DTT
APPS WEB GAMES

SINCE 2010

DTT Procedural UI

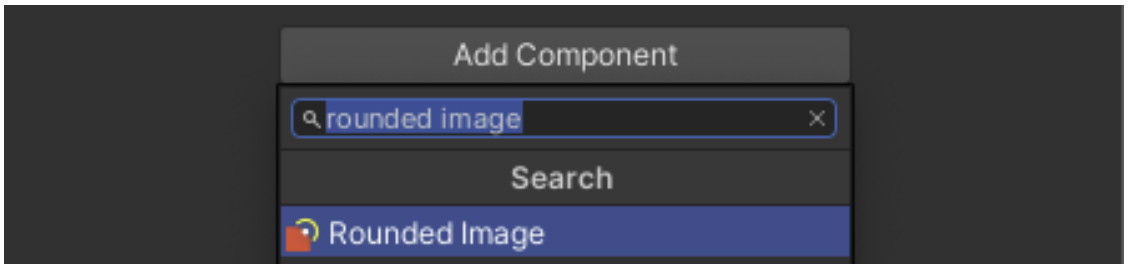
Documentation | V2.0.1 | 15-11-21

Table of Contents

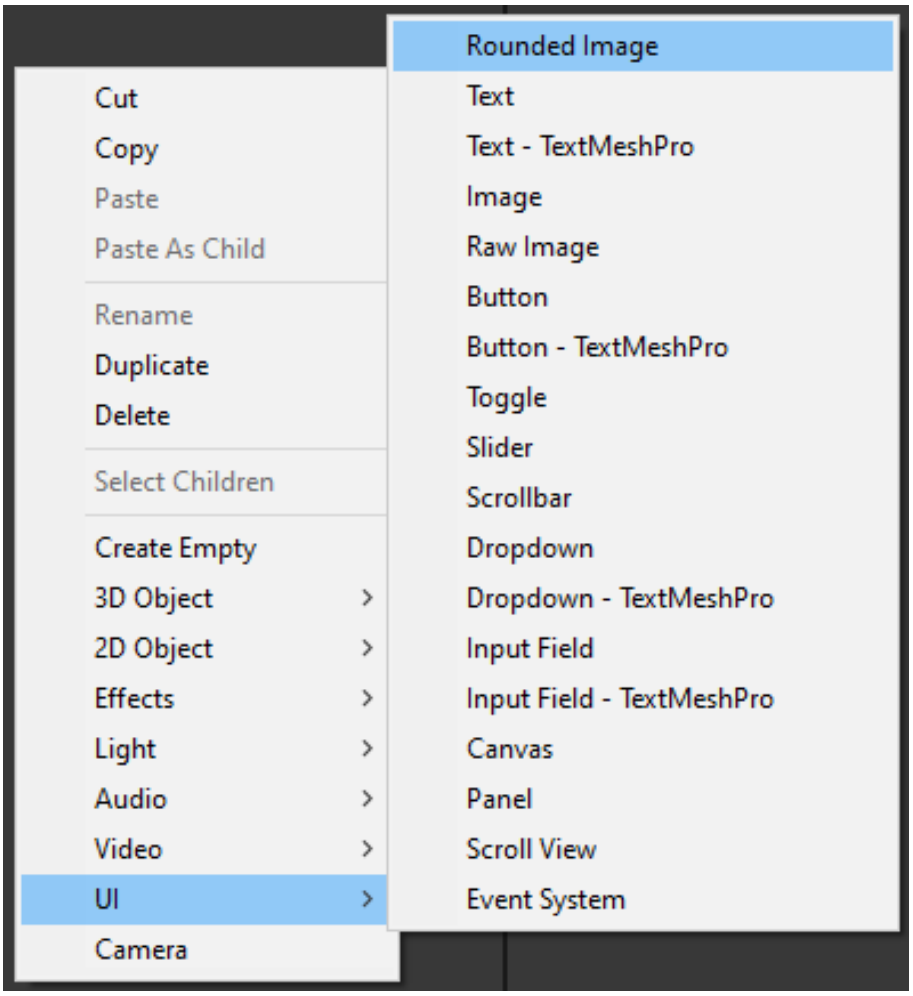
1. Get started quickly

This is a summary to help you get started quickly. More details are provided below.

To get started, use the inspector to add the 'Rounded Image' component to any UI element:



Or use the scene context menu to create a new Rounded Image UI object:



Then, use the **inspector** to adjust the UI element to your desired outcome using:

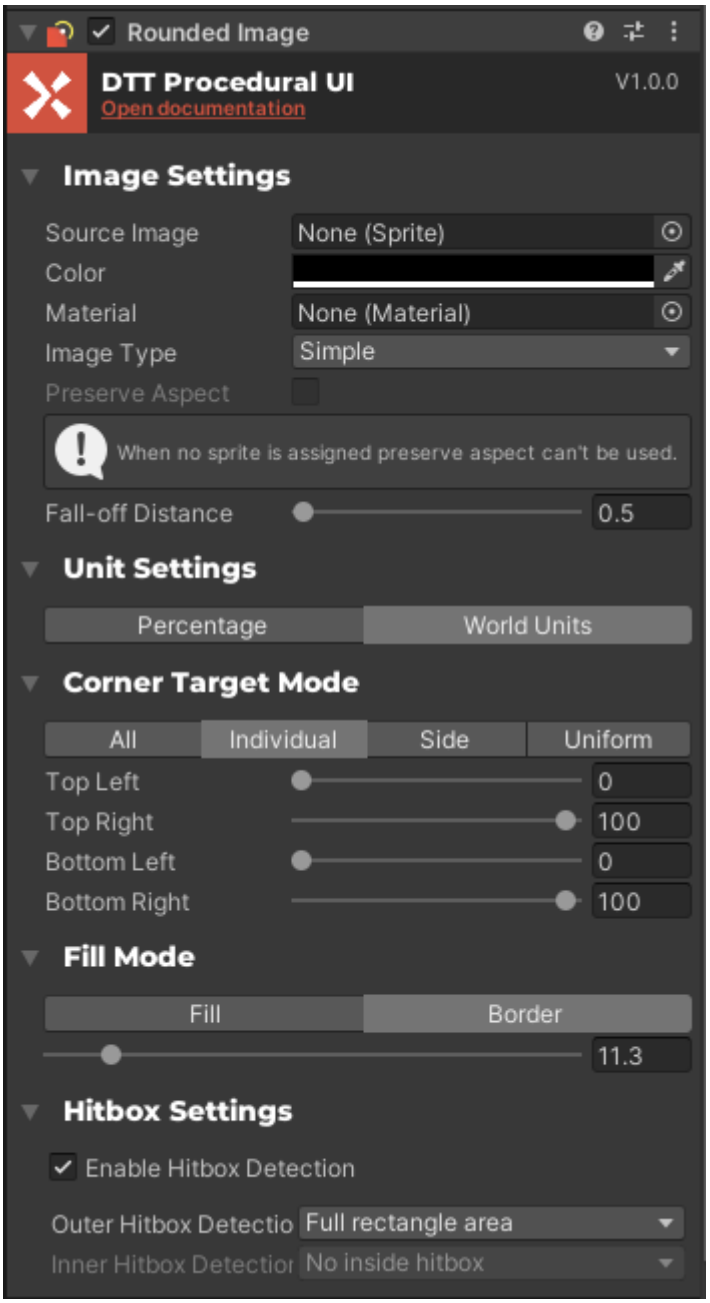


Image Settings

Procedural image is an extension of Image; all Image functionality are compatible, including support for 'sprite mode: multiple' and atlases.

Unit Settings

Allows you to switch the method of rounding.

Corner Target Mode

Choose how to adjust corners. Use the slider to set the amount of rounding.

Fill Mode

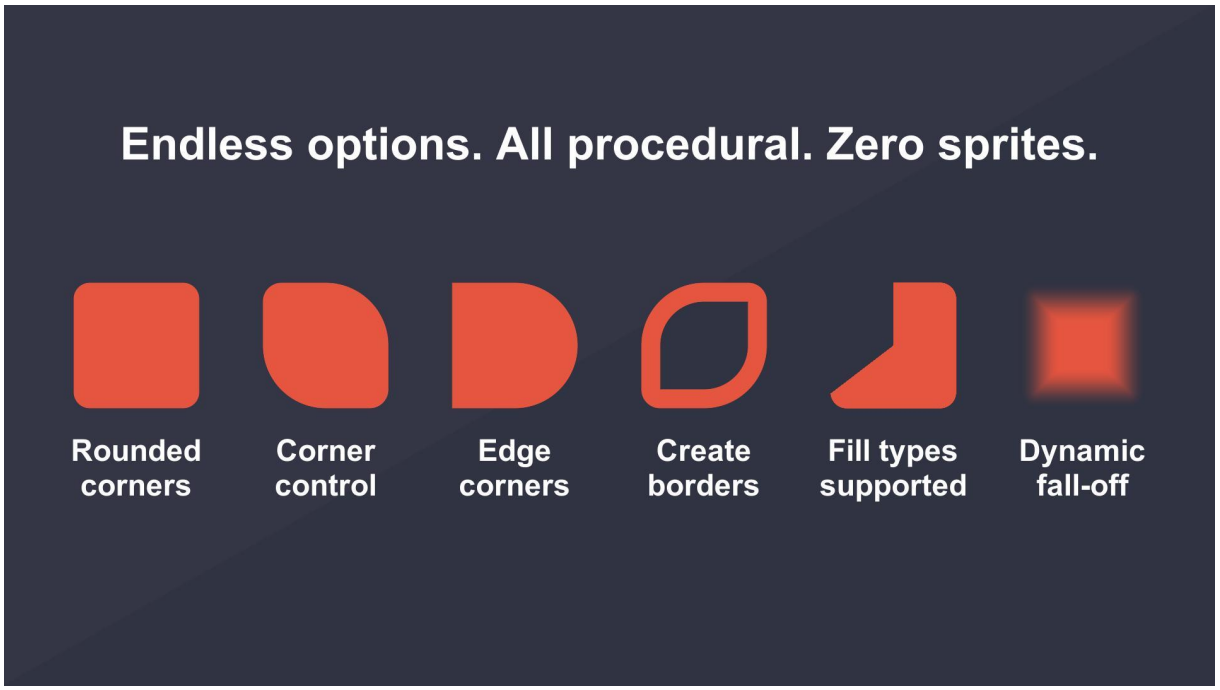
Allows you to set a border, including its width.

Hitbox Settings

Enable hit detection and indicate if you want the hit area to take rounded corners into account.

2. Introduction

DTT Procedural UI functions as an extension of the Unity UI Image Component. You can use it both with and without a sprite, which it renders in real time using shaders. It allows you a greater amount of freedom in creating a flat styled UI without having to create all individual sprites in external editors. This can save you large amounts of disk space, making for a more compact game build. This, combined with our extremely efficient shaders, makes it perfect for mobile development while still being able to achieve stunning effects even on high resolutions. You can also adapt the values in real time during gameplay, allowing you to achieve dynamic effects.

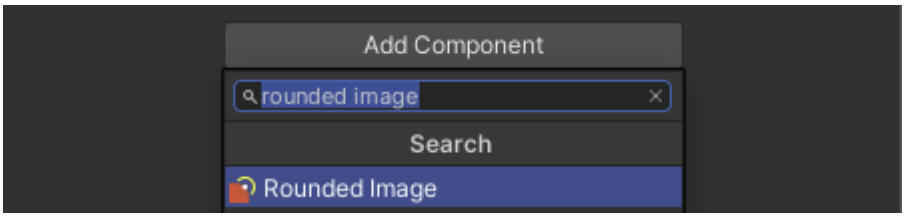


3. Changelog

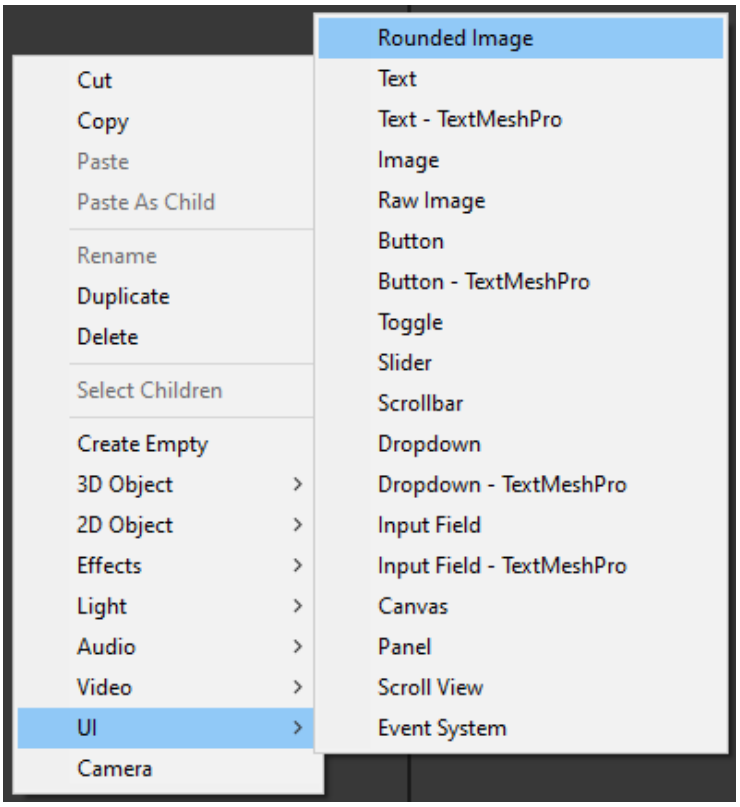
1. version 1.0.0 - initial release
2. Version 2.0.0 - 15-11-2021
    a. Added API section
    b. Fix
        i. Add version define for unity test framework to prevent compile errors from occuring for developers that use visual studio code.
    c. Add
        i. Add **CopyFrom** method to copy values from another RoundedImage component.
        ii. Add **Reset** method to support resetting the component through Unity's editor options.
        iii. Add **SetCornerRounding** method overload using ValueTuple<Corner, float>'s as arguments.
        iv. Add **ValueEquals** method
3. Version 2.0.1 - 19-11-2021
    a. Update
        i. Update Editor Utilities dependency to a more stable version.
4. Version 3.0.0
    a. Added
        i. Added GradientEffect component

4. Set-Up

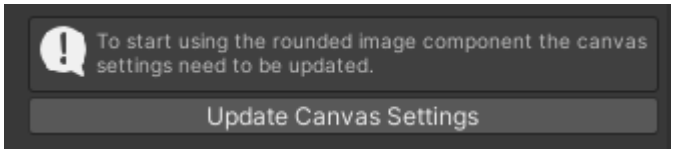
To get started, use the inspector to add the 'Rounded Image' component to any UI element after importing the package:



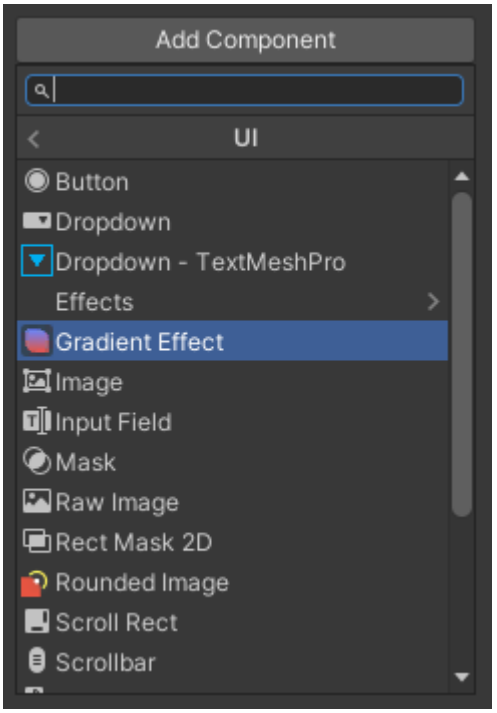
Or use the scene context menu to create a new Rounded Image UI object:



Depending on your cavas settings, you might see the following warning in your inspector after selecting the object:



This most commonly means that your canvas does not have the correct shader channels enabled. Simply click the button and this will be resolved. You're good to go!



Now to add the Gradient Effect component, press the 'Add Component' button at the bottom of the object inspector. Go to 'UI' and choose the Gradient Effect component. You can now add gradient effects to your image!

5. Editor

5.1 Rounded Image

Overview

This is the DTT Procedural UI Editor. It is a powerful tool with which you can adjust your images without using pre-prepared sprites. In this chapter we will break down the functionality of each section of the editor in detail.

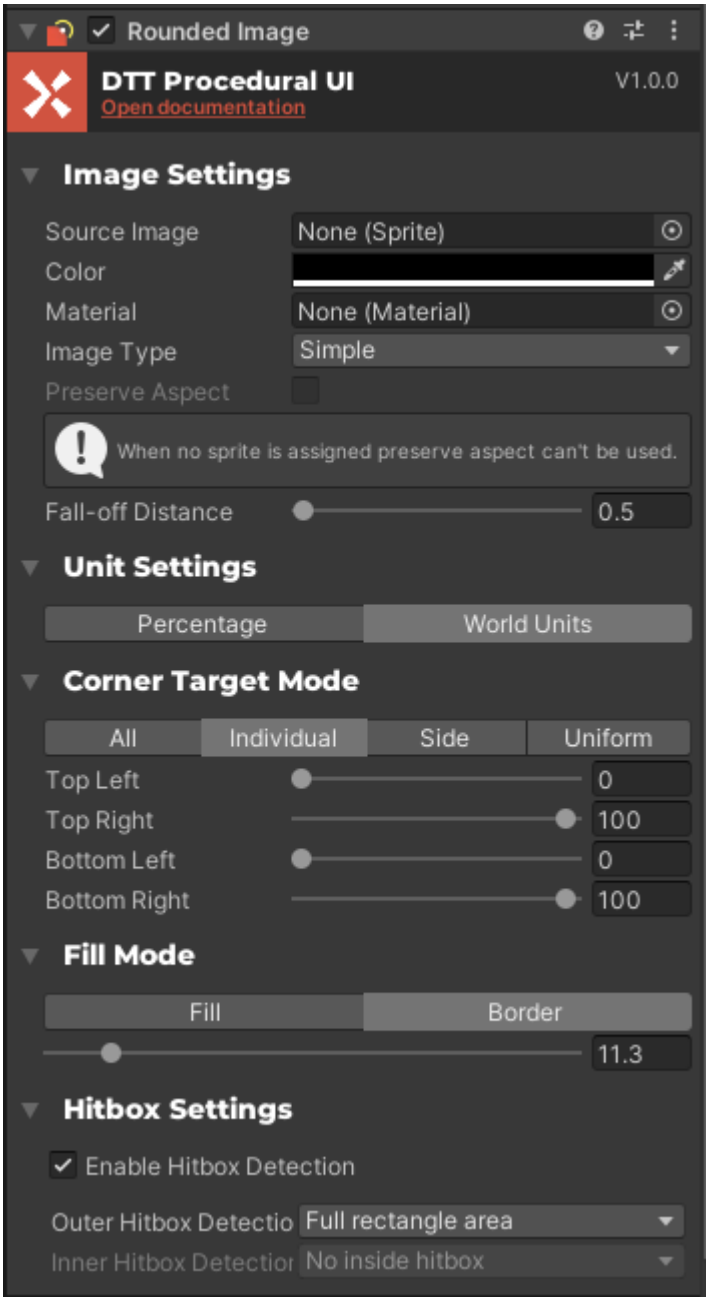
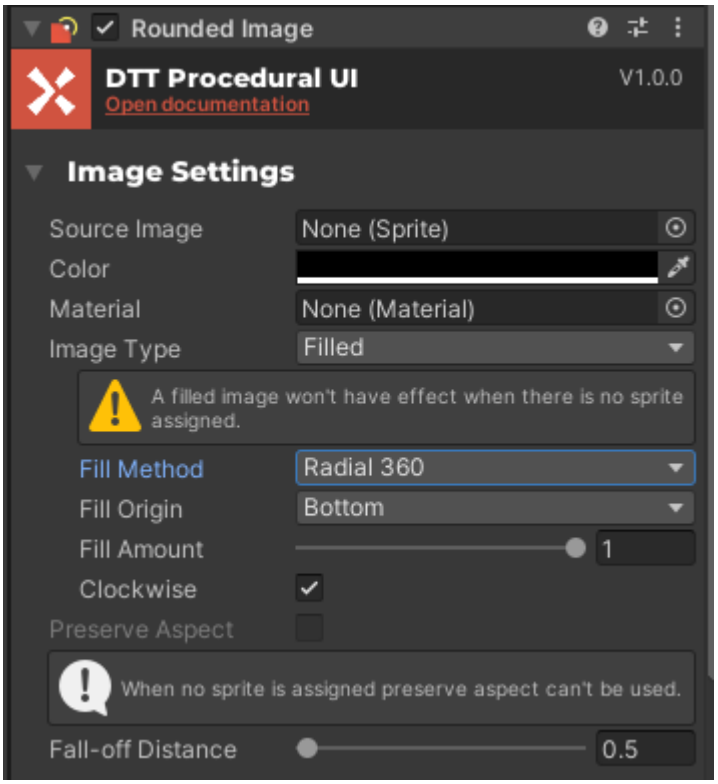


Image Options



As DTT Procedural Image is an extension of the Unity Image Component, you will find that the features you are familiar with are still available to you. This includes selecting sprites to adapt using the editor, changing the colour, filled image type, and sprite mode: multiple and atlases[1], and more.

One new feature you will notice in the bottom of this section is Falloff distance. Using this slider, you can adjust how the edge of your image is rendered. You can use this to create shadows.

[1] *sprite mode: multiple and atlases are not supported in versions of Unity older than Unity 2020.2.*

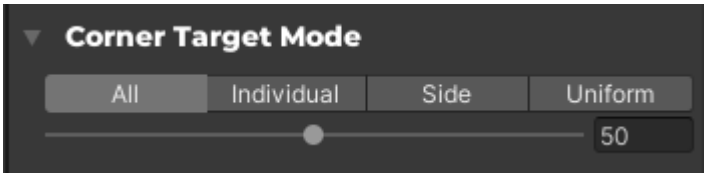
Unit settings

The Unit Settings section lets you choose how the amount of rounding, or the size of your border, is calculated:

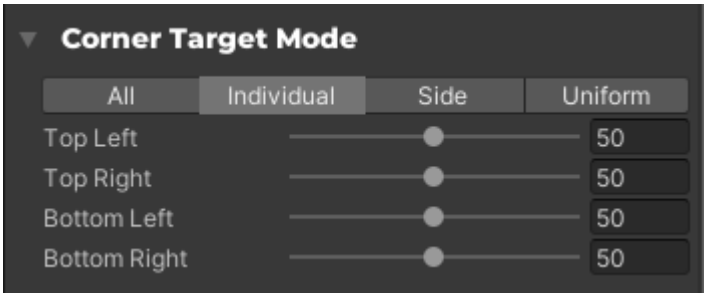
- **Percentage:** You can use a percentage value on your corner and borders. By using this, it will remain visually similar regardless of the size of the object.
- **World Units:** This is a hard value that will persist even if the object is scaled either larger or smaller. It will, however, lock when the maximum amount of rounding has been achieved.

These options give you the flexibility to decide what fits your project and use-case best.

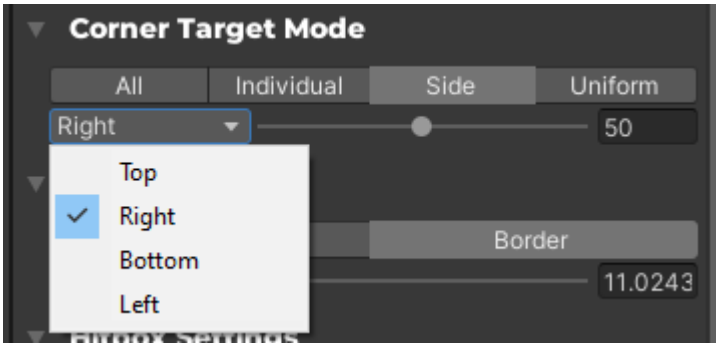
Corner target mode



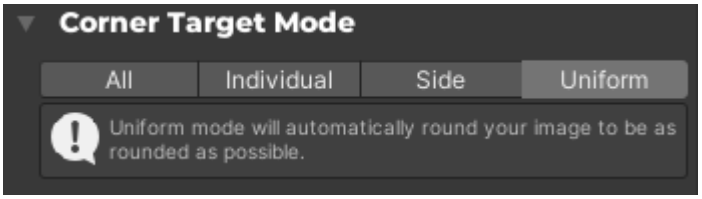
Using the corner target mode section, you can adapt the rounding effect on your image using either percentages or world units. Here you see **All Mode**. This mode rounds all four corners by the given amount. You can change this by using the slider or typing in the value.



When selecting **Individual Mode** you can adjust every corner in minute detail using the four individual sliders.



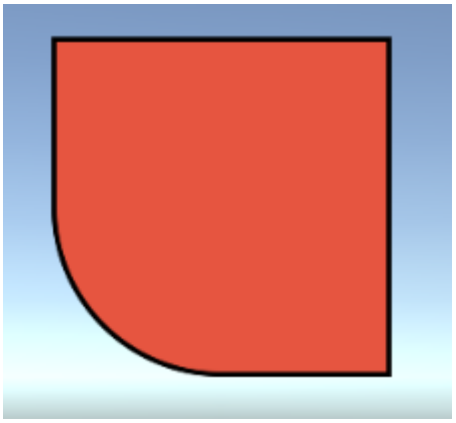
Using **Side Mode** the system will automatically take the two relevant corners and round them together. This allows you to quickly give your UI an edge without spending too much time on fine tuning.



Uniform Mode automatically rounds all sides to the maximum value with a single click, turning squares into circles. This very easily gives your UI that slight polished edge it might be lacking.

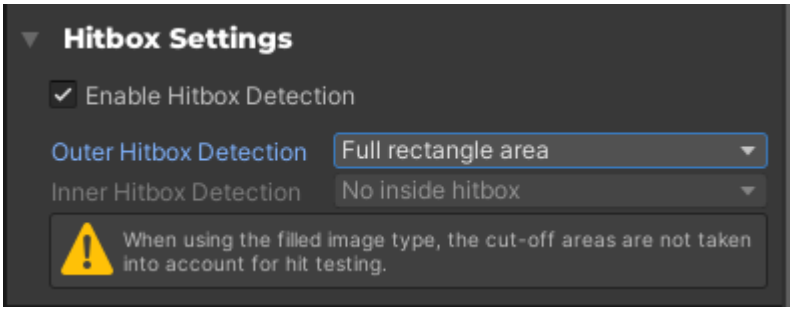
Fill mode

Fill Mode allows you to use the image as a border by opening a hole in its center. Using the slider, you can determine the size of your border in either percentages or world units. This can be a very powerful tool to create versatile effects.



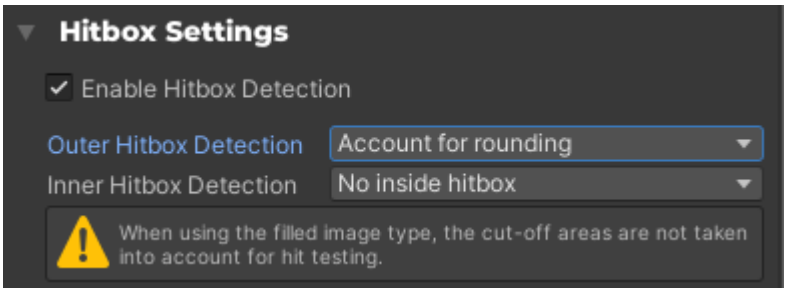
Here we used 2 objects to create this button. A black image with border mode turned on and set to 3%, and a separate image to serve as the clickable area.

Hitbox settings

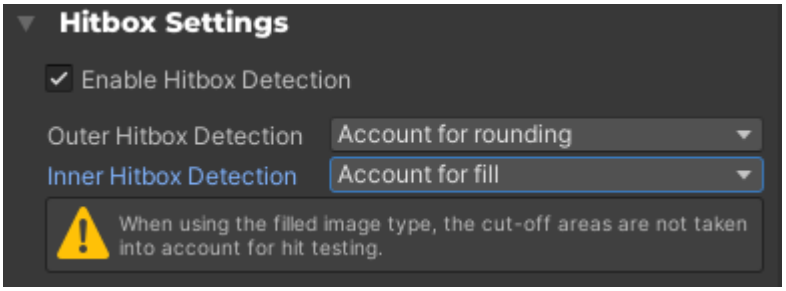


DTT Procedural UI comes with a custom hitbox solution. This allows you to dynamically adapt the interactive areas of your buttons to fit the new visuals, or ignore the new visuals if you so desire. If you set the values as shown in the above image, you can expect identical results as with using Unity's hit detection.

Also note that it is showing a warning related to 'filled image type'. The editor will always inform you if you are at risk of relying on an unsupported feature.



With 'Account for rounding' turned on, any effects applied to the outside of your image will be taken into account when checking for clicks or taps.



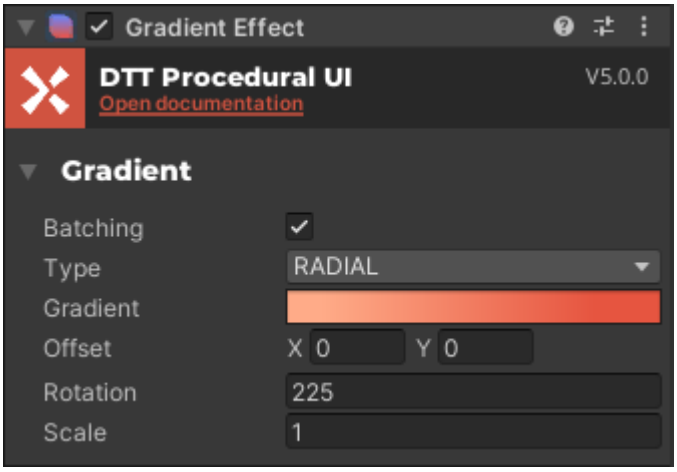
With 'Account for fill' turned on, any effects applied to the inside of the image, such as fill mode, will be taken into account[1].

Together these options give you the ability to choose where and when the user can click successfully.

[1] *Note that this references our fill mode option you can use to create borders. The standard Unity fill mode image type is currently not supported by this feature.*

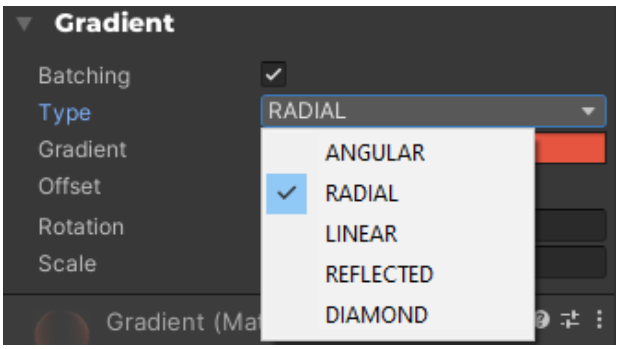
5.2 Gradient Effect

Gradient

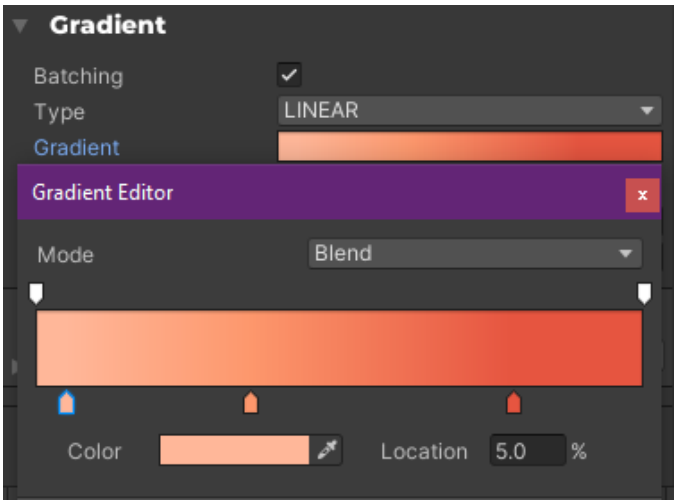


The DTT Procedural UI asset also comes with a component that allows you to easily add gradients to your UI images. The gradient component has a set of options that allow you to change the gradient effect. It works with and without sprites, and even works with the Rounded Image component.

The **Batching** option allows you to choose whether the gradient should use batching. This disables the ability to change the properties of the gradient.



The **Type** lets you specify which way the gradient should be drawn. Some examples can be seen above.



The **Gradient** field allows you to change the colours of the gradient. The top bar allows you to edit the alpha, and the bottom bar the color.

Offset allows you to edit the offset of which the gradient is drawn on the image.

Rotation rotates the gradient effect on the image. Lastly, **Scale** defines the size of the gradient effect on the image.

6. API

While we generally recommend working with DTT Procedural UI through the included editor, you might have the need to update the state of objects during your application's lifetime. The API does support this, and some examples are included below.

6.1 Rounding API

```
private void Start()
{
    // Get your rounded image reference.
    RoundedImage image = GetComponent<RoundedImage>();

    // Retrieve individual corner rounding.
    float bottomLeftRounding = image.GetCornerRounding(Corner.BOTTOM_LEFT);

    // Retrieve a dictionary with corner rounding info.
    ReadOnlyDictionary<Corner, float> rounding = image.GetCornerRounding();

    // Update individual corner rounding.
    image.SetCornerRounding(Corner.TOP_LEFT, 0.5f);

    // Update all corner rounding.
    image.SetCornerRounding(0.5f);

    // Update all corners individually.
    image.SetCornerRounding(0.25f, 0.15f, 0.35f, 0.5f);

    // Update individual corners using Tuples.
    image.SetCornerRounding((Corner.TOP_LEFT, 0.5f), (Corner.BOTTOM_LEFT, 0.25f));
}
```

6.2 Additional options for Rounding

```
private void Start()
{
    // Get your rounded image reference.
    RoundedImage image = GetComponent<RoundedImage>();

    // Copy values over from another rounded image.
    image.CopyFrom(otherImage);

    // Set border thickness.
    image.BorderThickness = 0.5f;

    // Set rounding unit.
    image.RoundingUnit = RoundingUnit.WORLD;

    // Set rounding mode.
    image.Mode = RoundingMode.BORDER;
}
```

6.3 Gradient API

```
private void Start()
{
    // Gets the gradient component.
    GradientEffect gradientEffect = GetComponent<GradientEffect>();

    // Allows you to enable or disable batching.
    // To change the gradient properties, batching must be turned off.
    gradientEffect.Batching = false;

    // Set the type of the gradient to the given GradientType.
    gradientEffect.Type = GradientEffect.GradientType.LINEAR;

    // Set the gradient.
    gradientEffect.Gradient = new Gradient();

    // Set the offset of the gradient.
    gradientEffect.Offset = new Vector2(1, 0);

    // Set the rotation of the gradient.
    gradientEffect.Rotation = 180f;

    // Set the scale of the gradient.
    gradientEffect.Scale = 2f;

    // Updates the gradient.
    // This doesn't need to be called when properties are changed.
    gradientEffect.UpdateGradient();
}
```

7. Known Limitations

- **Unity Fill Image:** Unity Fill image type effects are not taken into consideration with our custom hit detection (much like with the standard unity implementation). Research into this is ongoing.
- **The Readme:** If you close the Unity Editor while the readme panel is open it might prevent you from re-opening it using the editor button.
    - You can resolve this by resetting your layout (windows/layouts).
- **Sprite mode: Multiple:** Due to limitations within the engine, Sprite mode: multiple is not supported in versions older than Unity 2020.2.
- **Atlases:** Due to limitations within the engine, atlasas are not supported in versions older than Unity 2020.2.
- **Grey Buttons:** If you are working within a canvas group and your game object has an inactive button component, setting the canvas group's 'interactable' value to False can cause the image to render in gray.
    - Turning the button component on and off again will resolve this.
    - If you have removed the button component, re-adding it will resolve this.

8. Support and feedback

If you have any questions regarding the use of this asset, we are happy to help you out. Always feel free to contact us at:

unity-support@d-tt.nl

(We typically respond within 1-2 business days)

We are actively developing this asset, with many future updates and extensions already planned. We are eager to include feedback from our users in future updates, be they 'quality of life' improvements, new features, bug fixes or anything else that can help you improve your experience with this asset. You can reach us at the email above.

Reviews and ratings are very much appreciated as they help us raise awareness and to improve our assets.

DTT stands for Doing Things Together

DTT is an app, web and game development agency based in the centre of Amsterdam. Established in 2010, DTT has over a decade of experience in mobile, game, and web based technology.

Our game department primarily works in Unity where we put significant emphasis on the development of internal packages, allowing us to efficiently reuse code between projects. To support the Unity community, we are publishing a selection of our internal packages on the Asset Store, including this one.

More information about DTT (including our clients, projects and vacancies) can be found here:

https://www.d-tt.nl/en/