UGUI 3D Object Image - Manual

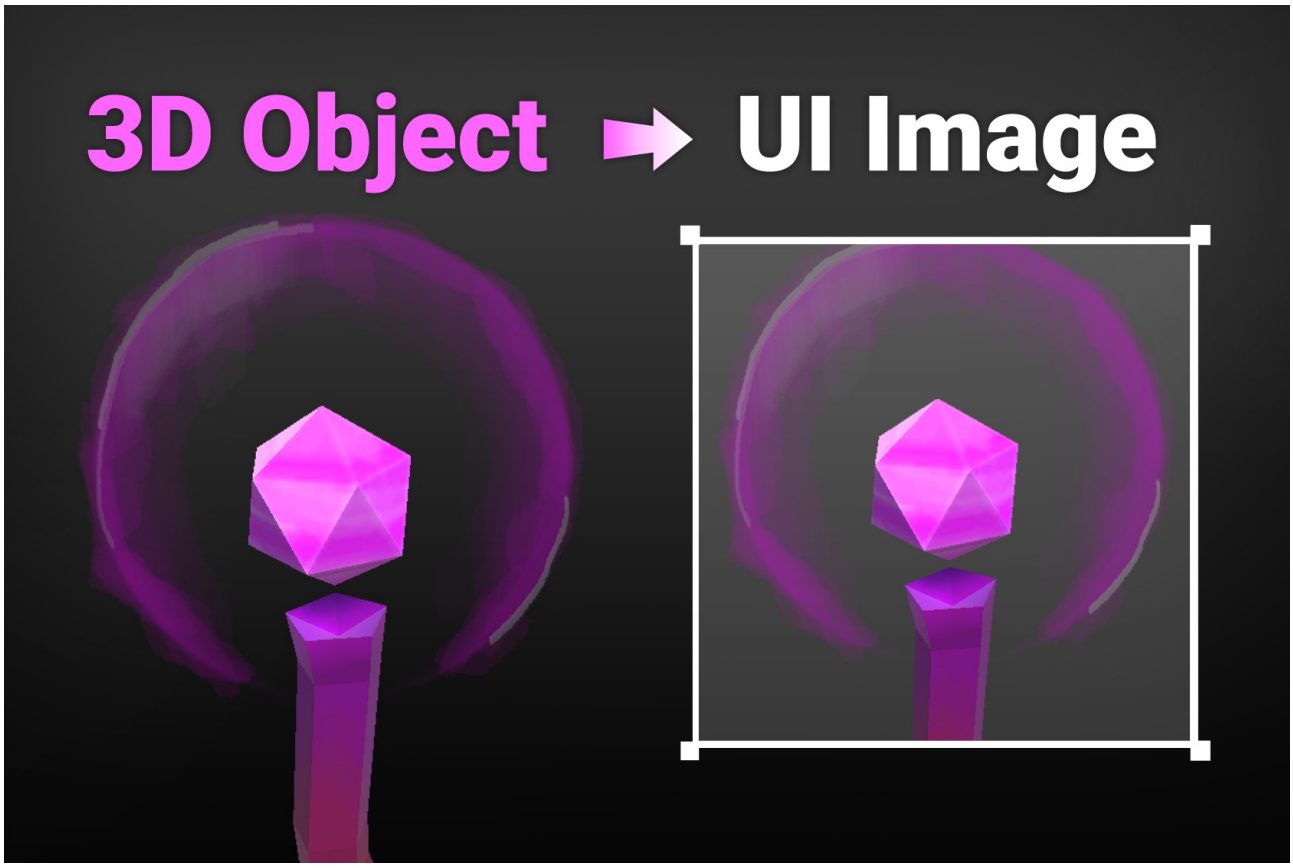


Table of contents

Requirements & Setup **2**
Requirements 2
How to create a World Object Image in the UI **3**
World Image Hierarchy 3
World Image Attributes **5**
Material 6
Color, Raycast Target, Raycast Padding, Maskable, OnCullState 6
World Objects 6
Resolution Width / Height 6
Camera Look At Position / Camera Offset 6
Camera Roll 7
Camera Follow Transform 7
Camera use Bounds To Clip 7
Camera Follow Bounds Center 8
Camera Auto Update Bounds 8
Use Render Texture (experimental) 8
Camera Near / Far Clip Plane 9
Camera Field of View 9
Camera Clear Type 9

Camera Depth....10
Camera Culling Mask....10
Camera Enable Post Processing....10
Camera Override....10
Render Texture Override....11
Prefabs (Prefab Instantiator)....12
Prefabs > Prefab....13
Prefabs > Position / Rotation / Scale....13
Prefabs > Parent....13
Prefab Source Asset....13
Prefab Parent Override....13
Mark As Do Not Save....13
Instantiate On Enable....14
Activate On Enable....14
On Enable Indices....14
Deactivate On Disable....14
Destroy On Destroy....14
Add To World Object List....14
Transparency....15
Alpha write (particles are invisible)....15
Clear Color + Premultiplied Alpha....16
Transparency via SCREEN SPACE CAMERA stacking (Built-In)....18
Transparency via SCREEN SPACE CAMERA stacking (URP / HDRP)....19
Scripting API....21
Adding / Removing Objects....21
Changing the render settings....21
Frequently Asked Questions....22
What about Transparency?....22
Particles are not shown in the Built-In renderer?....22
Does this support Post-Processing Effects?....22

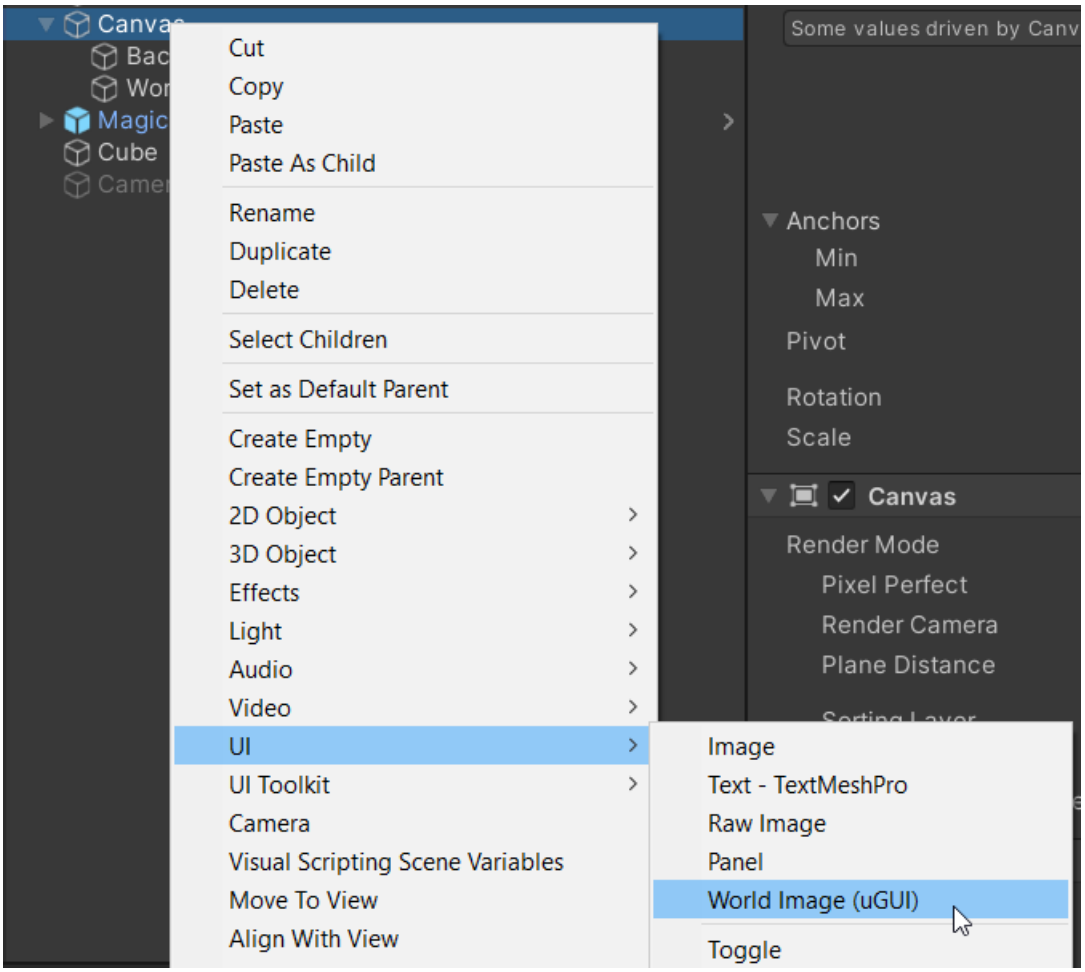
Requirements & Setup

Requirements

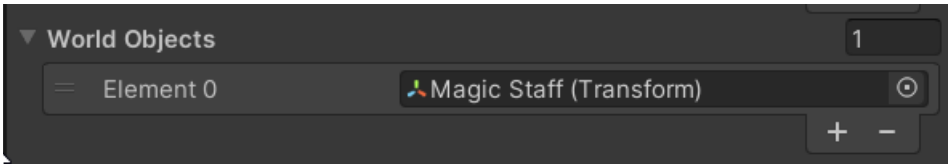
Unity 2021.3 or higher is required since that is the min version Unity allows for new assets in the store. However it may work just fine in older versions of Unity with minor changes.

How to create a World Object Image in the UI

You can add an image via **Right-Click > UI > World Image (uGUI)**:



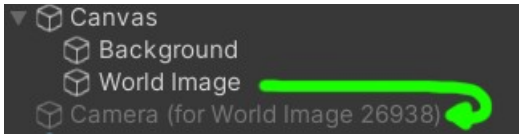
Once you have done that then you can drag in your object(s) into the „WorldObject" list, tweak the options and you are done.

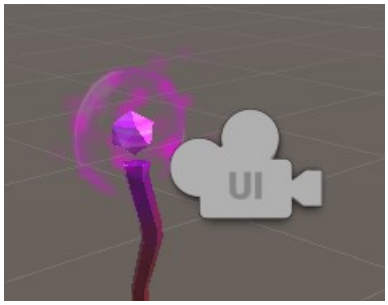


HINT: If you do not add any world object then the image will simply act as a camera into your 3D world. Use „CameraLookAtOffset" to position the camera in the world.

World Image Hierarchy

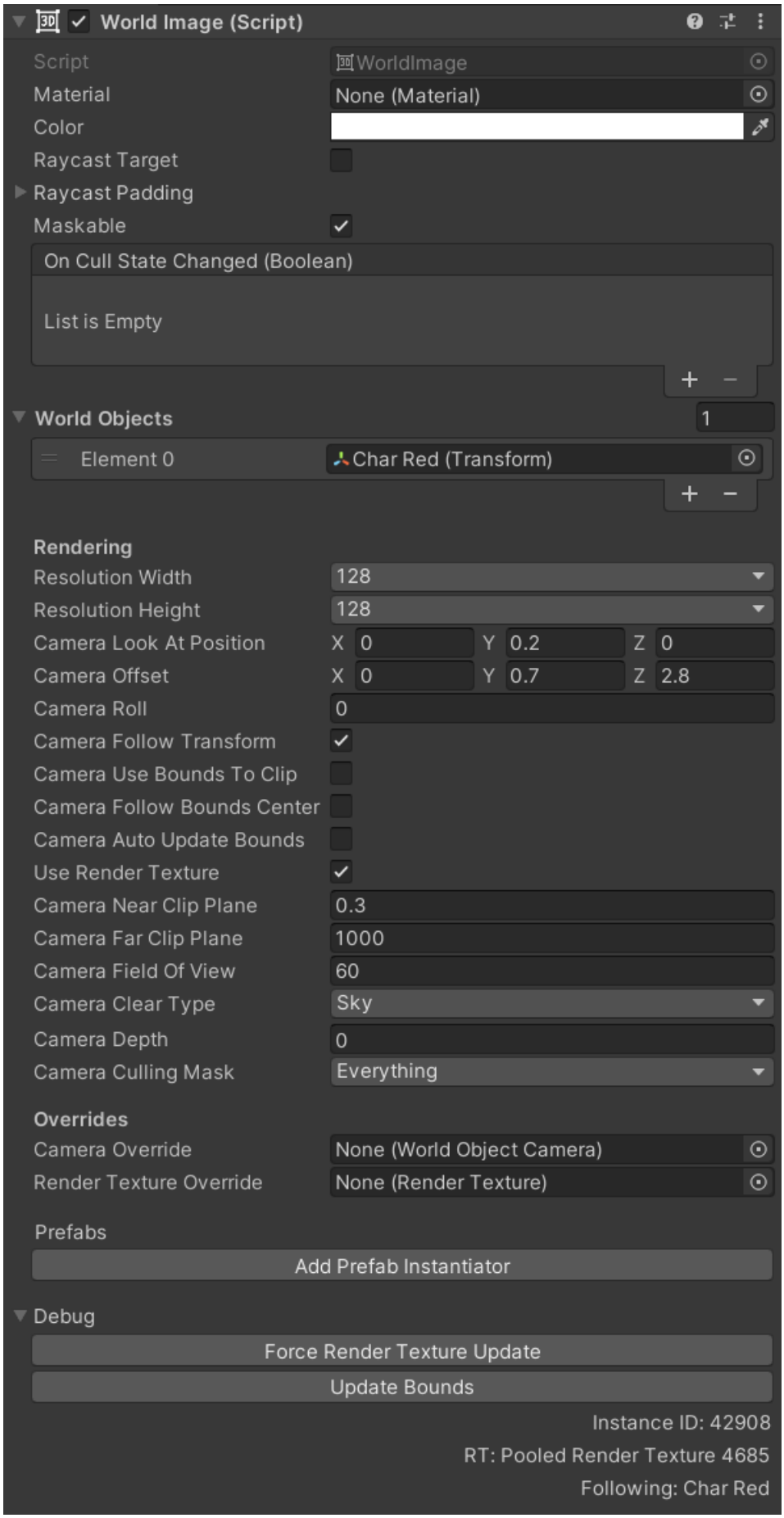
For each image in the canvas there will be a non-persistent temporary camera created in the scene to render your 3D world object.





The camera is used to render in to a render texture which is then used as the source for the UI image.

World Image Attributes



Material

This is the material that is used for the UI image. This works just like in the normal Image component.

NOTICE: Since the image is filled with a texture please make sure the material shader does actually support a texture (most shaders do).

Color, Raycast Target, Raycast Padding, Maskable, OnCullState

These are all inherited from the default „MaskableGraphic" component. They do work just like the default component.

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 Roll

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

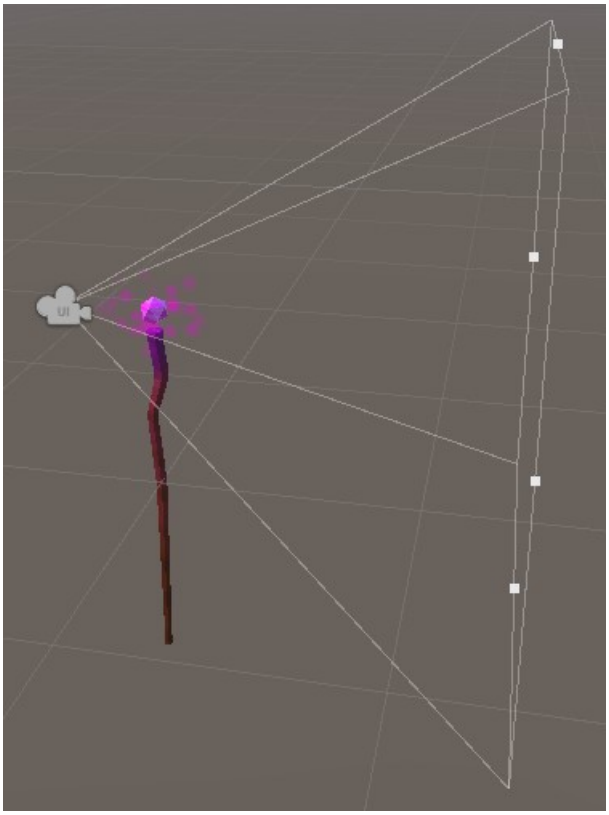
Camera Follow Transform

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.

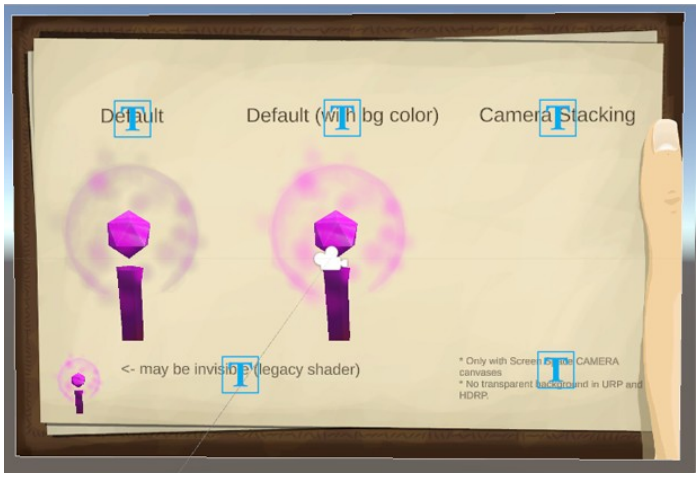
Use Render Texture (experimental)

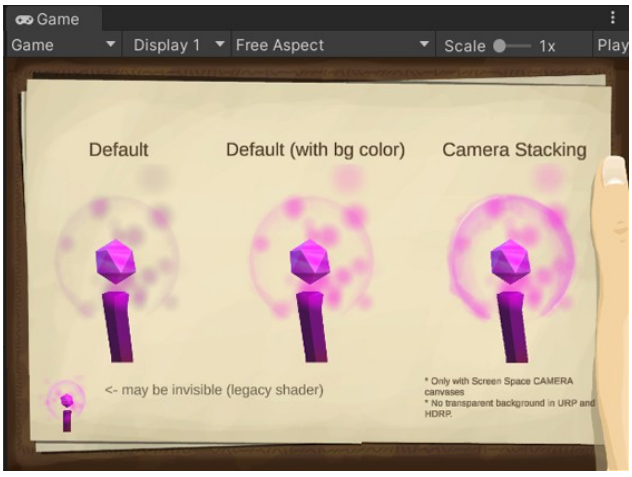
If disabled then the UI image will not render anything and instead camera stacking will be used.

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:

This option only works for SCREEN SPACE CAMERA canvases!

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





Camera Near / Far Clip Plane

Sets the near and far clipping plane of the camera.

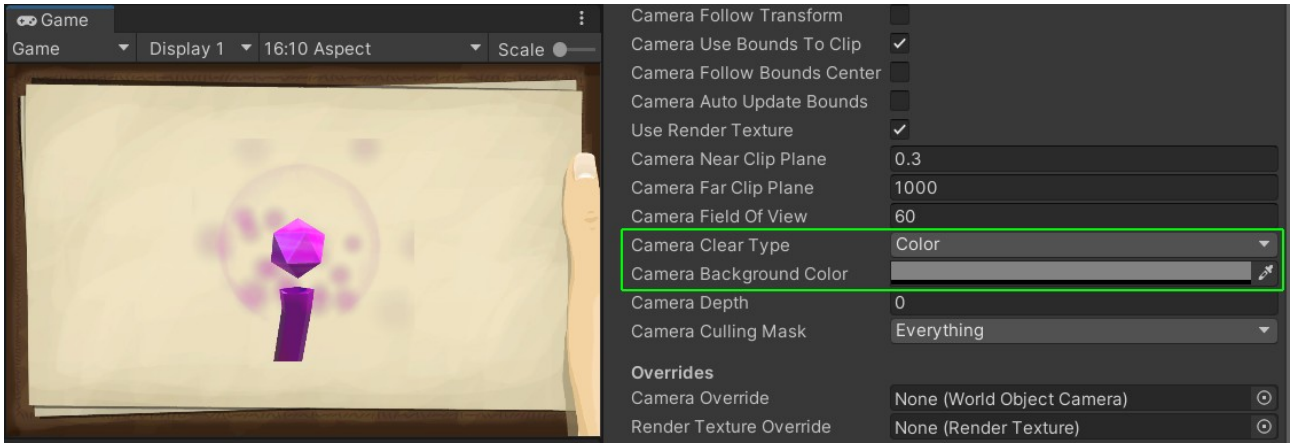
NOTICE: If „Camera Use Bounds To Clip“ is enabled then these are ignored because then the clipping planes are calculated based on the bounding boxes of the world objects.

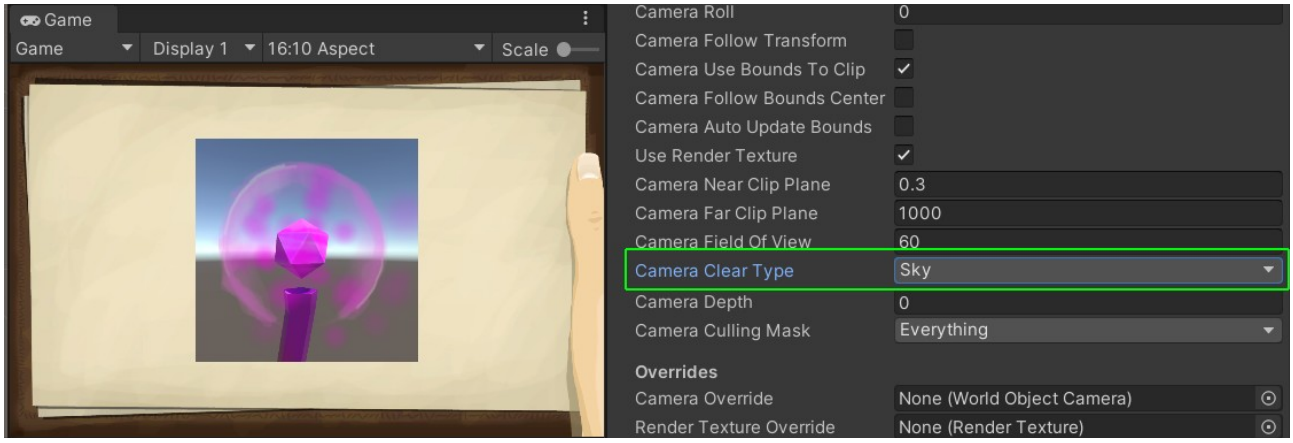
Camera Field of View

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.

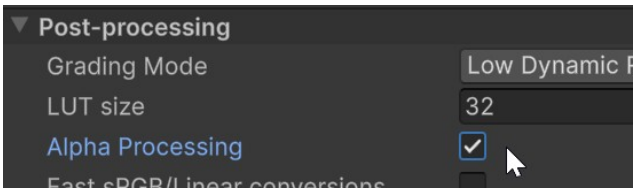
Camera Enable Post Processing

If enabled then the rendering of the object will mirror the post processing of the camera that is marked with the MainCamera tag. If none is found then post processing will be disabled.

NOTICE: If you use local volumes then those will only be applied if the rendered object (and the temp camera) are inside the volume bounds.

CAVEATS:

Transparency may no longer work if postpro is enabled. **In Unity < 6.0 post processing will kill any alpha and transparency (source).** In Unity 6+ you can (and have to) enable alpha support in the renderer (source).



Changing the post processing settings is not propagated to the rendering of the image. **If you change post processing you will have to call ApplyCameraPostProcessing() manually to update the post pro configuration.**

IN HDRP this setting has no effect since post-pro is always enabled by default.

In Unity 2022 URP post processing may applied to all cameras even is disabled (that’s a Unity “bug”). Please upgrade your Unity version if possible.

Camera Override

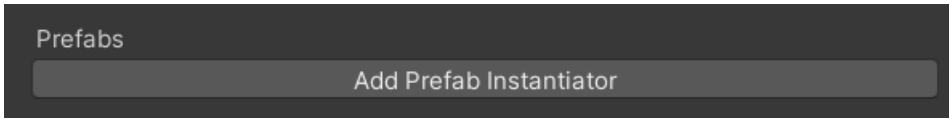
If set then this camera will be used to render into the render texture. Useful for debugging or if you want to use a custom camera.

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)

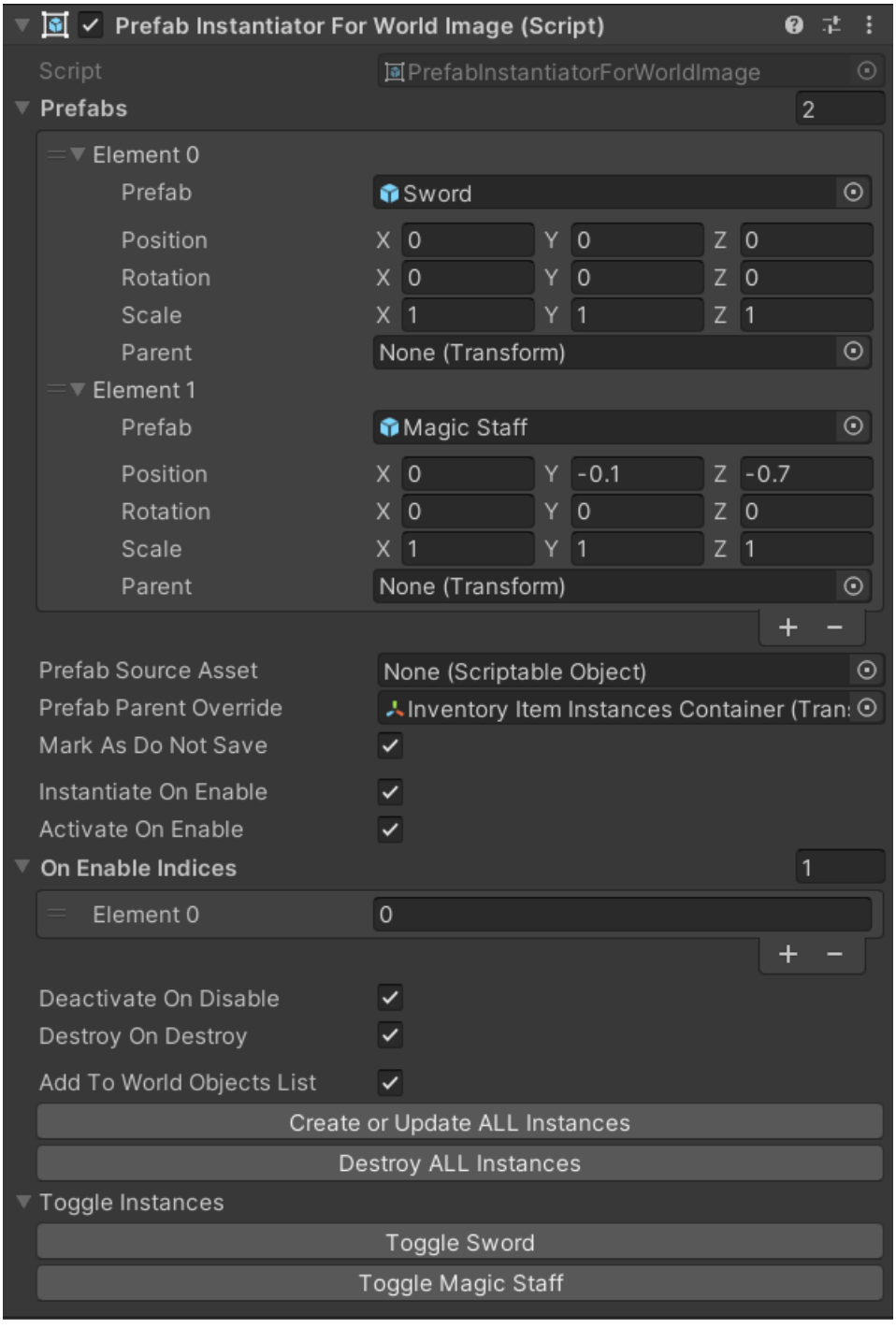
If you scroll down on the World Image you will find this button:



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

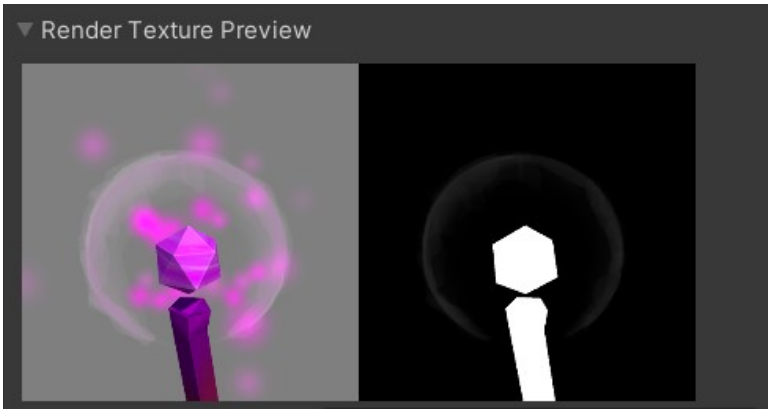
By default this asset uses render rextures to create the image of an object. Sadly there are two problems when it comes to rendering into render textures.

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).

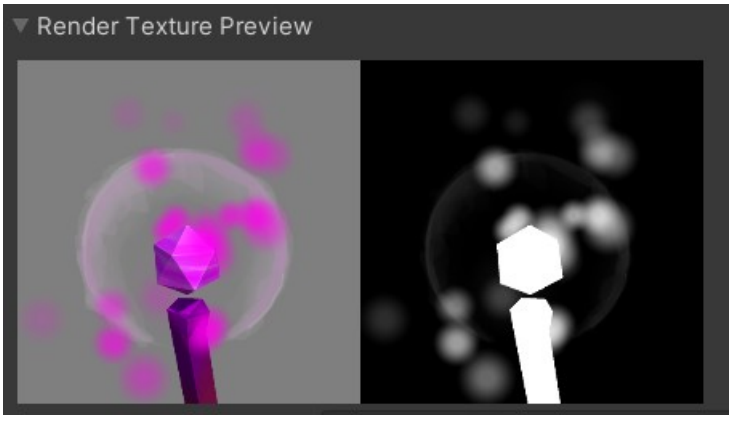
This leads to the effect that transparent objects are either invisible (particles) or rendered as pre-multiplied alpha causing opaque objects behind transparent ones to have an alpha value of „one minus alpha„ if rendered into a render texture.

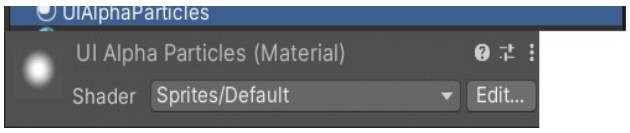
You can check if you material writes into alpha in the Render Texture Preview. If the right area is black then no alpha value was written an those pixels will remain fully transparent.



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")





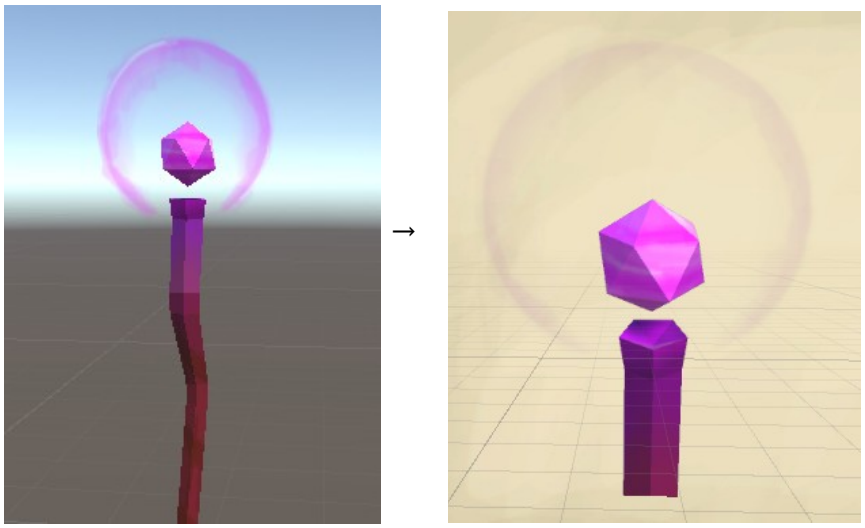
However, this does not solve problem number two (premultiplied alpha).

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.

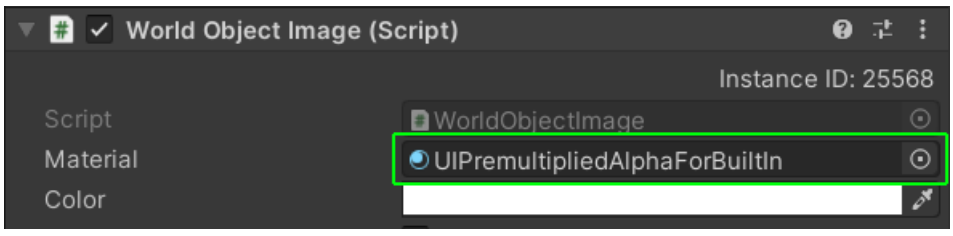
Notice how dull the spherical glow looks in the UI. That's because the transparent pixels are already multiplied with the background color of the render texture. The background color is grey (127,127,127) by default.



Unitys Built-In renderpipeline has such a shader that can be used with an all black background color: „Legacy Shaders/Particles/Alpha Blended". It is not selectable from the default shader dropdown but it is used in the „Default Particle" Material. You can also use the included „UIPremultipliedAlphaForBuiltIn" material I created for easier use.

NOTICE: The „Legacy Shaders/Particles/Alpha Blended" shader may or may not work on your hardware (it's called legacy for a reason). Use at your own risk.

If you want a proper solution then you will have to write your own UI shader that takes pre-multiplied alpha from render textures into account.



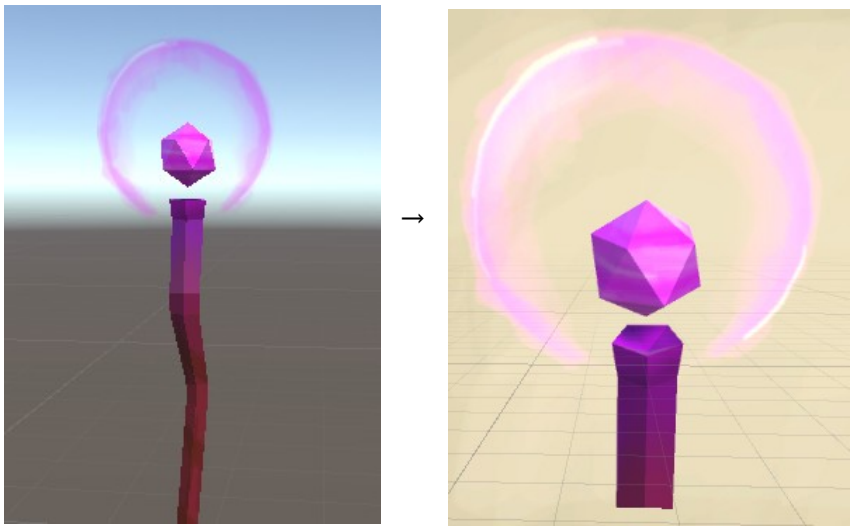
Don't forget to set the background color to black and full transparent if you use this:

While the result is not perfect it is much better than the default blending for some transparent materials. Extra caveats exists though:

a) Sadly the „Legacy Shaders/Particles/Alpha Blended" does not support the _Stencil property and thus **masking will not work if you use this shader**.

b) Since the default UNLIT particle shaders do not write any alpha values you can try using the LIT shader (particles/Standard Surface).

Notice how the transparent halo is much more noticable using this shader. Sadly the overwritten alpha values still interfer but at least this is much better than the default result.



Transparency via SCREEN SPACE CAMERA stacking (Built-In)

If you are not satisfied with how transparency renders and you do not want to make your own shaders to add proper transparency support then you can use a workaround.

How to camera stacking for transparency?

All you have to do it uncheck the „Use Render Texture“ checkbox on the UI image.

NOTICE: While this will give you perfect transparency it also comes with some limitations:

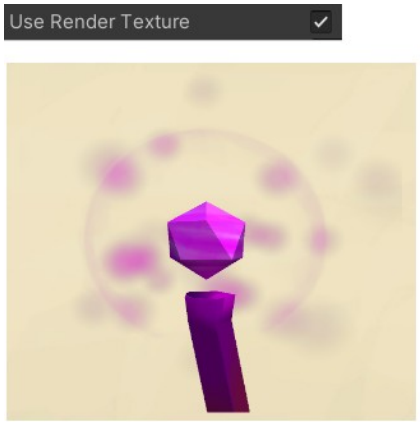
- The biggest one is that **this workaround only works with SCREEN SPACE CAMERA canvases**.
- Another one is that **your image will always be rendered on top of everything else** and you can not rotate nor scale (squeeze or strecht) your image.
    - HINT: You can change the order of images via the „CameraDepth“ option.
- No masking or any other interaction with the UI layout.
- It always renders the object in the current screen resolution.
- This **only works in the Built-In renderer** since URP and HDRP do not support positioning or resizing stacked cameras (source).

The reason for these limits is that this uses camera stacking. By stacking cameras we can use the default render process and do not need to worry about render textures.

Premultiplied alpha is no problem here since we simply render on top of the already existing frame buffer.

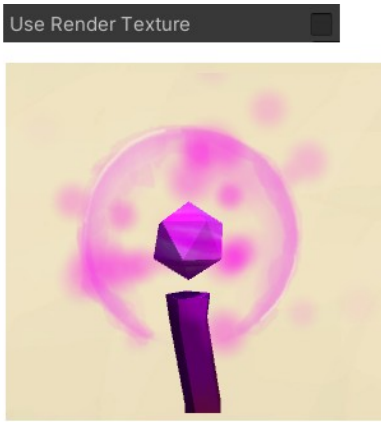
Here is a compariosn of the three supported transparency workflows:

Default transparency



Alpha Blend Material*

Camera stacking (Built-In)



* Alpha Blend material uses a legacy shader and may not work every time.

Transparency via SCREEN SPACE CAMERA stacking (URP / HDRP)

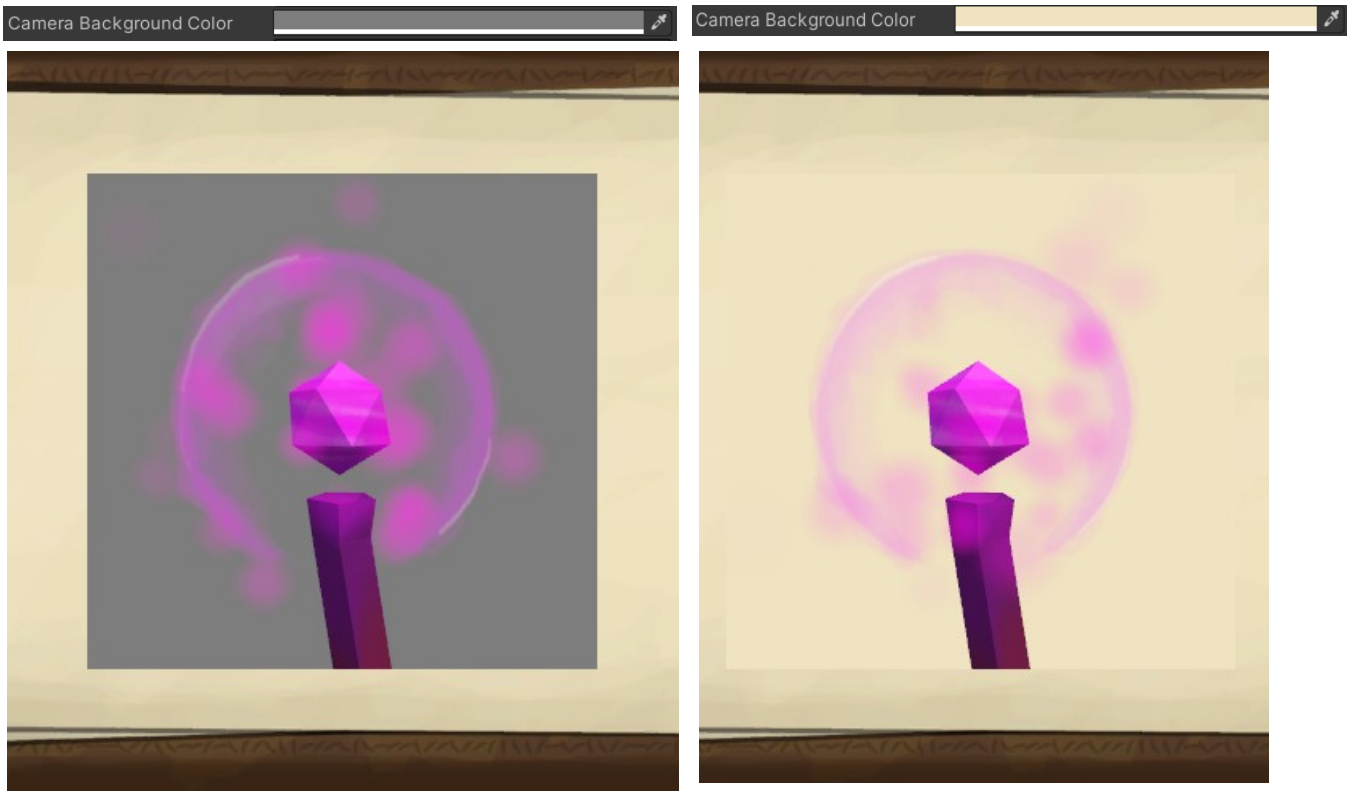
Sadly we can not (yet) position or resize stacked cameras in URP or HDRP (source). Therefore **this workflow is not supported in URP and HDRP**. Once Unity adds this feature it will be available.

Interestingly we can use the old camera stack (as known from the Built-In renderer) in URP and HDRP.

The big downside of this is that in the RPs „`CameraClearFlags.Depth`" is not supported and thus we can not have a transparent background.

However, if your UI has some area with a uniform color then you can use it and simply set the background color according to your UI.

Stacking with the default bg color: → Stacking with a back color that matches the UI:

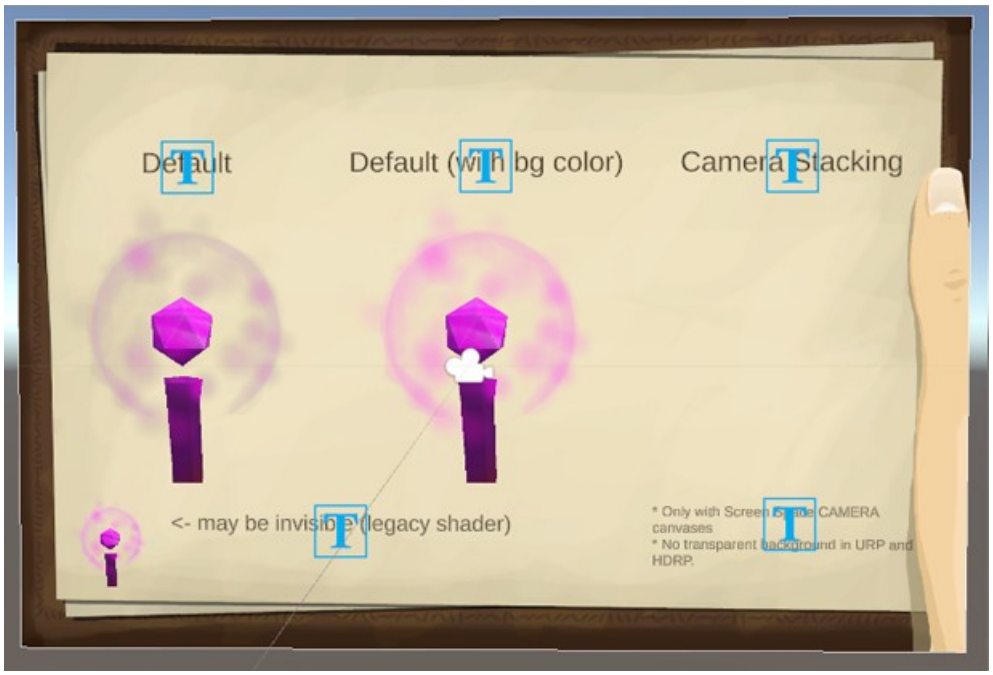


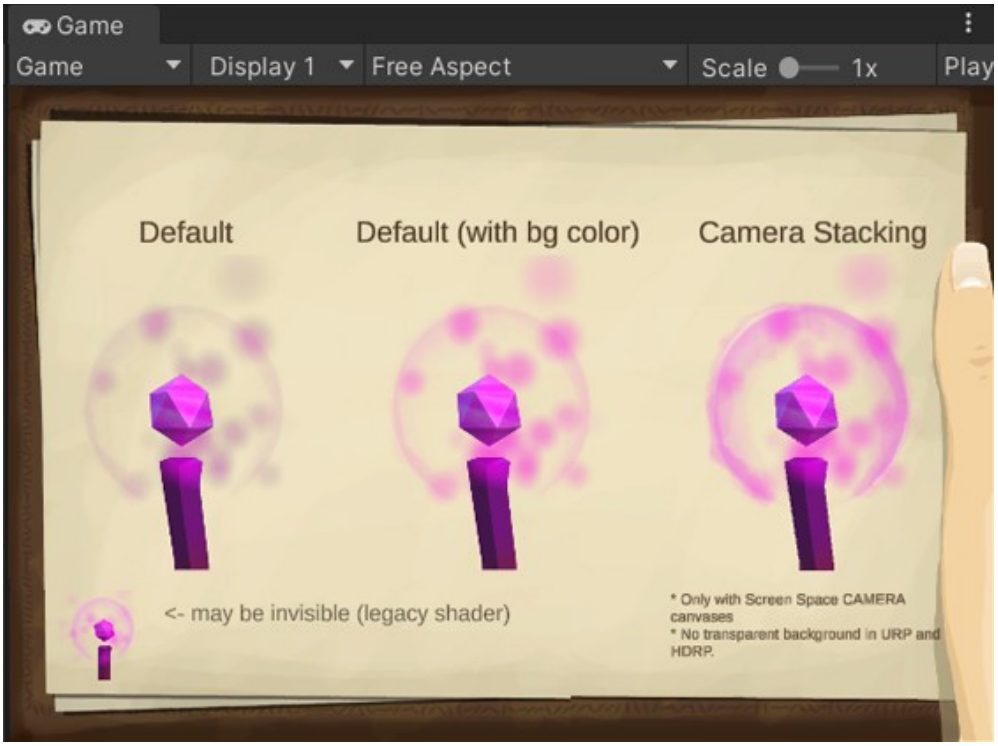
Not a great solution but it's better that nothing.

Another possible workaround is to add the image camera as a overlay camera to your UI camera. That way the image of object will be rendered on top.

However, you will not be able to position or resize it so it will always cover the whole screen. While can mask out parts with a stencil and a custom shader it is to complicated of a workflow and thus it was not added to this asset (more infos on this can be found here).

Please also be aware that if an image uses camera stacking then it will NOT be visible in the UI (scene view) but only in the game view.





Scripting API

Adding / Removing Objects

The image has public „`WorldObjects`“ (Enumerator) and „`WorldObjectsCount`“ properties. The „`WorldObjects`“ property is an enumerator and does not allow changing the internal list. The reason is that the image has to do some update operations if an object is added/removed from the list. If you want to change (add/remove) objects then please use the `AddWorldObject(Transform obj)` or `RemoveWorldObject(Transform obj)` methods.

Changing the render settings

There are „`ResolutionWidth`“ and „`ResolutionHeight`“ properties that you can use to control the rendering resolution. The resolutions are an enum that support 32x32 up to 2048x2048. However, if you really want you can also directly access the „`RenderTexture`“ and set all the properties manually.
NOTICE: Not all render modes use a render texture (i.e.: if UseRenderTexture is set to false camera stacking will be used without a render texture).

There also is a „`ObjectCamera`“ property. This is a wrapper for the actual camera of type „`WorldObjectCamera`“ that is used to render in to the render texture. Here you can set things like ClearType. To directly access the camera use the „`ObjectCamera.Camera`“ property.
NOTICE: Some of the properties of the camera are routinely updated by the image so not all of your changes may persist. If possible use the „Camera…“ properties of the image instead of changing the camera directly.

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).