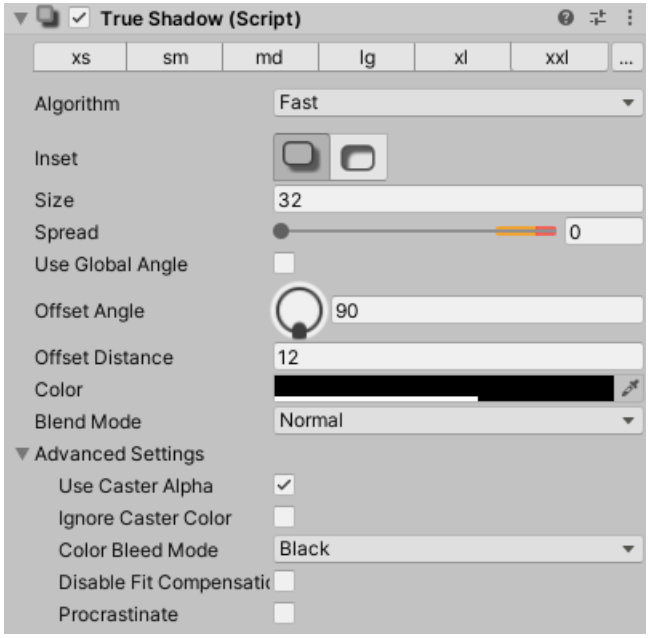
Table of Contents

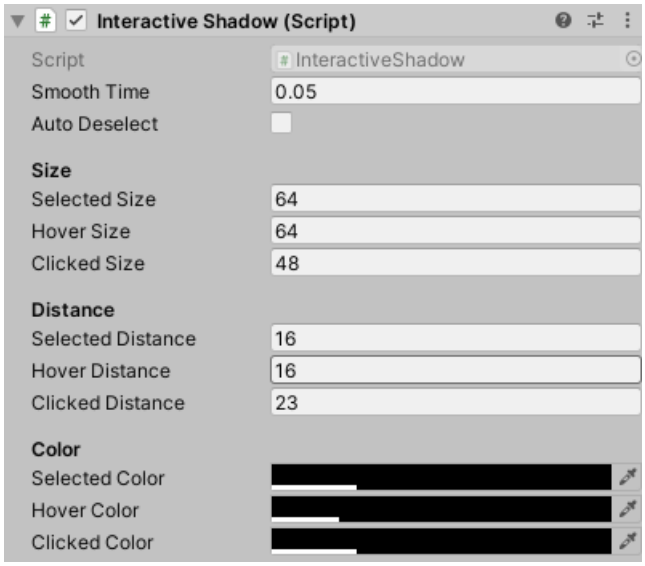
Getting Started

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

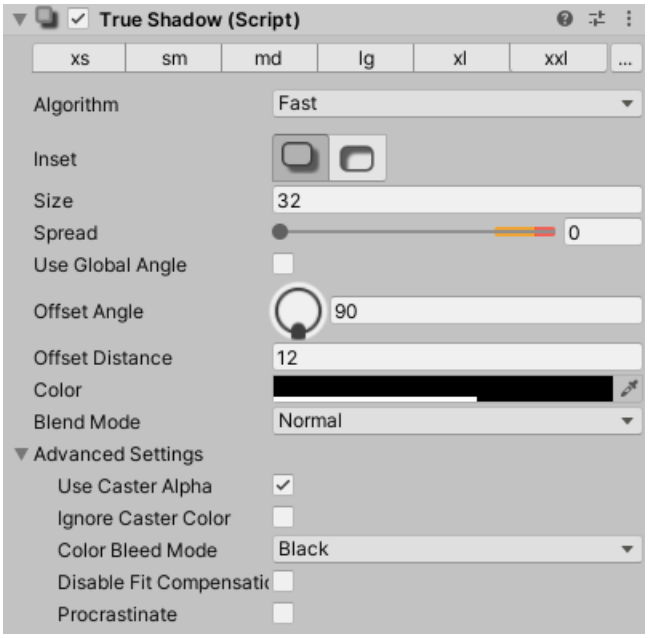


2. Tune it to your liking.

3. Optionally add `True Shadow Interaction Animation` to modify shadow properties based on user interaction.



Customize

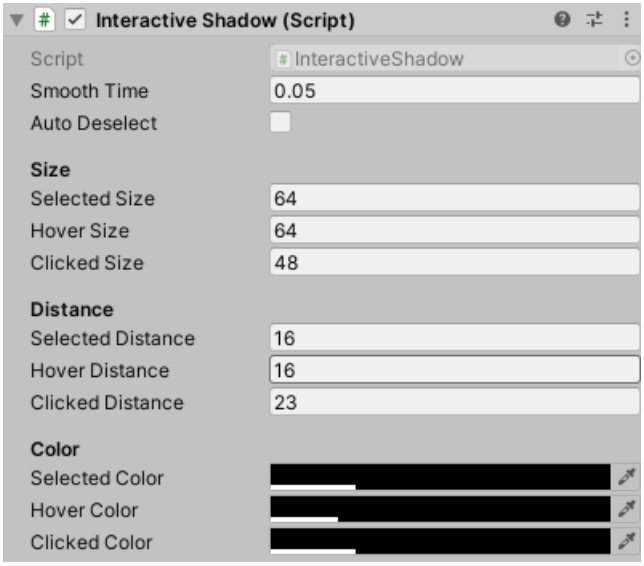


PROPERTY	DESCRIPTION
Quick Presets Bar	Quickly changes shadow settings. Customize the presets with the ... button
Algorithm	Accurate algorithm doesn't miss small features, but can be much slower for large or dynamic shadows. Fast is recommended in most cases
Inset	Choose between inner and outer shadow
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
	- **Multiply** : Recommended for dark shadow

PROPERTY	DESCRIPTION
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.
Skip Text Hasing	Skip expensive hashing of long text. If the text may change without changing length, you must set CustomHash for the change to be detected. For example, if the text comes from a document, the name or id of the document can be used.
Deallocate On Disable	By default, shadow kept the last generated shadow texture to avoid having to regenerate them on enable. Turn this on deallocate that texture to reduce memory usage
Disable Fit Compensation	May improve shadow fit on some casters with thin features

Interaction Animation

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

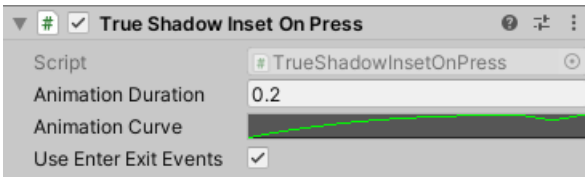


When created, the component will automatically choose sane defaults based on the settings on your True Shadow component. You can also Reset the component to repopulate its settings based on the current True Shadow settings.

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

Inset On Press

The `True Shadow Inset On Press` component smoothly animate between normal and inset shadow for that extra oomph

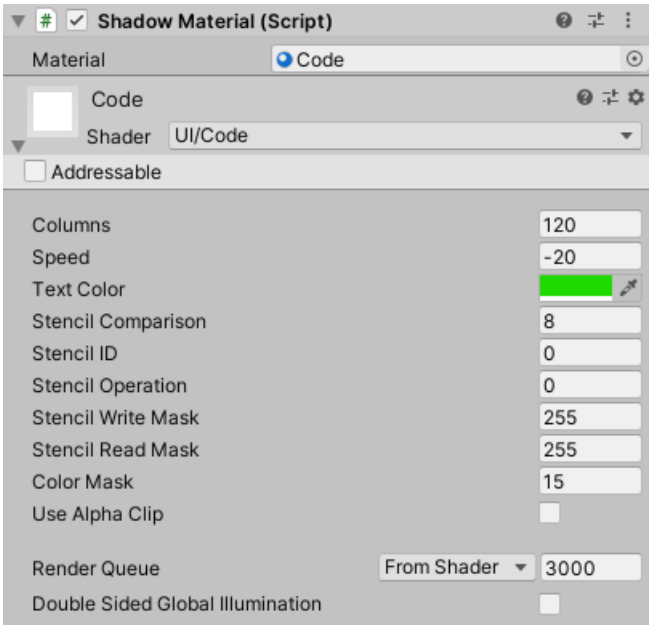


Use Custom Shader on shadows

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 `True Shadow Custom 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

SubMesh Objects

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

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

Making sure shadows fit and update correctly

True Shadow supports most custom UI components. However, it may need some help to achieve the best performance.

True Shadow caches shadow textures, and can even share them between identical UI elements. For custom UI with procedural shape, you need to provide True Shadow with a hash of your shape to let it know when the shape has changed. You can also disable the cache.

Problem: True Shadow doesn't fit the shadow caster correctly, and doesn't update when the caster changes.

Solution:

- Add the `True Shadow Disable Cache` component
  - *Advantage*: Quick and easy
  - *Disadvantage*: Copies of the same shadow caster will each generate their own shadow texture
  - *When to use*: Unique elements like paragraphs of text, or animated elements that change every frame
- Implement `ITrueShadowCustomHashProviderV2`
  - *Advantage*: Copies of the same shadow caster will share the same shadow texture. Saving both compute and memory.
  - *Disadvantage*: Some code is needed. Add overhead when shadows are unique or need to be regenerated every frame anyway.

True Shadow Disable Cache

By default, the component updates the UI only once in OnEnable. If the shadow caster is animated, turn on Every Frame. If it is only animated some of the time, make sure to turn this off when the caster is static to avoid unnecessarily generating new shadows.

Some 3rd-party UIs don't correctly mark themselves as dirty when animated. Turn on Mark Graphic Dirty if the shadow still doesn't update.

Custom Hash Provider

Below is an example implementation of the `ITrueShadowCustomHashProviderV2` interface:

```
using UnityEngine;
using UnityEngine.UI;

// Only necessary if you're publishing code for people who may not have True Shadow
#if LETAI_TRUESHADOW

partial class MyDissolveEffect : Graphic, ITrueShadowCustomHashProviderV2
{
    public event Action<int> trueShadowCustomHashChanged;

    void Update()
    {
        // Only invoke the event when the UI's shape has changed.
        if(material.GetFloat("_DissolveScrollSpeed") != 0)
        {
            // Only properties that can affect the shadow need to be included in the hash
            var myShapeHash = HashUtils.CombineHashCodes(
                material.GetFloat("_DissolveThreshold").GetHashCode(),
                material.GetFloat("_DissolveSoftness").GetHashCode()
            );

            // Or use a random number, if there are too many properties
            // This disables texture sharing, but still regenerates shadow only when needed.
            // var myShapeHash = Random.Range(int.MinValue, int.MaxValue);

            trueShadowCustomHashChanged?.Invoke(myShapeHash);
        }
    }
}
#endif
```

Use custom vertex data and material properties when rendering shadow

When rendering shadow, True Shadow copies 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.

- @LeTai.TrueShadow.PluginInterfaces.ITrueShadowCasterMeshModifier
- @LeTai.TrueShadow.PluginInterfaces.ITrueShadowCasterMaterialPropertiesModifier

Note for Asset Store Publishers

If you have an asset that is a custom shader on top of any subclass of UI.Graphic, including builtin UI components like Image, it most likely will work with True Shadow. I'd love to hear about your asset. Contact me at contact at leloctai dot com for any business related queries. I'm happy to provide True Shadow for free to test this out, and potentially showcase your asset on True Shadow's store page and website.

True Shadow will define the symbol `LETAI_TRUESHADOW` in projects with the asset. You can use this to compile out any code related to True Shadow when your asset is used in a project without it.

If your asset is a shader without any custom components, users can still benefit from a static method that lets them manually update True Shadow when they want to modify the shader properties at runtime:

```
#if LETAI_TRUESHADOW
    public static void UpdateTrueShadowHash(TrueShadow shadow)
    {
        shadow.CustomHash = ...
    }
#endif
```

Gradual integration

If you have a lot of custom properties, you may think it is too time consuming to integrate with True Shadow. However, you can improve user experience a lot with just 3 extra lines of code:

```
class YourClass : .., ITrueShadowCustomHashProviderV2  // Hide warning
{
    public event Action<int> trueShadowCustomHashChanged;

    void OnValidate(){ // Or OnGUI for custom inspector
        // Set a random hash
        trueShadowCustomHashChanged?.Invoke(Random.Range(int.MinValue, int.MaxValue));
    }
}
```

This help True Shadow update as users configure your asset in the inspector. Accurate hash for more efficient caching can be added gradually later.

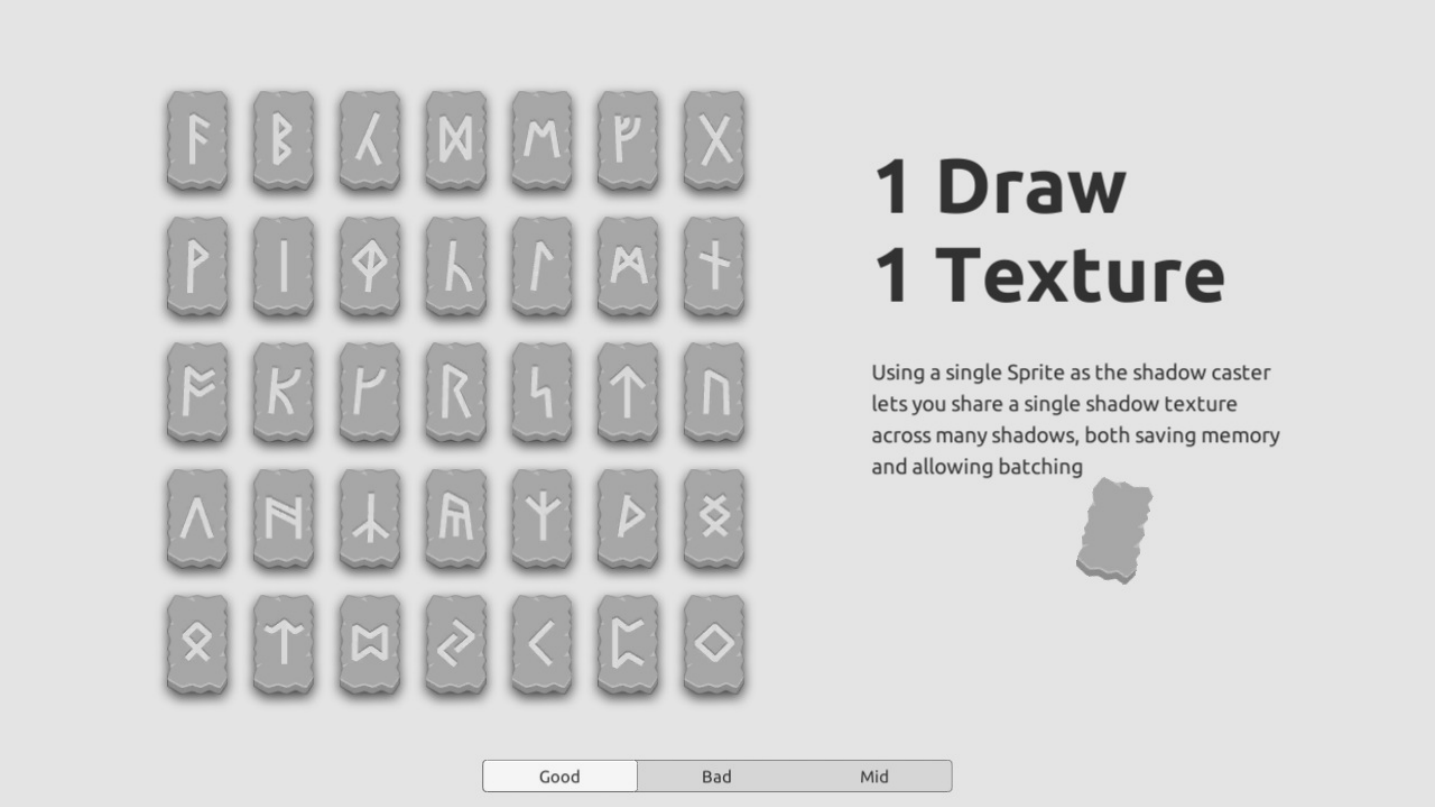
Batching

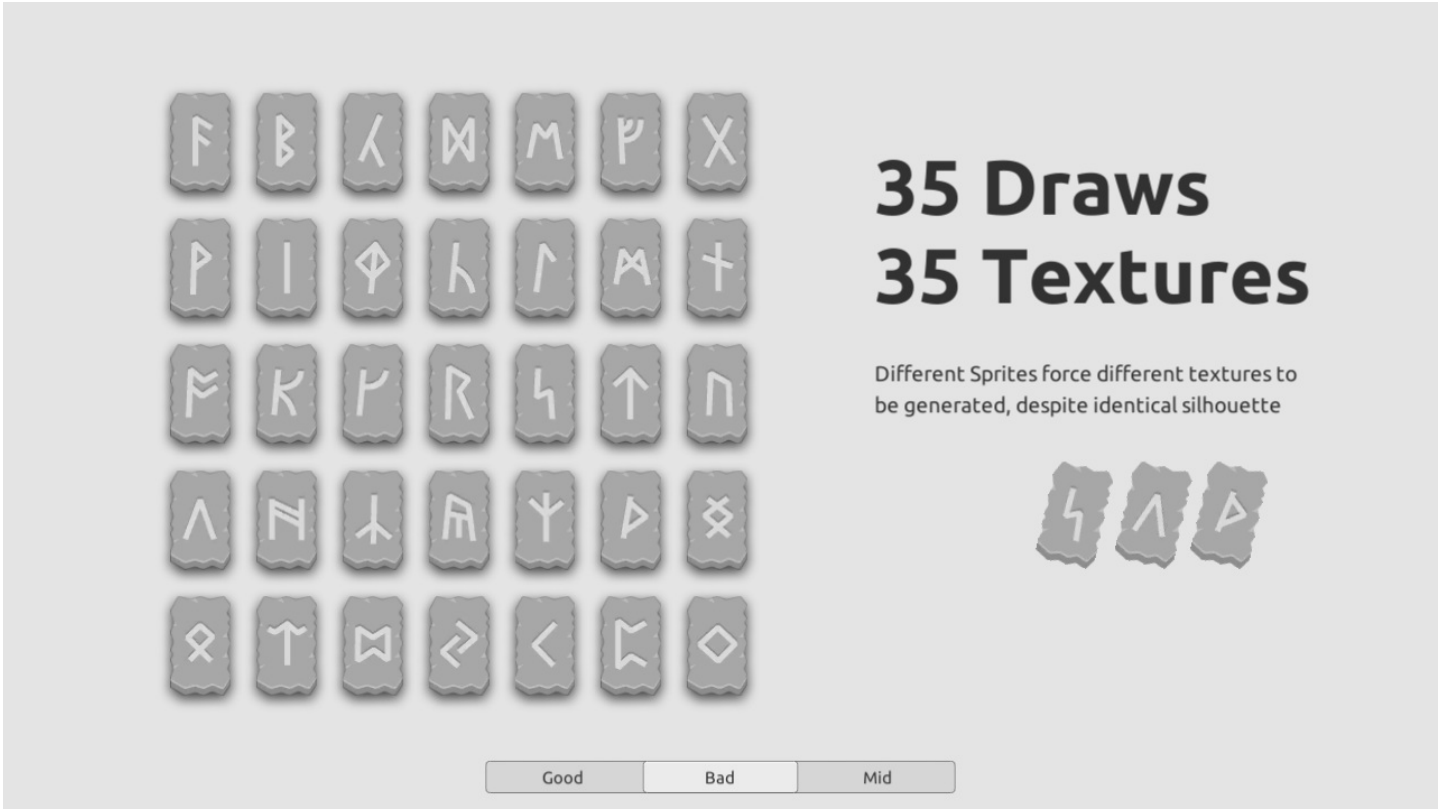
TIP

Check the `Batching` Demo scene to see how the below examples are implemented

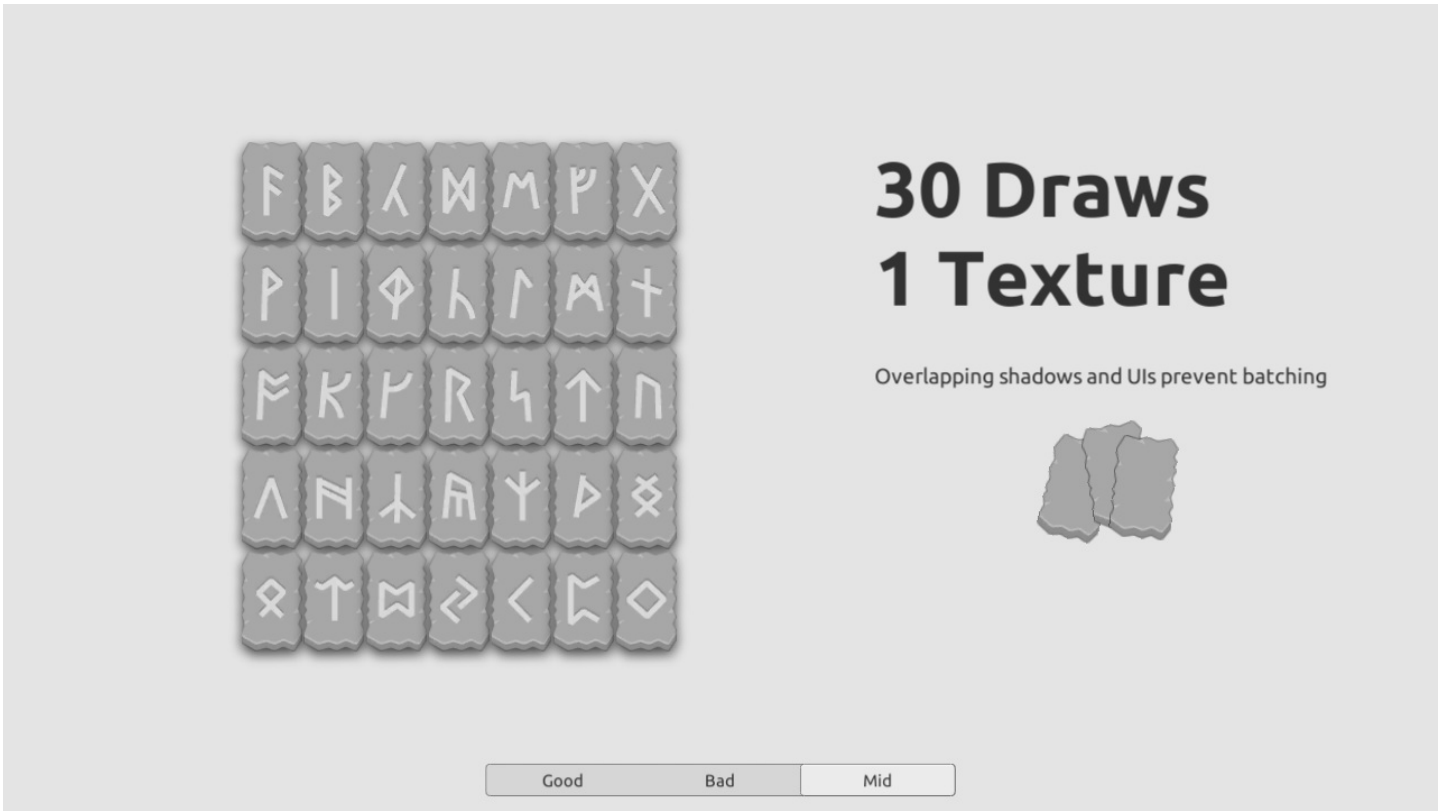
True Shadow can share a single shadow texture between different shadow casters. This reduce shadow generation cost, memory usage and allow batching of draw calls in certain situations.

Only identical shadow caster and shadow settings can share a texture. If you have a lot of similar shadow caster, it can be worthwhile to have a dedicated Sprite as a child shadow caster to improve performance and memory usage:





Even if the shadow texture is shared, overlapping UIs will prevent batching, as usual.



Scripting Considerations

Transform.GetChild

True Shadow displays shadows by creating hidden GameObjects as children of the shadow caster. If you are writing scripts that interact with the hierarchy of the shadow caster in some way, you should keep these hidden objects in mind.

For example, calling `transform.childCount` on the parent will also count the shadow object. The index passed to `transform.GetChild()` must also be modified accordingly.

Generally, it is not a good idea to use `transform.GetChild()` to reference objects. It is brittle to change in the same way a string-based reference is, and even more opaque. A strongly typed reference is always preferable:

- Assign the references using the inspector for scene objects, and in a Prefab for spawned ones.
- Use `FindObjectOfType` or `GetComponentInChildren`.