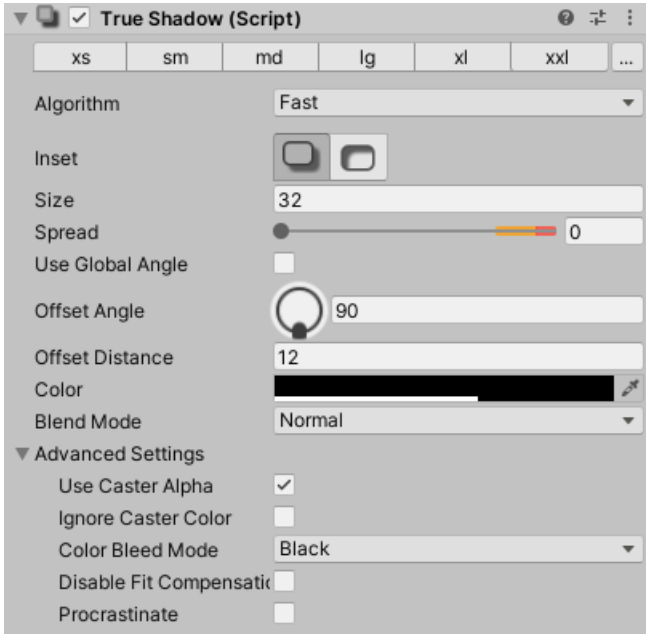
Table of Contents

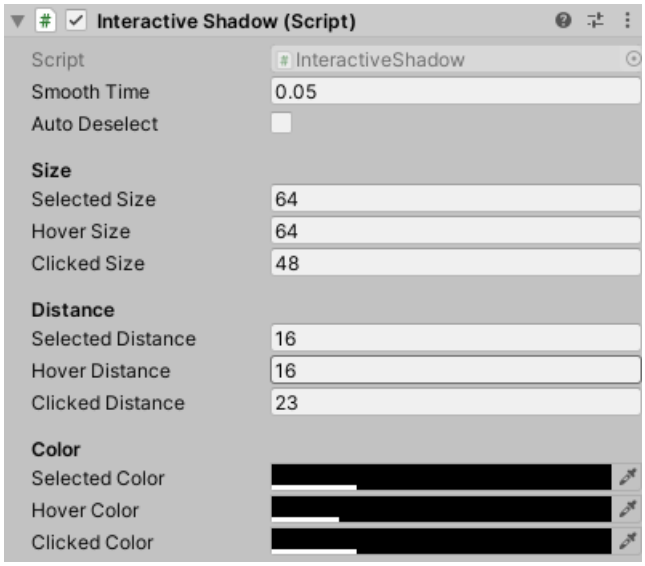
Getting Started

1. Add `True Shadow` to your UI element.

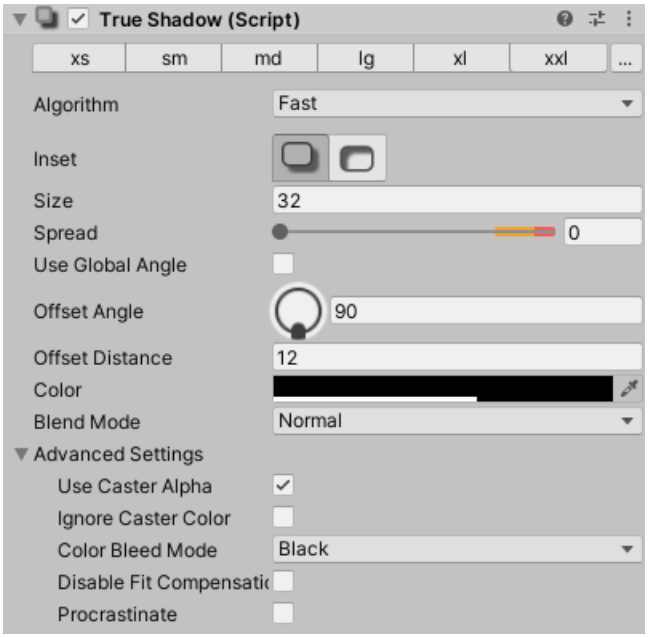


2. Tune it to your liking.

3. Optionally add `Interactive Shadow` to modify shadow properties based on user interaction.



Customize

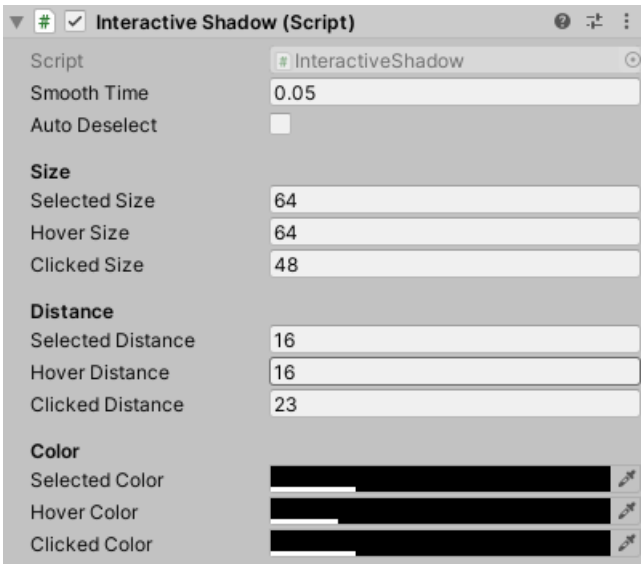


PROPERTY	DESCRIPTION
Quick Presets Bar	Quickly changes shadow settings. Customize the presets with the ... button
Algorithm	Accurate algorithm doesn't miss small features, but can be much slower for large or dynamic shadows. Fast is recommended in most cases
Inset	Choose between inner and outer shadow
Size	Size of the shadow
Spread	Make the shadow thicker
Use Global Angle	Share the same angle across many shadows
Offset Angle	Direction to offset the shadow toward
Offset Distance	How far to offset the shadow
Color	Tint the shadow
BlendMode	Blend mode of the shadow
	- **Normal** : Recommended for colored shadow/glow
	- **Additive** : Recommended for bright glow
	- **Screen** : Recommended for light shadow/glow

PROPERTY	DESCRIPTION
	- **Multiply** : Recommended for dark shadow
Use Caster Alpha	Whether or not the alpha channel of the Graphic affects the shadow
Ignore Caster Color	When on, the color of the shadow will be what is specified in the Color property. When off, the shadow color will be based on the color of the shadow caster Graphic.
Color Bleed Mode	How to obtain the color of the area outside of the source image. Automatically set based on Blend Mode. You should only change this setting if you are using some very custom UI that requires it

Interactive Shadow

The `Interactive Shadow` component allow you to quickly create shadows that can react to user interaction, such as by mouse or game controller.



When created, the component will automatically choose sane defaults based on the settings on your True Shadow component. You can also Reset the component to repopulate its settings based on the current True Shadow settings.

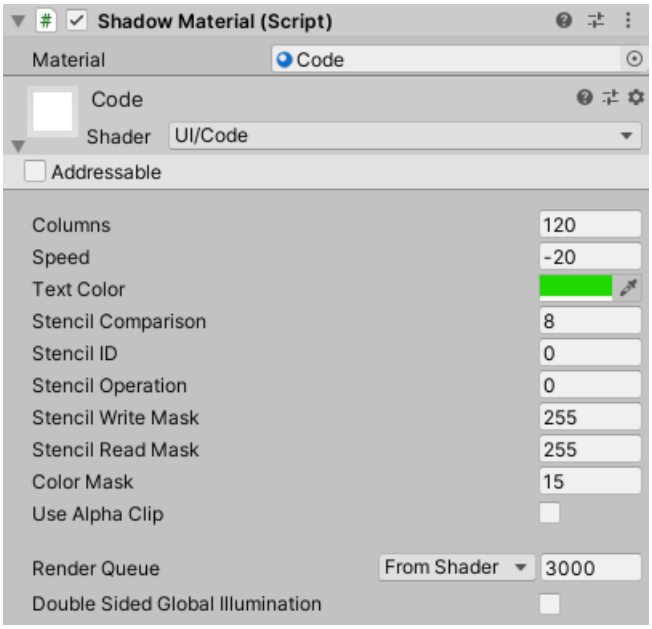
The component supports 3 states: Hovered, Selected and Clicked. These states work the same as the builtin `Selectable`, such as `Button`.

Use Custom Shader on shadows

Some example custom shaders included in the assets:

To use a custom shader on the shadows:

1. Create a Material for the Shader you want to use
2. Add the Shadow Material Component beside the True Shadow component, and assign the created Material



See the `Custom Shadow Material` scene for an example.

Notes on TextMesh Pro

True Shadow supports TMP. However, due to its use of submesh objects, additional code may be necessary.

SubMesh Objects

When using multiple fonts on a TMP, including fallback fonts, glyphs using fonts other than the primary font are rendered with SubMesh objects. Since these are separated UI elements, they need their own True Shadow component.

> **TIP**
>
> You can make these objects visible under `Project Settings > TextMesh Pro > Settings > Hide Sub Text Objects`. If this setting doesn't exist, they should already be visible.

Static text

For texts that don't changes at runtime, you can use the Copy to Sub-Meshes button in the inspector to apply the shadow from the main TMP on any submeshes to preview changes in the inspector. At the start, this is also done automatically.

Dynamic text

If the text are dynamically changed at runtime, some code are neccesarry. These submesh objects don't send the usual dirty signals like other UIs, so True Shadow can't detect when they're changed without expensive polling. When you change these submeshes, you need to call `TrueShadow.CopyToTMPSubMeshes()` to update the shadow on the submeshes.

Since the submeshes are not generated until `Canvas.willRenderCanvases`, you likely have to delay calling this function:

```
using UnityEngine;
using LeTai.TrueShadow;

public class ChangeLocalizedText : MonoBehaviour
{
    private TMPro.TextMeshProUGUI tmp;
    private TrueShadow            trueShadow;

    private void Awake()
    {
        tmp = GetComponent<TMPro.TextMeshProUGUI>();
        trueShadow = GetComponent<TrueShadow>();
    }

    void Start()
    {
        Canvas.willRenderCanvases += OnWillRenderCanvases;
    }

    bool needUpdateTMPSubmeshesShadow = false;

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            tmp.text = Time.frameCount;
            needUpdateTMPSubmeshesShadow = true;
        }
    }

    void OnWillRenderCanvases()
    {
        if (needUpdateTMPSubmeshesShadow)
        {
            needUpdateTMPSubmeshesShadow = false;
            trueShadow.CopyToTMPSubMeshes();
        }
    }
}
```

Still doesn't work?

When no longer in use, the submesh objects are hidden using CanvasRenderer.SetMesh(null). Before Unity 2022.2, there is no way to detect this. Thus, in those versions, you need to destroy the objects to get their shadow to disappear. They will be regenerated by TMP when needed.

```
...
        if (Input.GetMouseButtonDown(0))
        {
            tmp.text = Time.frameCount;

            // Needed before Unity 2022.2
            var submeshes = GetComponentsInChildren<TMPro.TMP_SubMeshUI>();
            for (var i = 0; i < submeshes.Length; i++)
            {
                Destroy(submeshes[i].gameObject);
            }

            needUpdateTMPSubmeshesShadow = true;
        }
...
```

Integration with custom UI types

Make sure shadow update

True Shadow reuses the shadow of identical UI elements. It tells whether 2 UI elements are the same by calculating a hash from their properties.

If some properties of your custom UI type affect the look of the shadow caster, changes to them will not be reflected in the shadow by default.

The fastest way to solve this is to disable the shadow cache for these elements. However, this will be slower and increase graphic memory consumption, potentially by a lot.

To help True Shadow distinguish instances of custom UI type from each other, set the `CustomHash` property. Here is an example for the builtin `Text` type:

```
[ExecuteAlways]
[RequireComponent(typeof(TrueShadow))]
class TextHashProvider : MonoBehaviour, ITrueShadowCustomHashProvider
{
    TrueShadow shadow;
    Text text;

    void OnEnable()
    {
        shadow = GetComponent<TrueShadow>();
        text = GetComponent<Text>();
    }

    void CallThisWhenTextPropertiesAreChanged()
    {
        shadow.CustomHash = HashUtils.CombineHashCodes(
            text.text.GetHashCode(),
            text.font.GetHashCode(),
            (int)text.alignment
            // ...and more as needed
        );
    }

    // Example usage
    void Update(){
        text.text = Time.frameCount;
        CallThisWhenTextPropertiesAreChanged();
    }
}
```

Not all properties of the Text need to be included in the hash. If the properties affect the size of the final UI mesh, True Shadow can detect it automatically.

Use custom vertex data and material properties when rendering shadow

When rendering shadow, True Shadow copy the mesh, vertex data, and material properties from the shadow caster. This will result in the correct shadow in most cases.

In some cases, these data may depend on rendering parameters. For example, you may use the render target size to set a material property. In this case, you must provide True Shadow with the correct property by implementing one of these interfaces on a Component attached to the same GameObject as the True Shadow instance.

- @LeTai.TrueShadow.PluginInterfaces.ITrueShadowCasterMeshModifier

- @LeTai.TrueShadow.PluginInterfaces.ITrueShadowCasterMaterialPropertiesModifier

Support

If you need assistance regarding the asset or have a feature request, feel free to contact me by the form below or at: https://letai.freshdesk.com/support/tickets/new

You will receive an automated confirmation email shortly after submitting a ticket. If not, double check the email address used and try again.

Support request