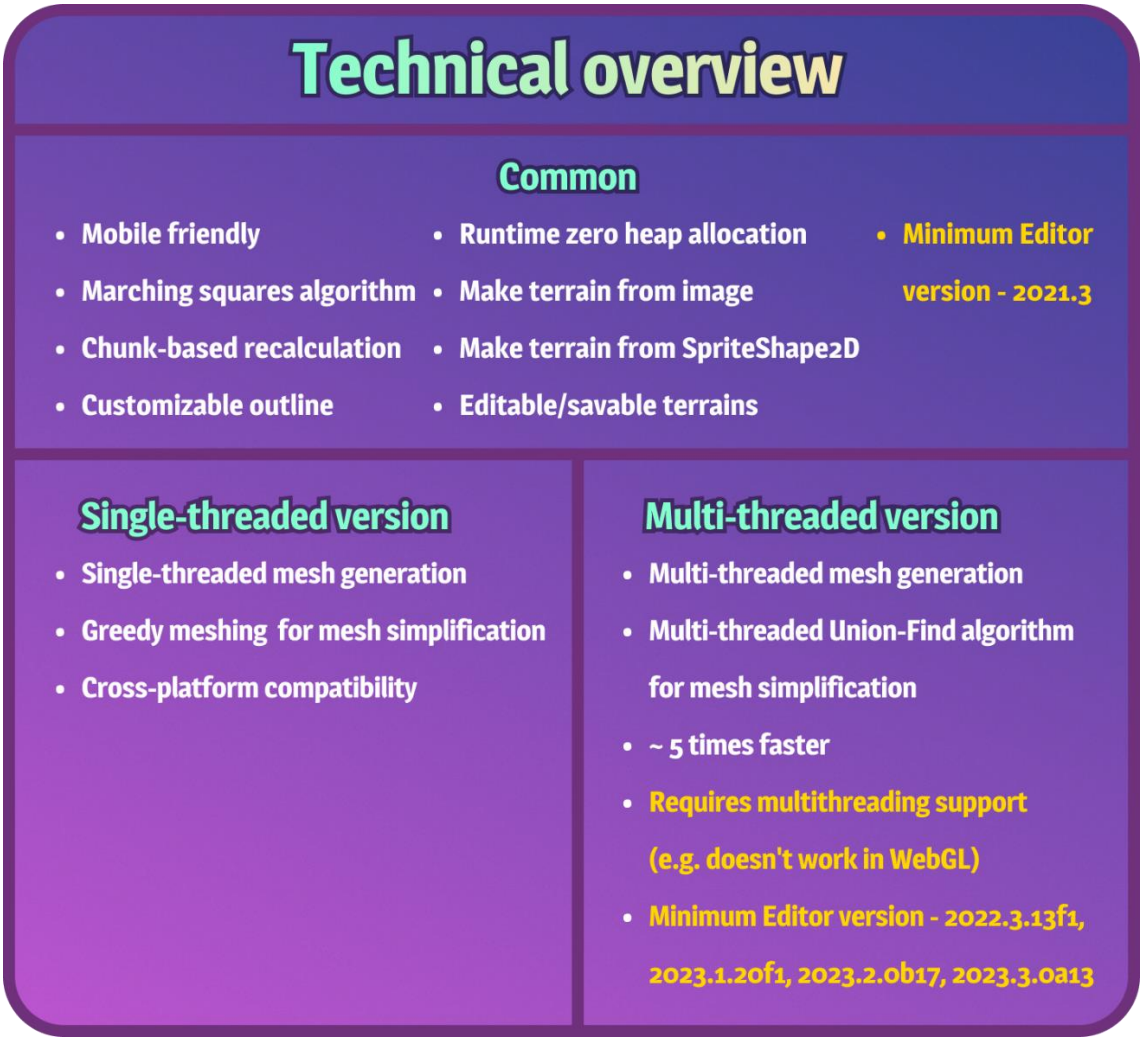
Terraforming Terrain 2D

Thank you for purchasing! I would be so grateful if you left your feedback at the Asset Store. It means a lot to me and will help to improve the asset.

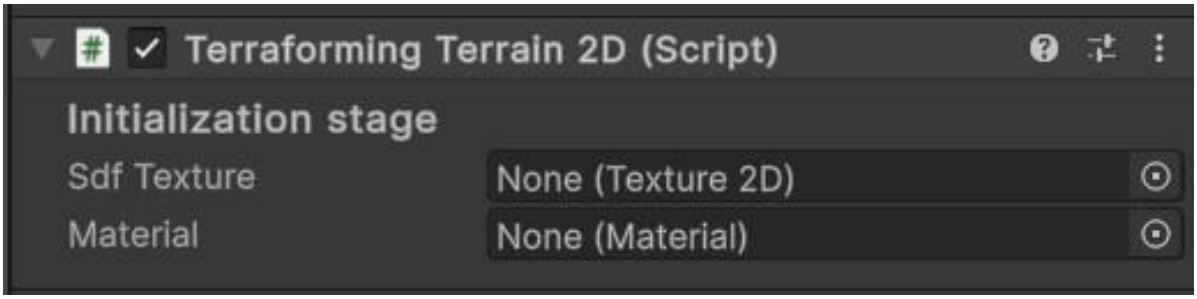
Also, if you have any questions or proposals, don't hesitate to contact me via dunnospace@gmail.com. I would be glad to help.

Overview



Setup

1. Create an SDF texture using **SDFTextureFromImageGenerator** or **SDFTextureFromSpriteShapeGenerator**
2. Attach a **TerraformingTerrain2D** script to an empty **GameObject**



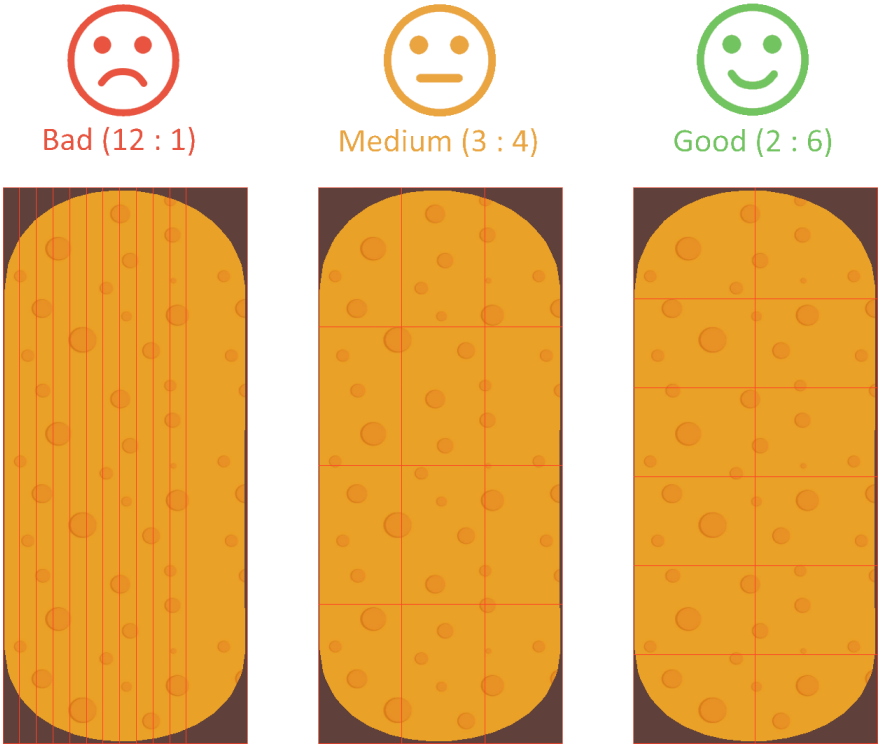
3. Set your generated SDF texture.
4. Set material with ***"Dunno/Terrain"*** shader.
5. Here you go :)

Chunk partition

Terraforming Terrain 2D uses procedural mesh generating for terraforming and it is crucial to avoid heavy operations and **recalculate only affected chunks**.

Therefore, the *Chunk section* is the most important part with the greatest impact on **performance.**

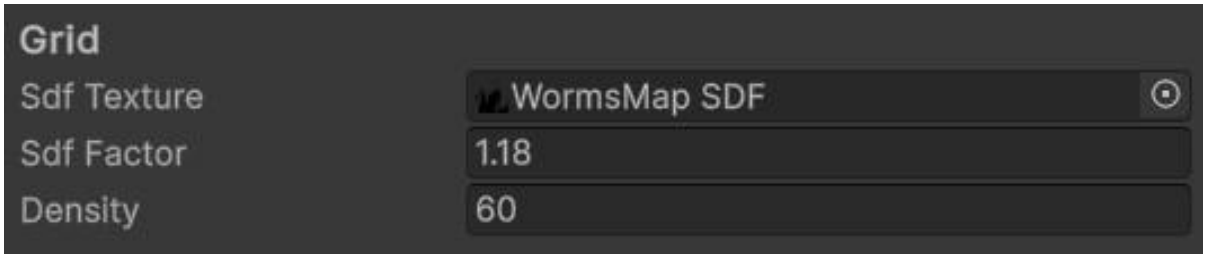
General tip: Try to keep your chunks in a way the ratio of their sides is minimal. This ensures that interaction with terrain involves **minimal chunks.**



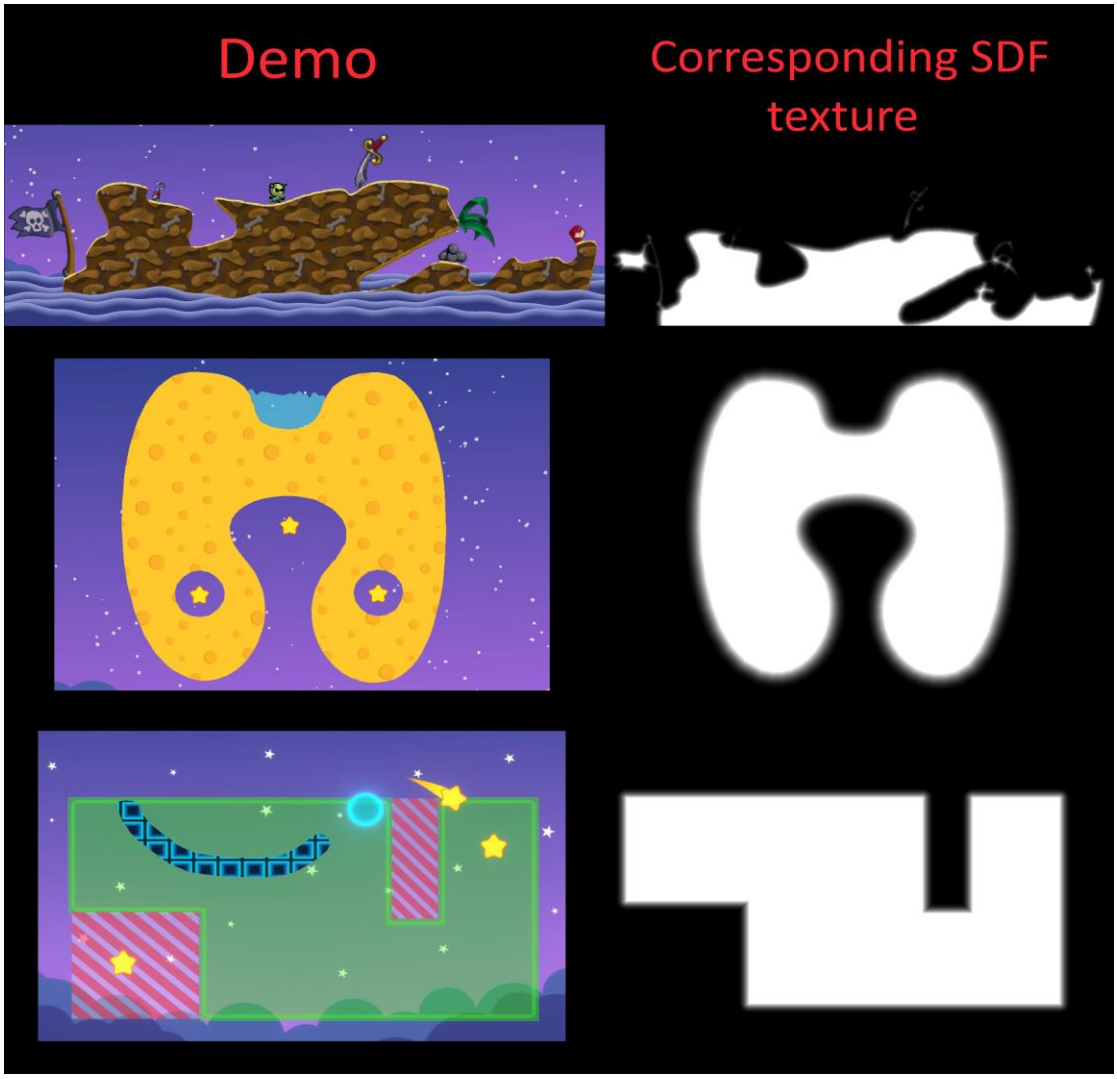
Each chunk is rendered in a separate **draw call.**

General tip: Don't create too many chunks. Find a balance between chunk recalculation (**ALU bound**) and rendering (**Draw call bound**).

Grid section

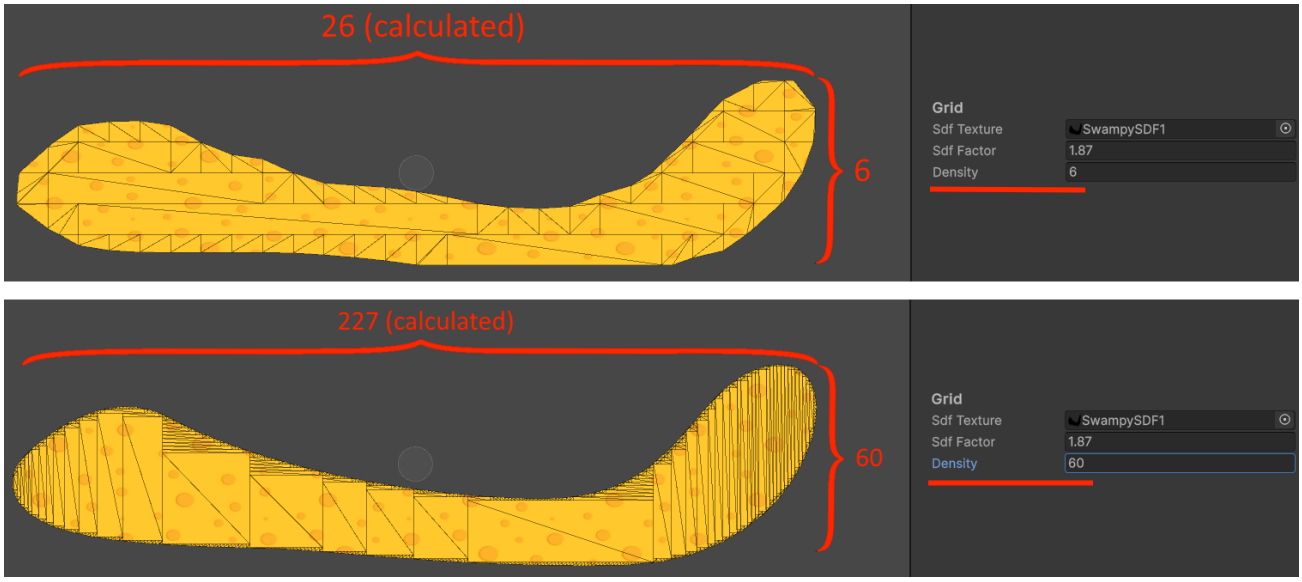


SDF Texture defines the shape of your terrain. This is the main required input property from the user.



SDF factor defines how strongly algorithm should shape contour (terrain) according to *SDF Texture*

Density defines the count of marching squares at the Y-axis. The X-axis is calculated automatically using the ratio of the SDF Texture



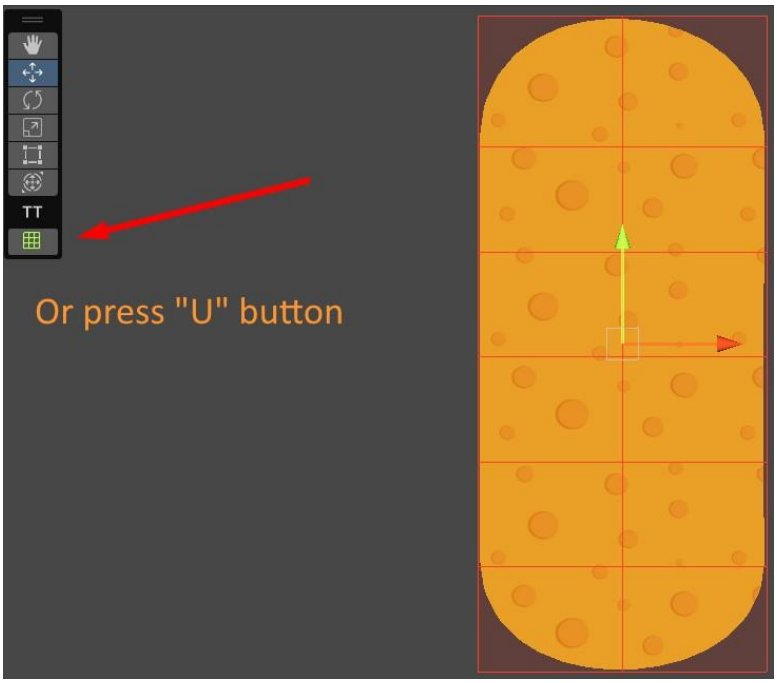
Marching Squares is an algorithm that generates an **approximation** for a contour line of a two-dimensional scalar field. That means that TerraformingTerrain2D doesn't create a terrain that is an **exact match** to an original image / SpriteShape2D.

However, using the *Density* property you can reduce the error between the original image / SpriteShape2D and the generated terrain.

General tip: Use this property wisely. A higher *Density* value means higher visual quality but at the same, it requires more CPU/GPU calculations. The *Density* property has the second greatest impact on **performance** after a chunk partition.

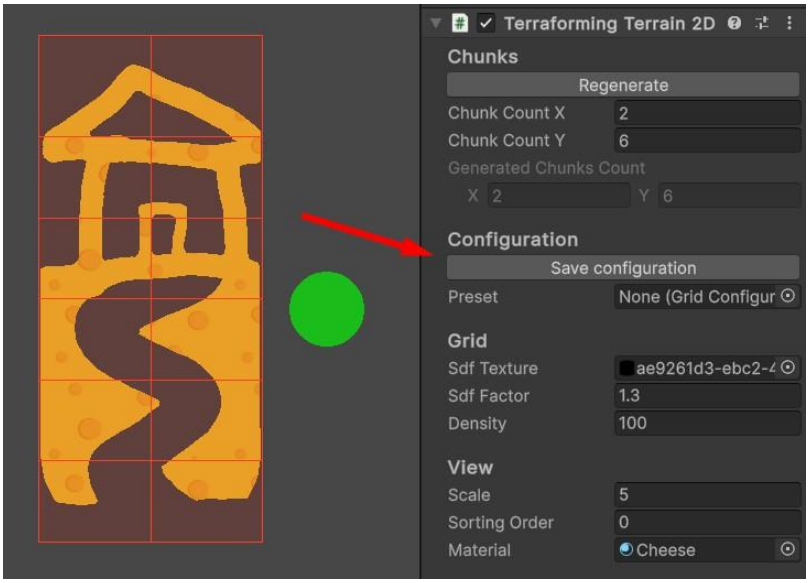
Manual editing and configuration

To start manually editing your terrain - select your terrain **GameObject** and click Edit tool or press the **'U'** button.

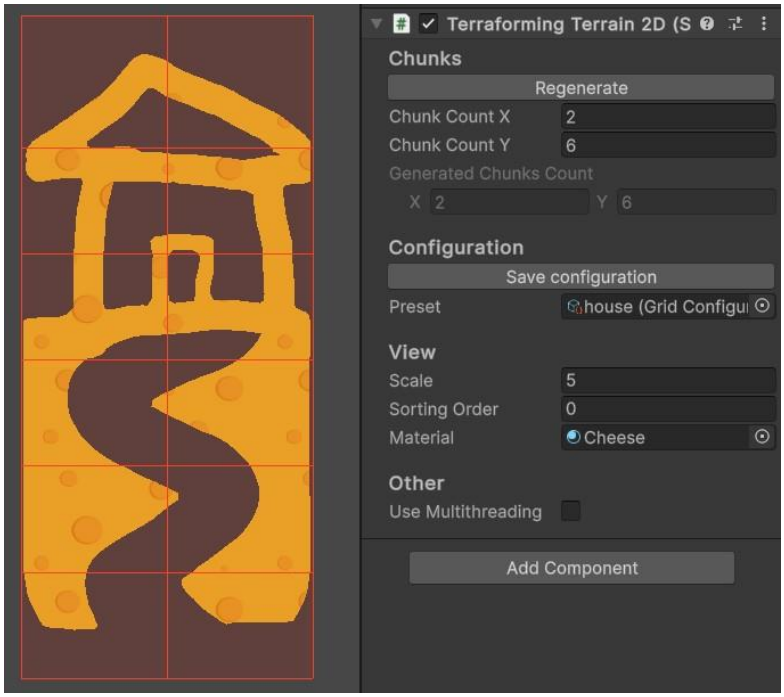


Hold the Ctrl button and use LMB for erasing terrain and RMB for painting. Use the wheel to change the radius.

After you are done click the *Save configuration* button.



Be aware: save your configuration right away you finish editing. Otherwise, your changes will be overwritten.



After saving a configuration file - you can't change any property of the *Grid section* anymore. However, you can still manually edit your terrain and override your existing configuration.

How to terraform?

The asset provides one entry point for terraforming **– method Terraform**. You can accept it directly from the **TerraformingTerrain2d** component or any of its chunks.

```
public class TerraformingTerrain2d: MonoBehaviour
{
    public TerraformingTerrainData Data => _data;

    public void Terraform(Vector2 position, float radius, TerraformingMode mode)

    public void Clear()
}
```

```
chunk.TerraformingPresenter.Rebuild(position, radius, terraformingMode);
```

Terraform() will … well terraform :) Use *TerraformingMode.Carve* for carving and *TerraformingMode.Fill* for painting

Clear() will remove all modifications made to the terrain, restoring it to its default state.

General tip: If you want to terraform **continuously** like in the "Where is my water?" – take direct reference to TerraformingTerrain2d and call method *Terraform()*

```
public class Brush : MonoBehaviour
{
    private TerraformingTerrain2D _terrain;

    private void Update()
    {
        if (Input.GetMouseButton(0))
        {
            _terrain.Terraform(transform.position, 2f, TerraformingMode.Carve);
        }
    }
}
```

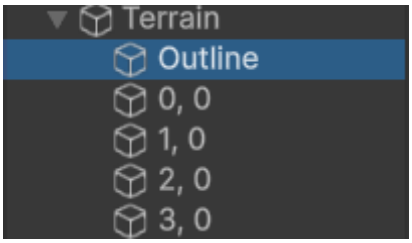
But if you want to terraform from an explosion (like in “Worms” you can get a chunk component from OnTriggerEnter2d) and call *Rebuild* from it

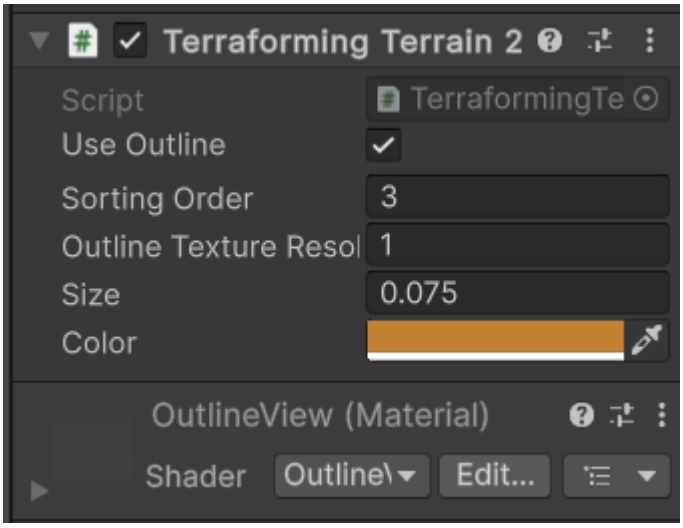
```
public class Missle : MonoBehaviour
{
    // Making sure that the missile will explode once as
    // OnTriggerEnter2D could be called a few times
    private bool _wasExploded;

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (_wasExploded == false)
        {
            if (other.TryGetComponent(out TerraformingTerrain2dChunk chunk2D))
            {
                _wasExploded = true;
                chunk2D.TerraformingPresenter.Rebuild(transform.position, 2f, TerraformingMode.Carve);
            }
        }
    }
}
```

Outline

Outline **GameObject** is created right away when you create a terrain. Just select an *Outline* **GameObject** and browse some settings.





Use Outline property only defines if the outline needs recalculation. If you want to turn off outline visibility – just switch off **GameObject** or **MeshRenderer.**

Outline Texture Resolution defines the multiplier that scales the resolution of the RenderTexture that would be created. RenderTexture resolution is created with the following formula:

RT resolution = SDF Texture resolution * *Outline Texture Resolution*

General tip: Use this property wisely. Higher *Outline Texture Resolution* means higher visual quality but at the same, it requires more GPU calculations. Try to keep it around 1-2.