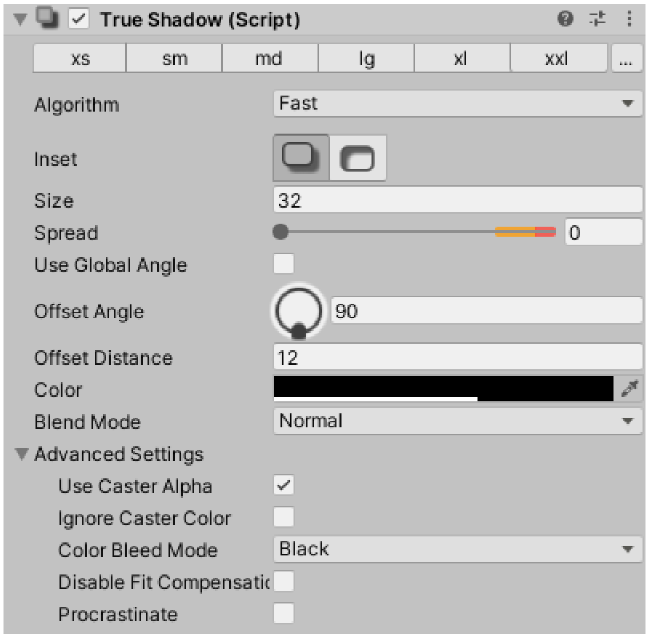
Table of Contents

Getting Started

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

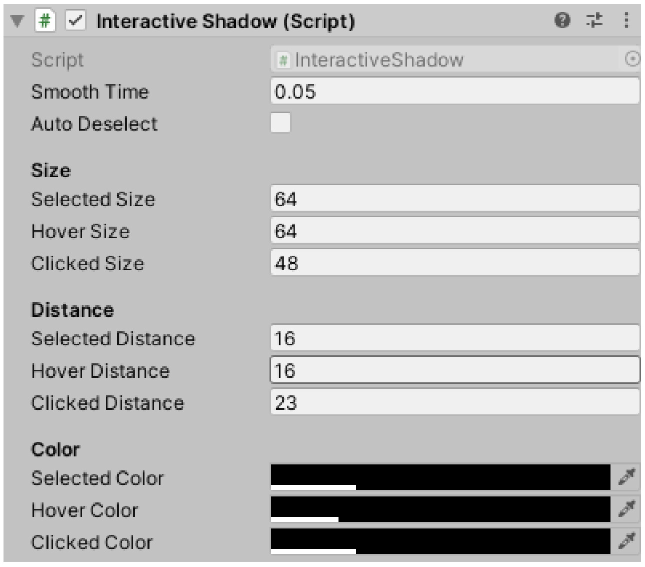


True Shadow (Script)

xs · sm · md · lg · xl · xxl · ...	
Algorithm	Fast
Inset	
Size	32
Spread	0
Use Global Angle	☐
Offset Angle	90
Offset Distance	12
Color	⬛
Blend Mode	Normal
Advanced Settings	
Use Caster Alpha	✓
Ignore Caster Color	☐
Color Bleed Mode	Black
Disable Fit Compensatic	☐
Procrastinate	☐

2. Tune it to your liking.

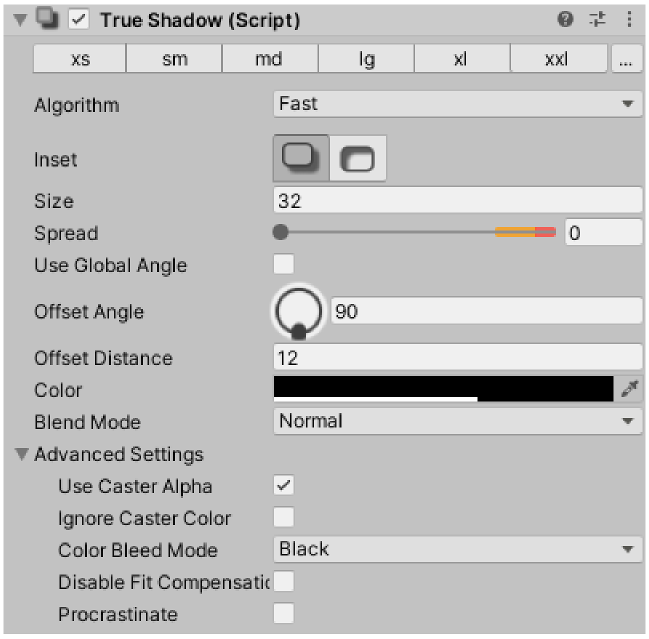
3. Optionally add `Interactive Shadow` to modify shadow properties based on user interaction.



Interactive Shadow (Script)

Script	InteractiveShadow
Smooth Time	0.05
Auto Deselect	☐
Size	
Selected Size	64
Hover Size	64
Clicked Size	48
Distance	
Selected Distance	16
Hover Distance	16
Clicked Distance	23
Color	
Selected Color	⬛
Hover Color	⬛
Clicked Color	⬛

Customize



True Shadow (Script)

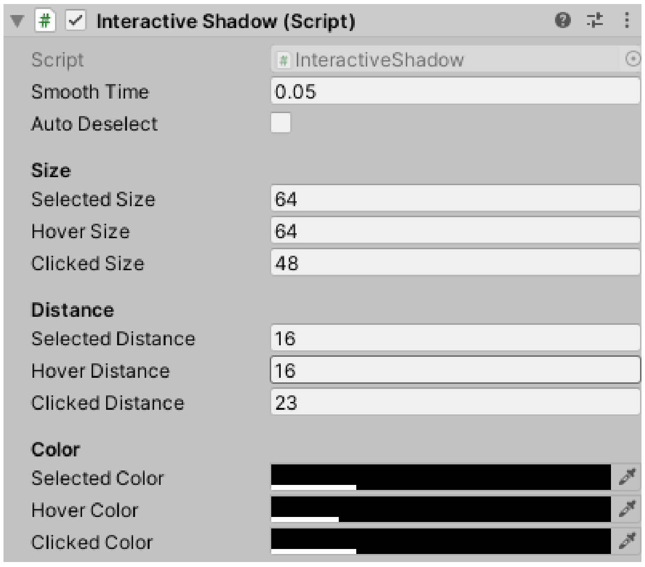
Algorithm	Fast
Inset	
Size	32
Spread	0
Use Global Angle	☐
Offset Angle	90
Offset Distance	12
Color	■
Blend Mode	Normal
Advanced Settings	
Use Caster Alpha	✓
Ignore Caster Color	☐
Color Bleed Mode	Black
Disable Fit Compensatic	☐
Procrastinate	☐

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.



Interactive Shadow (Script)

Script	InteractiveShadow
Smooth Time	0.05
Auto Deselect	☐
Size	
Selected Size	64
Hover Size	64
Clicked Size	48
Distance	
Selected Distance	16
Hover Distance	16
Clicked Distance	23
Color	
Selected Color	■
Hover Color	■
Clicked Color	■

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.

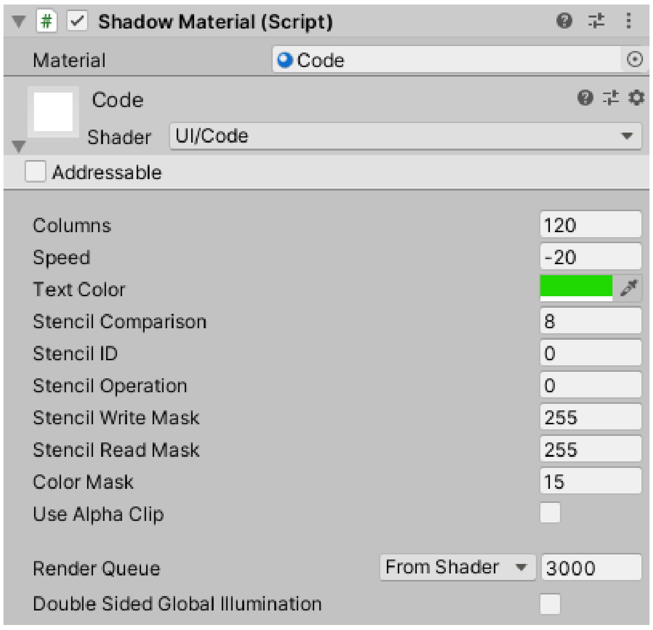
The component supports 3 states: Hovered, Selected and Clicked. These states work the same as the builtin `Selectable`, such as `Button`.

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



Material	Code

Code

Shader: UI/Code

☐ Addressable

Columns	120
Speed	-20
Text Color	🟩
Stencil Comparison	8
Stencil ID	0
Stencil Operation	0
Stencil Write Mask	255
Stencil Read Mask	255
Color Mask	15
Use Alpha Clip	☐
Render Queue	From Shader 3000
Double Sided Global Illumination	☐

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

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

Let's say you have a shader for a dissolve effect on your UI. True Shadow will generate the correct shadow for your UI. However, as you change the dissolve amount, the shadow does not update.

This is because True Shadow can cache generated shadow textures, and even share them between identical looking UI elements. Since normal UI don't have a dissolve property, True Shadow doesn't know the UI has changed.

There are 2 ways to solve this:

- Add the `Disable Shadow Cache` component. This is suitable for effects that update every frame. Or simply if you wish to not do any additional steps.
- Give True Shadow a hash of your properties.

Custom Hash

Step 0

If you have a custom component, you'll want to implement the `ITrueShadowCustomHashProvider` interface to hide True Shadow warning about unsupported components.

Step 1

Calculate a hash from all your custom properties, and set TrueShadow.CustomHash

```
void UpdateTrueShadowHash()
{
    shadow.CustomHash = HashUtils.CombineHashCodes(
        DissolveAmount.GetHashCode(),
        DissolveSoftness.GetHashCode()
        // ...and more as needed
    );
}
```

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.

If your have too many properties, or if they're expensive to access, a random hash also works:

```
    shadow.CustomHash = Random.Range(int.MinValue, int.MaxValue);
```

This make the shadow responds to changes, but prevents reuse of shadow texture between identical copies

Step 2

Update the hash when any of your properties is changed. Some common place are: property setters, OnValidate, custom inspector OnGUI

```
public float DissolveAmount {
    set {
        //... Your logic
        UpdateTrueShadowHash();
    }
}

void OnValidate(){
    UpdateTrueShadowHash();
}

void OnGUI(){
    if (GUI.changed)
        UpdateTrueShadowHash();
}
```

A complete example

This example uses a separate component to keep it simple

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

    void OnEnable()
    {
        shadow = GetComponent<TrueShadow>();
        fx = GetComponent<DissolveEffect>();
    }

    void UpdateTrueShadowHash()
    {
        shadow.CustomHash = HashUtils.CombineHashCodes(
            fx.DissolveAmount.GetHashCode(),
            fx.DissolveSoftness.GetHashCode()
        );
    }

    void Update(){
        fx.DissolveAmount = Time.frameCount * 0.01f;
        UpdateTrueShadowHash();
    }
}
```

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 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. For example:

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

If your asset is a shader without any custom components, users can still benefits from a static method that let 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 4 extra lines of code:

```

class YourClass : ..,
                ITrueShadowCustomHashProvider  // Hide warning
{
    // Reference True Shadow
    TrueShadow shadow;

    void OnEnable(){
        // Get it
        shadow = GetComponent<TrueShadow>();
    }

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

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

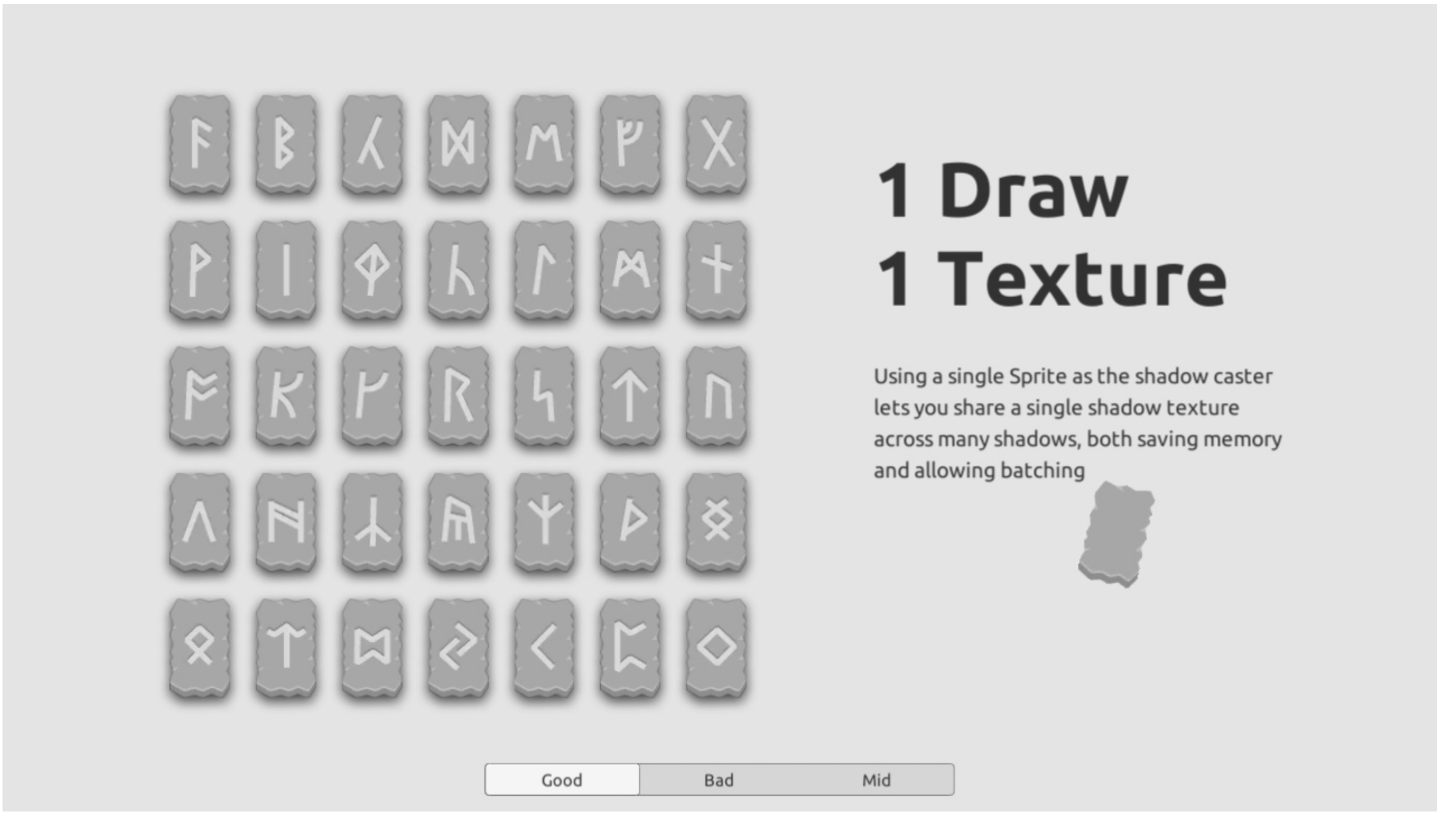
Batching

TIP

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

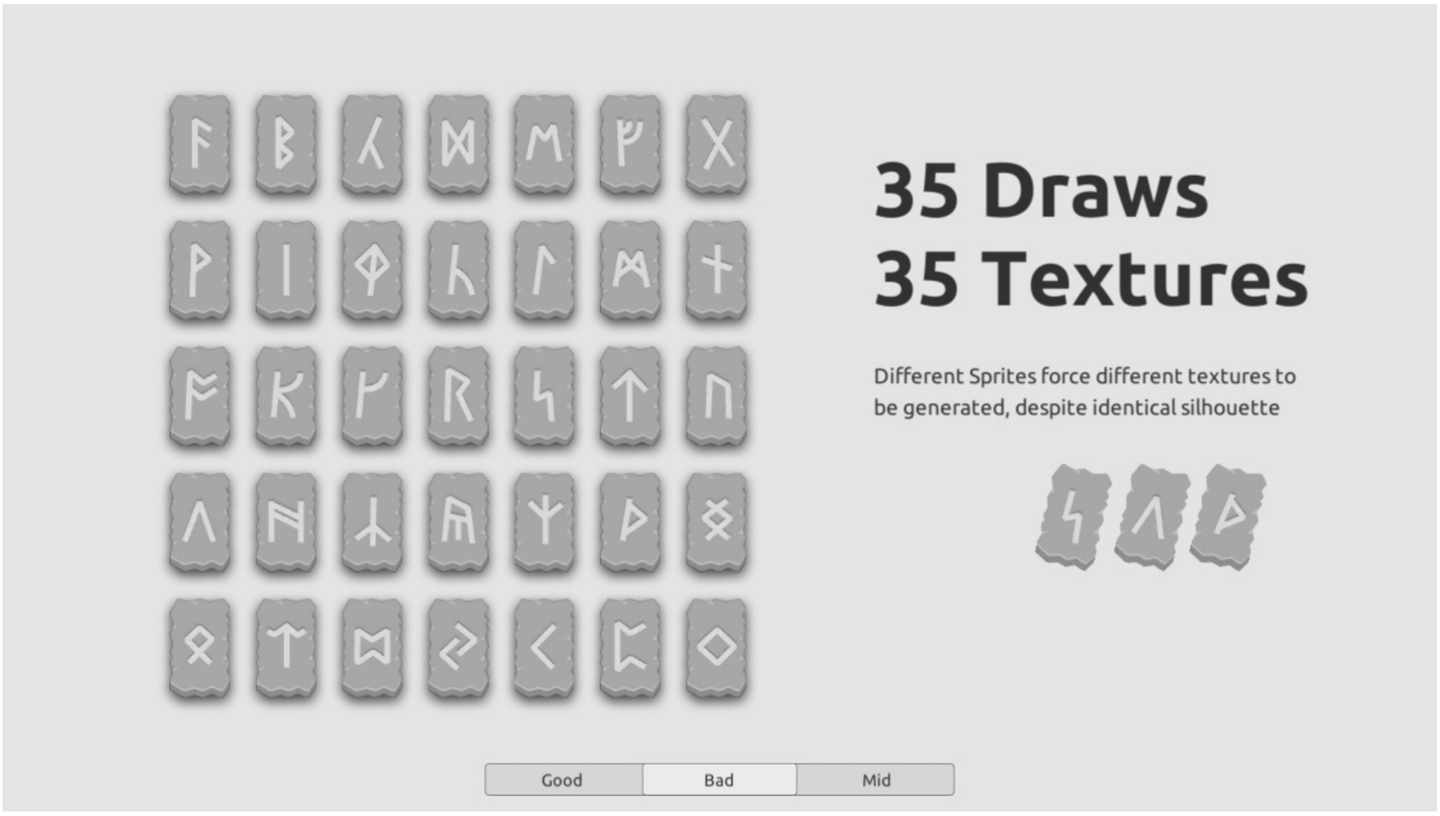
True Shadow can share a single shadow texture between different shadow casters. This reduce shadow generation cost, memory usage and allow batching of draw calls in certain situations.

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:



1 Draw
1 Texture

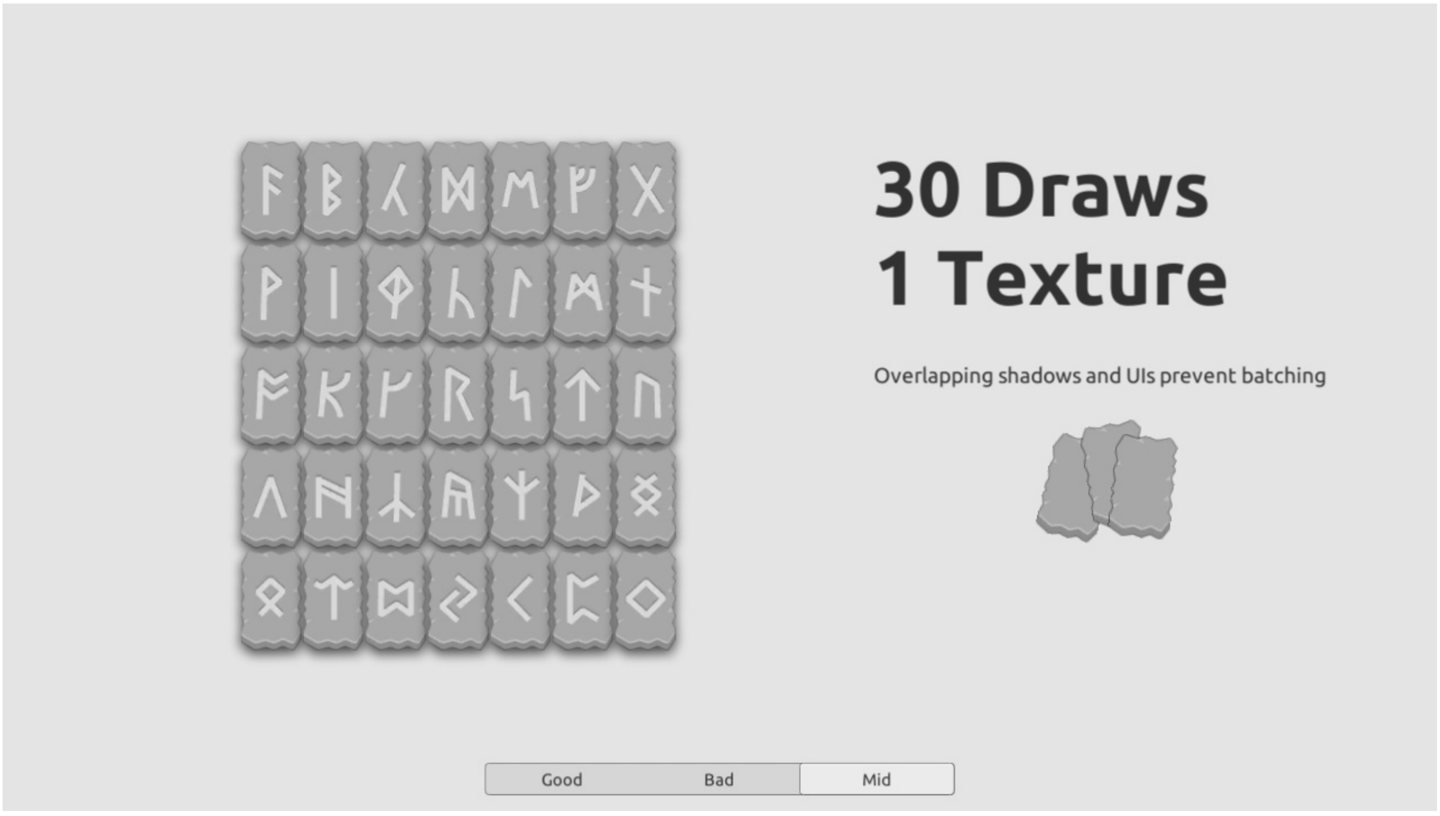
Using a single Sprite as the shadow caster lets you share a single shadow texture across many shadows, both saving memory and allowing batching



35 Draws
35 Textures

Different Sprites force different textures to be generated, despite identical silhouette

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



30 Draws
1 Texture

Overlapping shadows and UIs prevent batching

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`.

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