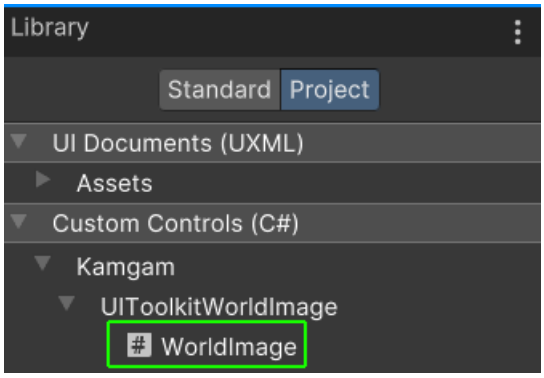




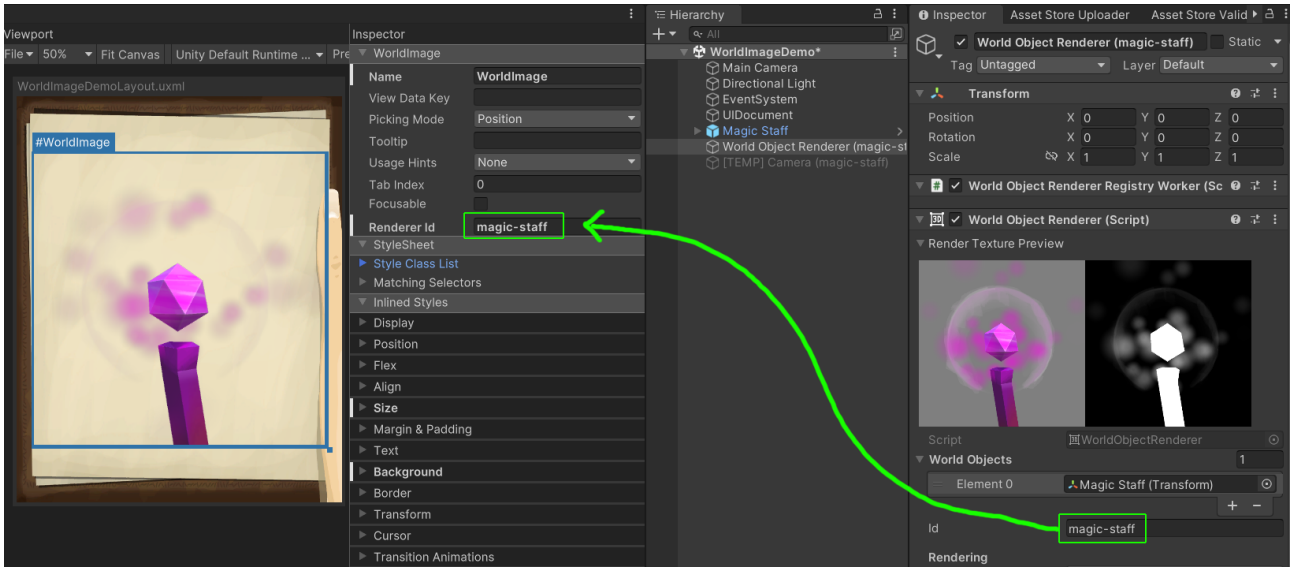
Step #4:

Finally we can add a „World Image“ in the UI Builder.

You can find it in the UI Builder Library under: **Project > Custom Controls (C#) > Kamgam > UIToolkitWorkdImage > WorldImage**



Once the image is added all you have to do is put in the ID of your object renderer. After that you should get the rendered image immediately:



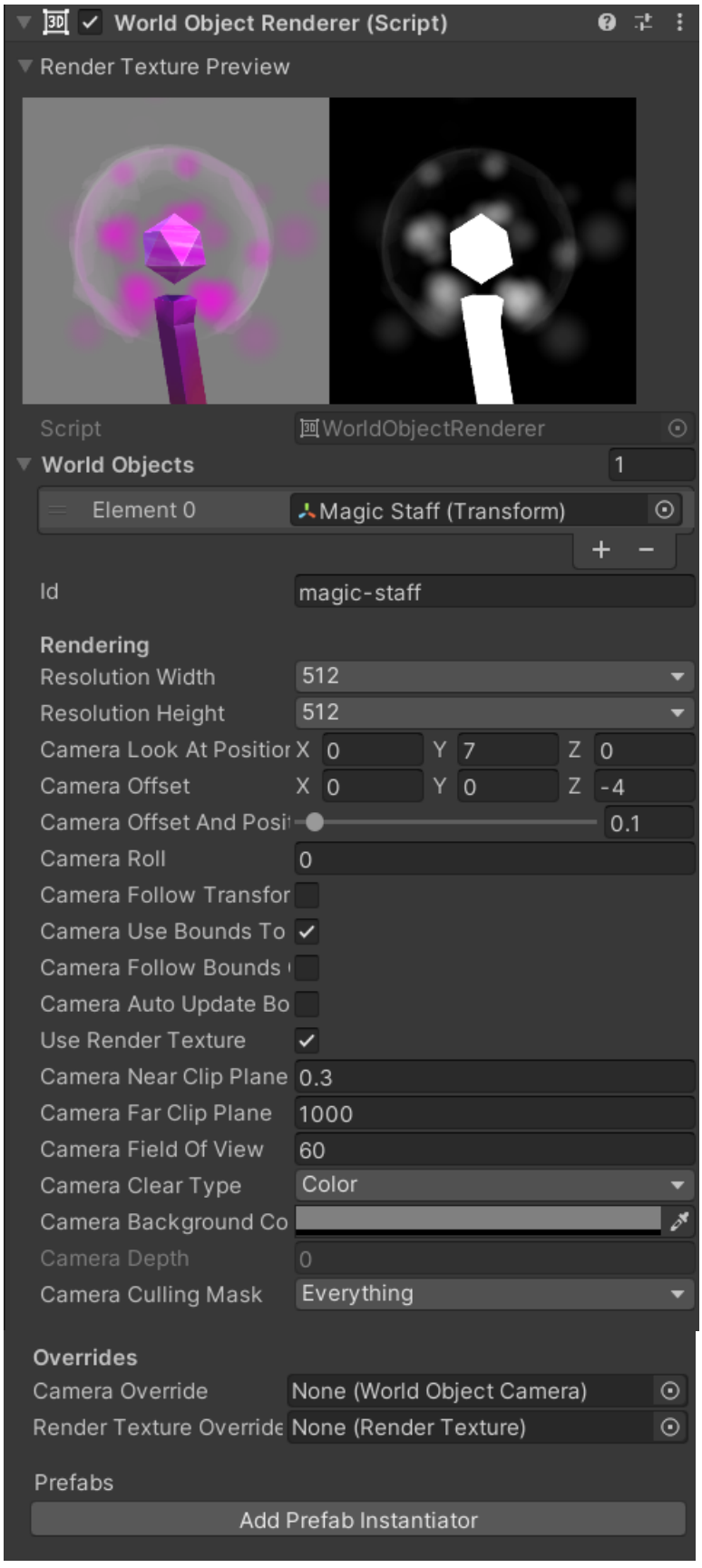
And that's it.

HINT: Multiple images can show the same renderer output (simply set the all their ids to the same renderer id).

INFO: You may wonder why the WorldImage in the UI Builder has only one custom attribute (Renderer Id). That's because compared to the default inspector the UI Builder lacks some features (object references and buttons for example).
I know this is has changed in recent Unity versions so I may change this in the future (once the UI Builder has matured enough).

Renderer Properties

Since the attributes in the UI Builder are not yet ready to receive object references in all Unity versions the properties of the image are all configured on the World Object Renderer in the scene.



Render Texture Preview

This is the preview of the rendered texture of this renderer. If you change the properties of the renderer you will see it reflected here.

HINT: If it's all grey in the beginning then try a bigger CameraOffset.z value (the camera is probably inside the 3D object).

World Objects

This is the list of objects that will be used to position the camera that renders the image. Usually the camera will center of the FIRST object of the list (the objects transform.position). However you can also make it center on the bounding box of all objects (more on that below).

If you leave it empty then the camera will be placed at the world position set in „Camera Look At Position" (more on that below).

Resolution Width / Height

Since the texture for the image is taken from a RenderTexture we have to define the resolution of the texture. You can choose from a variety of resolutions.

NOTICE: You may winder why you can not enter the resolution freely. The reason is that the render textures are actually pooled (see RenderTexturePool in code). In order for the pool to work we have to group the textures by certain criteria. One of these is the resolution. That's why one a few select resolutions are allowed. These are all divisible by 2 which is beneficial for some graphics calculations.

Camera Look At Position / Camera Offset

The image of an object is rendered by a non-persistent camera (notice the grey UI camera).

There are two parameters that define where the camera is positioned and where it is looking at.

The „Camera Look At Position" defines the position of the camera. The offset is then added to the position and with it the look direction is defined.

Camera Osset And Poisition Mulitplies

This once does not actually change anything if dragged around. It's just a multiplier for all camera position and offset values. If set to a low value (like 0.1f) then this makes it easier to precisely position the camera (by dragging the offset values in the inspector).

Camera Roll

The camera roll is an angle that is applie clock-wise to the look direction.

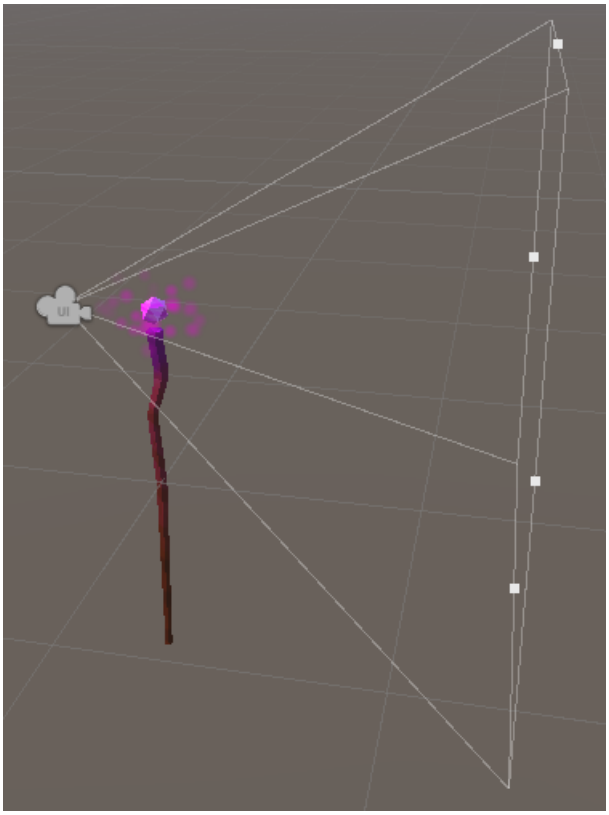
Camera Follow Transform

If enabled then the offsets (CameraLookAtOffset and CameraOffset) are calculated within the local space of the first WorldObject transform. This means the camera will follow the rotation, scale and position of the transform.

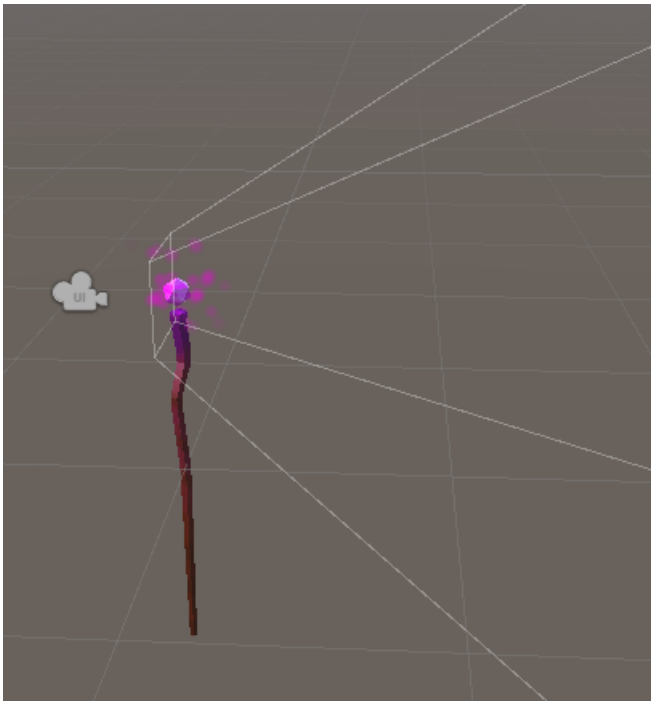
Camera use Bounds To Clip

If enabled then the near an far clipping plane of the rendering camera will be automatically reduced to the size of the 'WorldObjects' combined bounding boxes.

ON:



OFF (near: 0.3 → far: 1000)



Using this option is a nice way to exclude the surroundings of an object without using layers (see „Camera Culling Mask“).

Camera Follow Bounds Center

If enabled then the offsets (CameraLookAtOffset and CameraOffset) are calculated based on the bounding box center.

If disabled then the offsets are based on the position of the very first entry in the 'WorldObjects' list.

HINT: Turn this off if any of your objects are animating or else the camera might be jumpy since the bounds will change with the animation.

Camera Auto Update Bounds

If enabled then the bounds to center on will be updated every frame.

Keep disabled if possible and instead call 'UpdateCameraClippingFromBounds()' manually.

Use Render Texture (experimental)

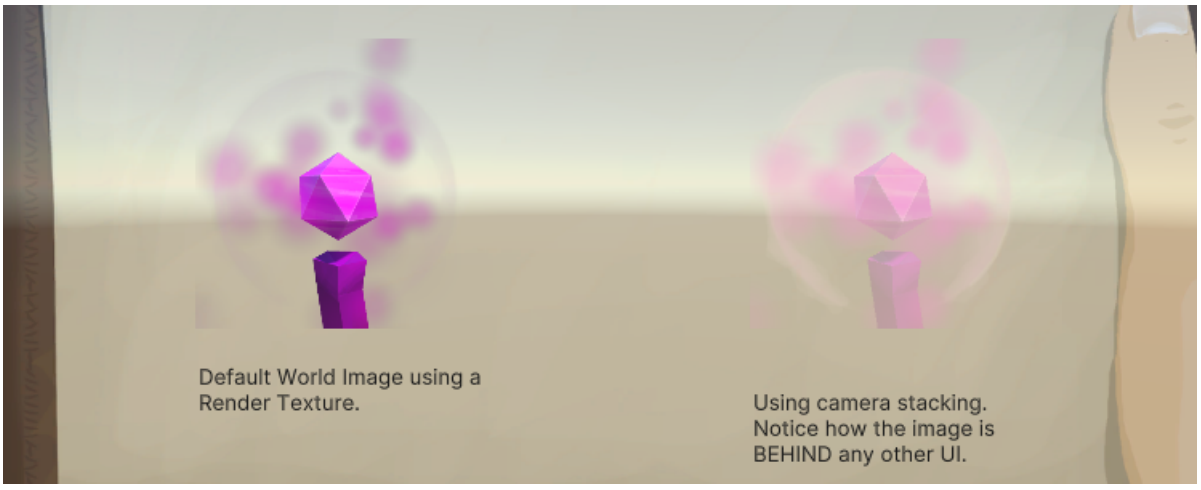
If disabled then the UI image will not render anything and instead camera stacking will be used.

Render Textures usually use pre-multiplied alpha which leads to many problems. The advantage of disabling this is that it gives better results for transparent materials. However, this comes with some major caveats:

* All UI will be IN FRONT of the image if camera stacking is used (more on that below).

* Transparent backgrounds are NOT supported in URP and HDRP if camera stacking is used.

This is showcased in the transparency demo scene (notice the right image is behind the UI):



Camera Near / Far Clip Plane

Sets the near and far clipping plane of the camera.

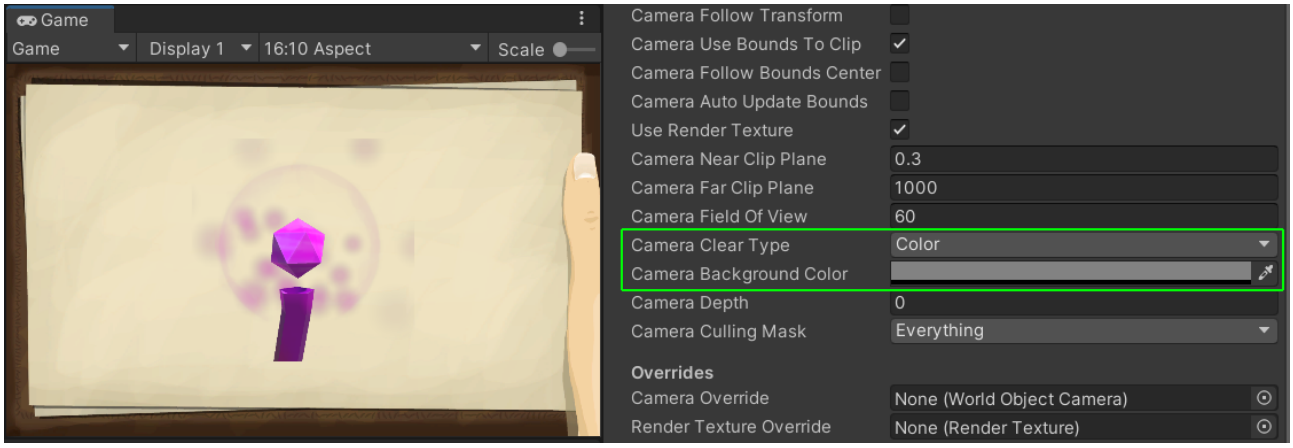
NOTICE: If „Camera Use Bounds To Clip“ is enabled then these are ignored because then the clipping planes are calculated based on the bounding boxes of the world objects.

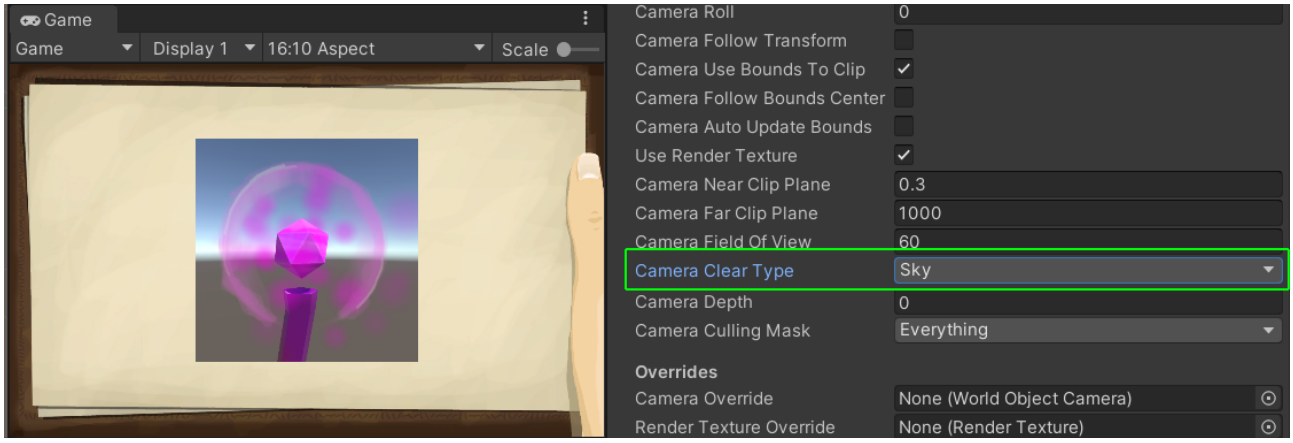
Camera Field of View

Set the field of view of the camera.

Camera Clear Type

This is very similar to the usual camera clear types. You can choose to either use a color (with transparency) or the skybox.





Camera Depth

The camera depths is only used if camera stacking is enabled (i.e. if „UseRenderTexture“ is disabled). It sets the camera depth (called „priority“ in HDRP).

Camera Culling Mask

Defines what layers the object camera will render. You can use this in addition to „CameraUseBoundsToClip“ to fine tune what is rendered.

Camera Override

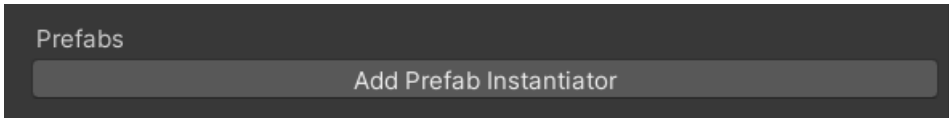
If set then this camera will be used to render into the render texture. Useful for debugging or if you want to use a custom camera.

Render Texture Override

If set then this render texture will be used as the render tearget of the object camera. Useful for debugging or if you want to use a custom render texture.

Prefabs (Prefab Instantiator)

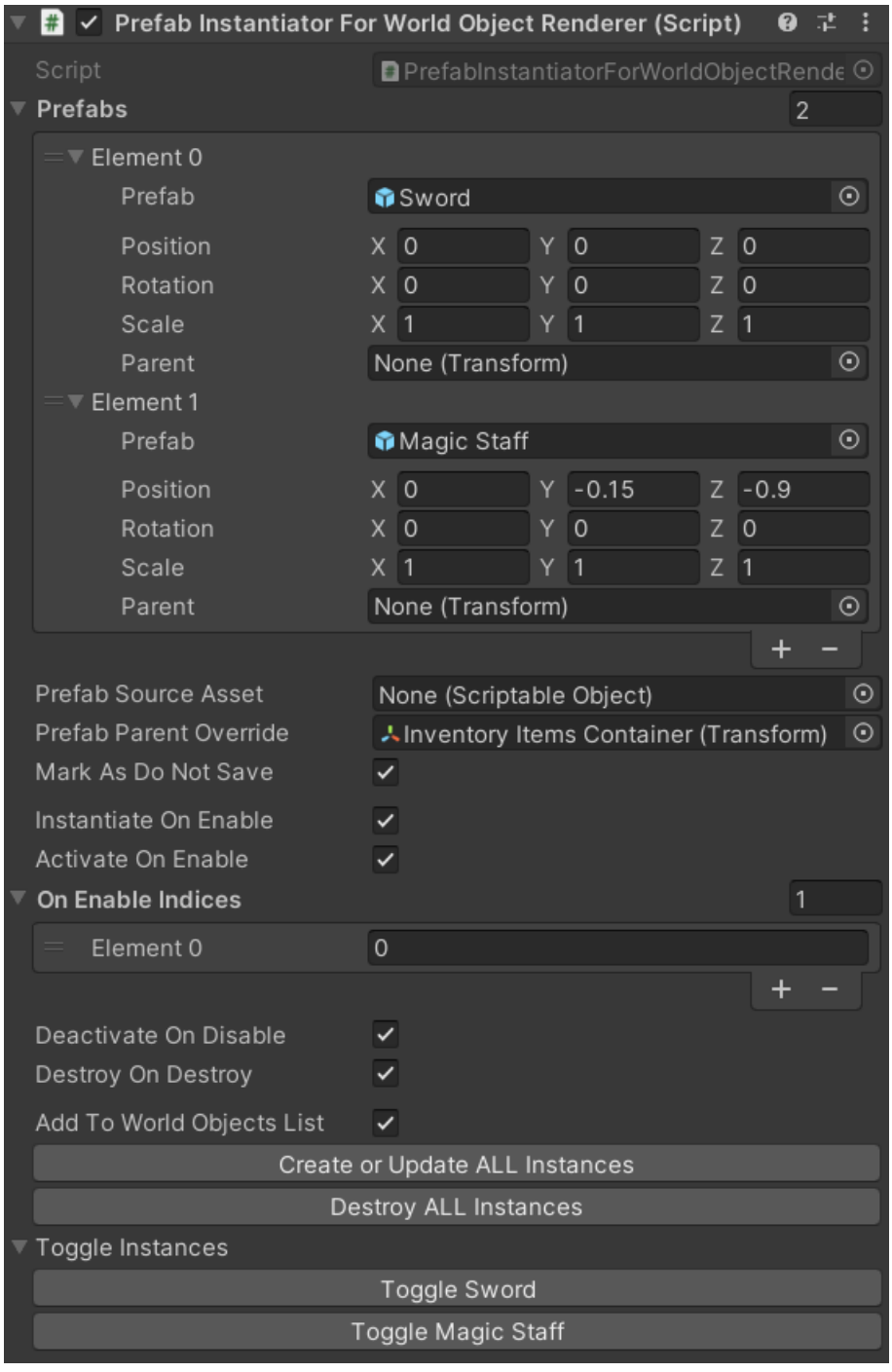
If you scroll down on the World Image you will find this button:



That button will add a „PrefabInstantiator" component to your world image.

HINT: You can find an example of PrefabInstantiator in the „WorldImagePrefabInventoryDemo" scene. The sword and the staff are prefabs that are instantiated on-demand.

The instantiator component contains a list of prefabs. In there you can specify what prefabs to instantiate, when to instantiate and where to instantiate them.



It also has some editor buttons so you can preview how the instances will look.

HINT: Take a look at the „WorldImagePrefabInventoryDemo“ scene. In there the instantiator is used to display the sword and magic staff.

Prefabs > Prefab

This is where you drag in your prefab object from the Assets.

Prefabs > Position / Rotation / Scale

Here you can specify the transform properties of the prefab instance.

Prefabs > Parent

You can specify a parent object for the prefab instance. If you leave it empty then the prefab will be instantiated in the root level of the current active scene.

HINT: Setting the parent for each prefab is tiresome. If you want the same parent for all prefab use the „Prefab Parent Override“ (see below)

Prefab Source Asset

Sometimes you want to configure your prefabs dynamically. In a real game the items shown in the inventory will likely come from the code or another Scriptable Object, not the instantiator list. If you set a source here then this source will take precendence over the normal Prefabs list.

HINT: You can also set an override via code (check out the public methods of the PrefabInstantiator).

Prefab Parent Override

If set then this will be used as the parent for each prefab instance. Very handy for dynamic prefabs sources or long lists of prefabs.

Mark As Do Not Save

By default the HideFlags of the prefab instances are set to:

```
HideFlags.DontSaveInBuild | HideFlags.DontSaveInEditor | HideFlags.NotEditable
```

This is done because the instantiator controls the lifecycle of these instances. It can create, update and destroy them.

If you wish to treat the instances like normal game objects then you can disable this flag.

Instantiate On Enable

Should the prefabs be instantiated in OnEnable? Disable this if your prefab is costly to intantiate and you want to control is manually via CreateOrUpdateInstances().

HINT: You can limit the prefabs that should be instantiated with the „OnEnableIndices" list (see below).

Activate On Enable

If the image is enabled then the prefab instances will be enabled too. This is handy because InstantiateOnEnable will only activate the instances once (on instantiation). Use this if you want to keep your instances around and only enable/disable them.

HINT: You can limit the prefabs that should be activated with the „OnEnableIndices" list (see below).

On Enable Indices

A list of indices of the prefabs that should be instantiated and/or enabled in OnEnable().
NOTICE: this list is used for both „InstantiateOnEnable" and „ActivateOnEnable".

HINT: If you want to control this yourself then please disable the *OnEnable options and use the public methods of the Instantiator to do this manually.

Deactivate On Disable

If enabled then all instances will be deactivated if the WorldImage is disabled.

Destroy On Destroy

If enabled then all instances will be destroyed if the WorldImage is destroyed.

Add To World Object List

If enabled then each instance is added to the WorldImages WorldObjects list. Usually this can be left on. If you disable it then the instances will not contribute to the bounding box calculations of the world image.

Transparency

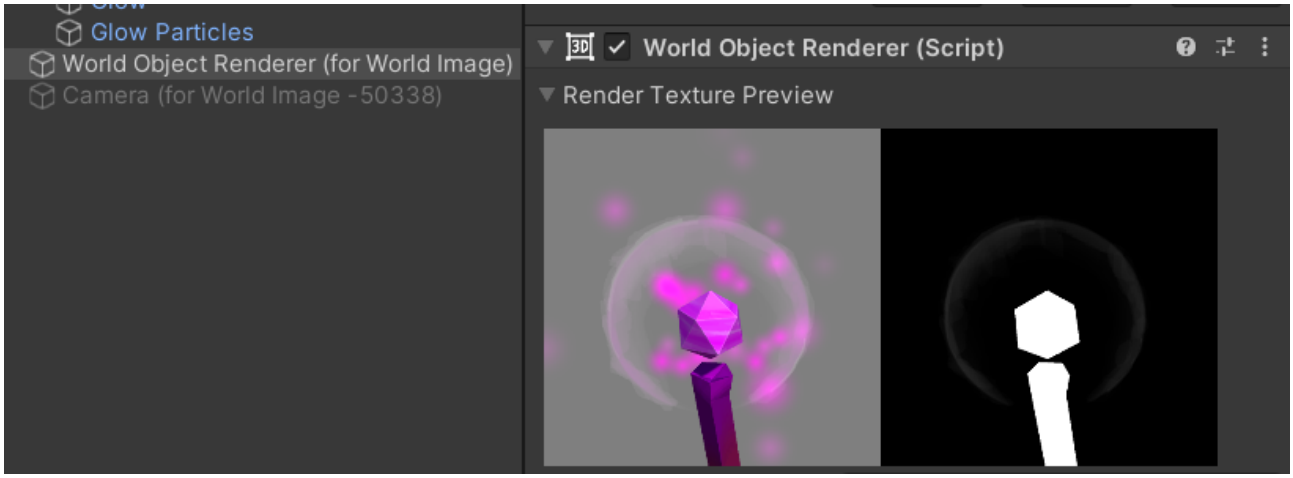
By default this asset uses render rextures to create the image of an object. Sadly there are some problems when it comes to rendering into render textures.

Alpha write (particles are invisible)

The first problem is alpha write. Most of Unity's default shaders do either overwrite (replace) all alpha information (Unlit/Transparent) OR they do not write any alpha information at all (particle shaders).

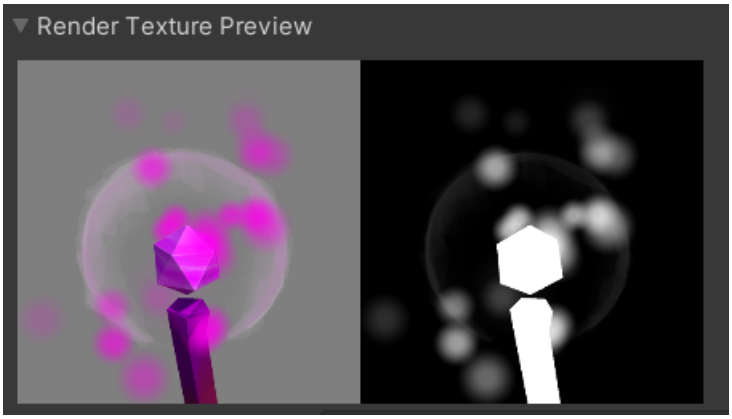
This leads to the effect that transparent objects are either invisible (particles) or rendered as pre-multiplied alpha causing opaque objects behind transparent ones to have an alpha value of „one minus alpha„ if rendered into a render texture.

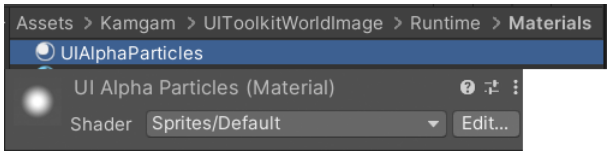
You can check if you material writes into alpha in the Render Texture Preview. If the right area is black then no alpha value was written an those pixels will remain fully transparent.



Notice the missing particle alpha in the right image.

One possible workaround is to use a shader that writes some alpha values (like „Default/Sprite")





However, this does not solve problem number two (premultiplied alpha).

Clear Color + Premultiplied Alpha

The second problem is the fact that in a render texture the background does not come from the frame buffer. That's why you have to define how the initial color is set (clear type + background color).

The rendered color values in render textures are pre-multiplied with alpha by default. This means that the shader displaying them will needs to take this into account.

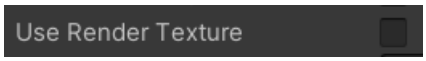
Sadly the default UI Tookit shader does NOT do that and thus pixels without alpha values will be invisible (zero alpha). As of now we can not assign custom materials to UI Toolkit rendered UI and thus it's not possible to fix this even if we had a UI shader that could fix it.

Hopefully Unity will allow us to assign custom materials in the future.

HINT: Once you can use custom shaders you would be ablet to set the background color to all black and then have the shader combine that image properly with the previously rendered pixels.

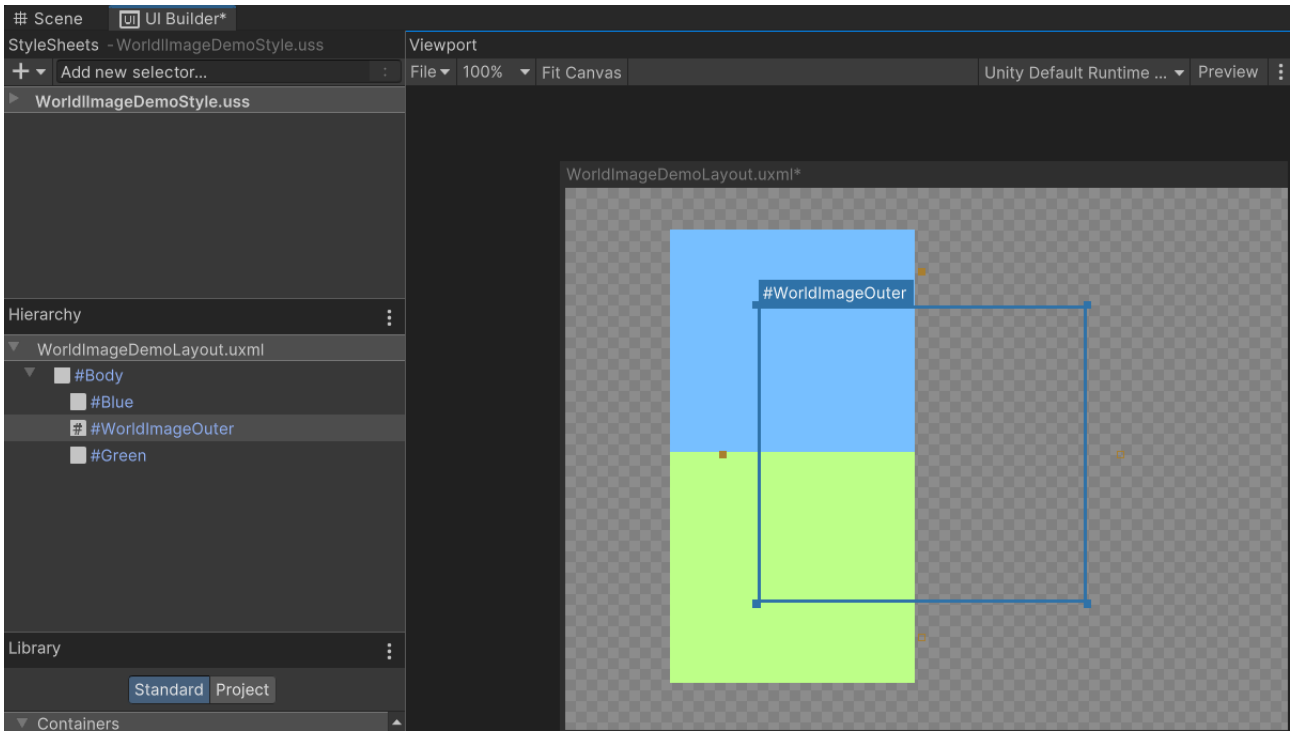
Transparency via camera stacking

If „use render textures“ is disabled then the rendering will use camera stacking. This means instead of rendering in to a texture the object camera will render directly on top of the frame buffer. This allows for better transparency (avoiding the whole pre-multiplied-alpha issues).

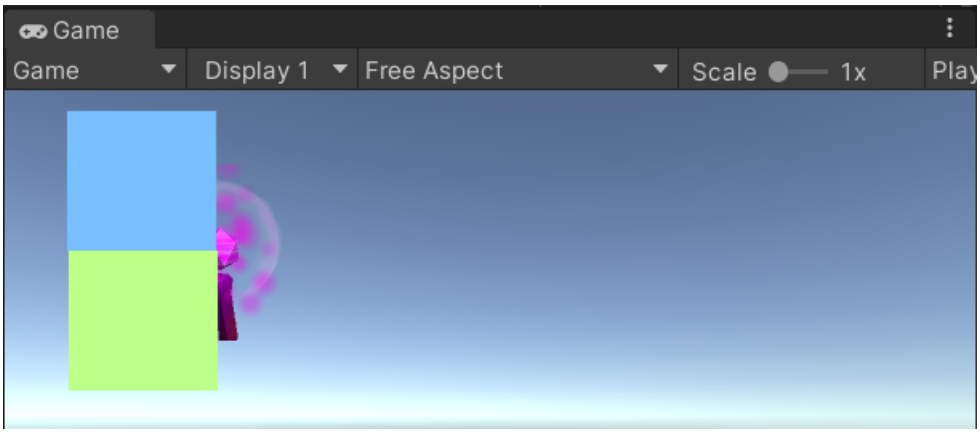


The Built-In render pipeline also supported the clear-type depth (URP and HDRP support varies).

However the image will not be shown in the UI Builder and there is one very big caveat:



The image will always be rendered BEHIND any UI Toolkit UI:



I think it is safe to say that camera stacking is of very limited use with UI Toolkit. It only really works if your UI has elements intersecting with the world image.

Frequently Asked Questions

What about Transparency?

There are some caveats with transparency. Please check the „Transparency“ section above for more details.

Particles are not shown in the Built-In renderer?

Since the default UNLIT particle shaders do not write any alpha values you can try using the LIT shader (Particles/Standard Surface) instead for your particle material. This will make them show up, though the particles will be lit.

Another alternative would be the Sprites/Default shader. That one is unlit and that's the one used in the demo for the particles (see UIAlphaParticlesForBuiltIn material).

Does this support Post-Processing Effects?

As with transparency post-processing is tricky if used on a camera that renders into a render texture (primarily for those effects that do require transparency). Some may work, others might not. However, you can always use the camera stacking workflow to overcome this problem (see „Improved transparency support“ above).

I get the error „Destroy may not be called from edit mode! Use DestroyImmediate instead.“ when entering/exiting play mode.

Sadly this has been a long staning Unity error, see: https://forum.unity.com/threads/case-1426900-error-destroy-may-not-be-called-from-edit-mode-is-shown-when-stopping-play.1279895

Let's hope they fix (and backport) it someday.

All my Object Renderers are greyed out in the scene?

That's probably because either all you WorldImages in the UI Builder are disabled OR you may have loaded the wrong UI layout file (uxml). If the renderer can not find a matching image (by id) then it will disable itself to save resources (no allocate a render texture). Once you have loaded or enabled the matching UI Image it should enable again.