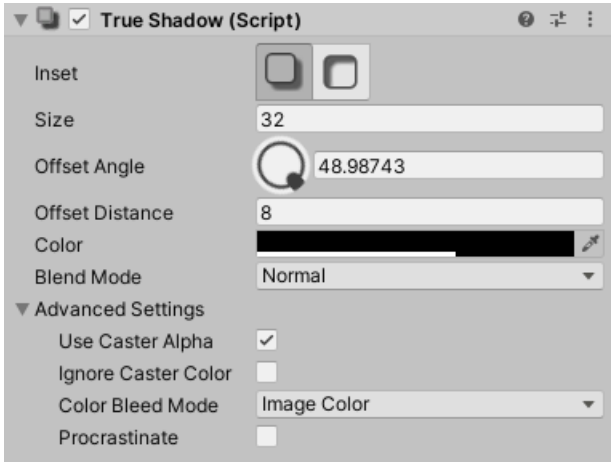
Table of Contents

Getting Started
Customize
Interactive Shadow
Scripting Considerations
Integration with custom UI types

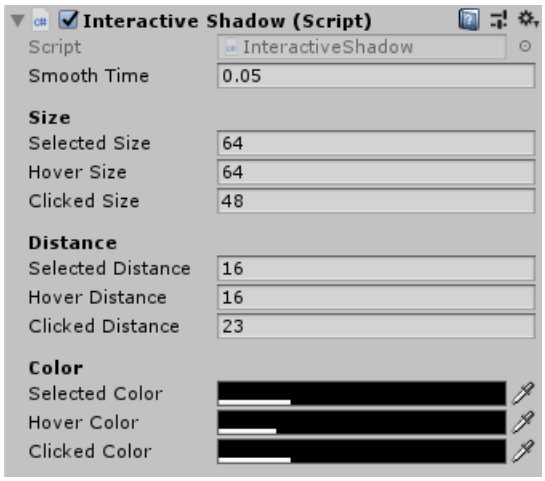
Getting Started

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



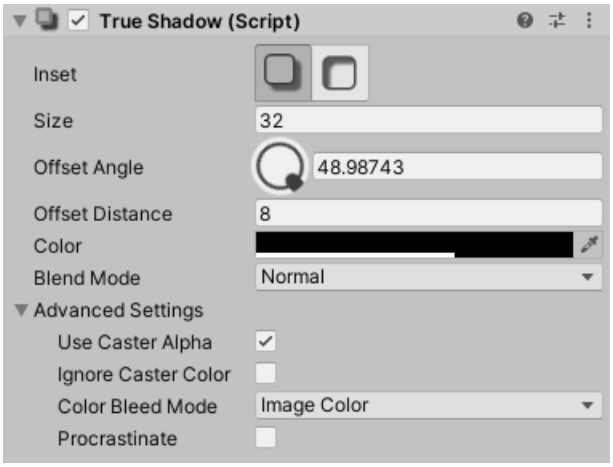
2. Tune it to your liking.

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



Customize

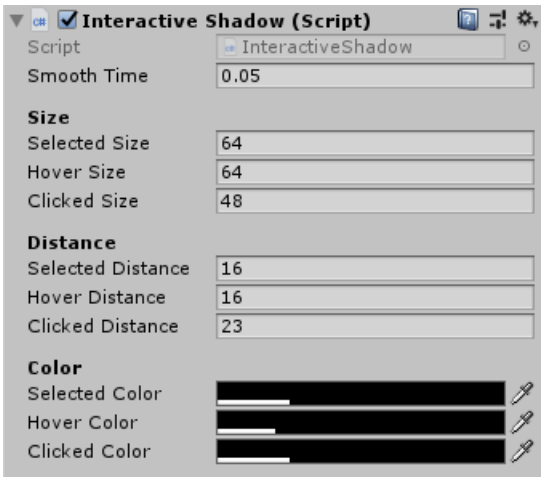
True Shadow



- **Inset**: Choose between inner and outer shadow
- **Size**: Size of the shadow
- **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
- **Use Caster Alpha**: Whether or not the alpha channel of the Graphic affect the shadow
- **Ignore Caster Color**: When on, the color of the shadow will be what 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 require 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`.

Scripting Considerations

True Shadow display shadows by creating hidden GameObject positioned right after the caster UI element in the hierarchy. This is so that the shadow is rendered right above the UI element.

If you are using script that interact with the hierarchy of the shadow casting UIs in some way, you should keep these hidden object in mind. For example, calling `transform.childCount` on the parent will also count the shadow object.

If you spawn shadow casting UIs at runtime, you must not assume the amount of children will be equal to the amount you instantiated. If you instantiate UIs in a loop, you must keep these hidden objects in mind when calculating the instantiated UIs `siblingIndex`.

Generally, it is not a good idea to use `siblingIndex`/`transform.GetChild()` to reference objects. Better alternative are to assign the elements in the inspector, or to use more specific API such as `FindObjectOfType`, `GetComponentInChildren`.

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