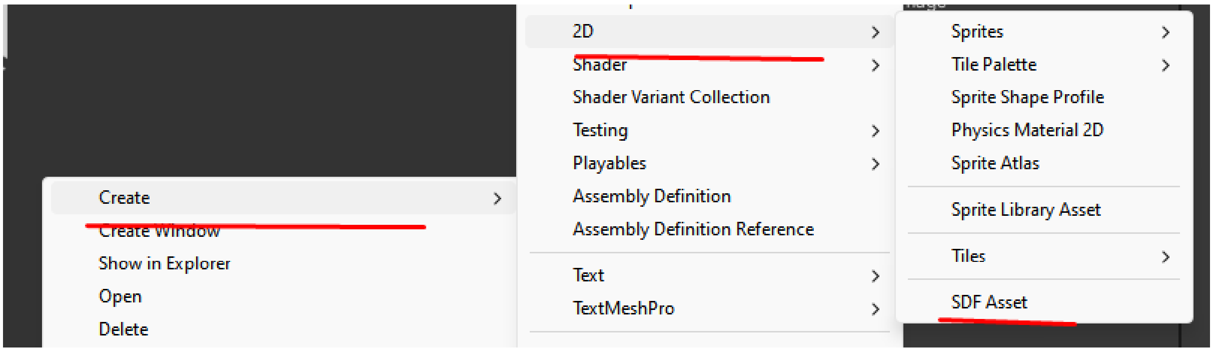
Starting Decoupled Pipeline

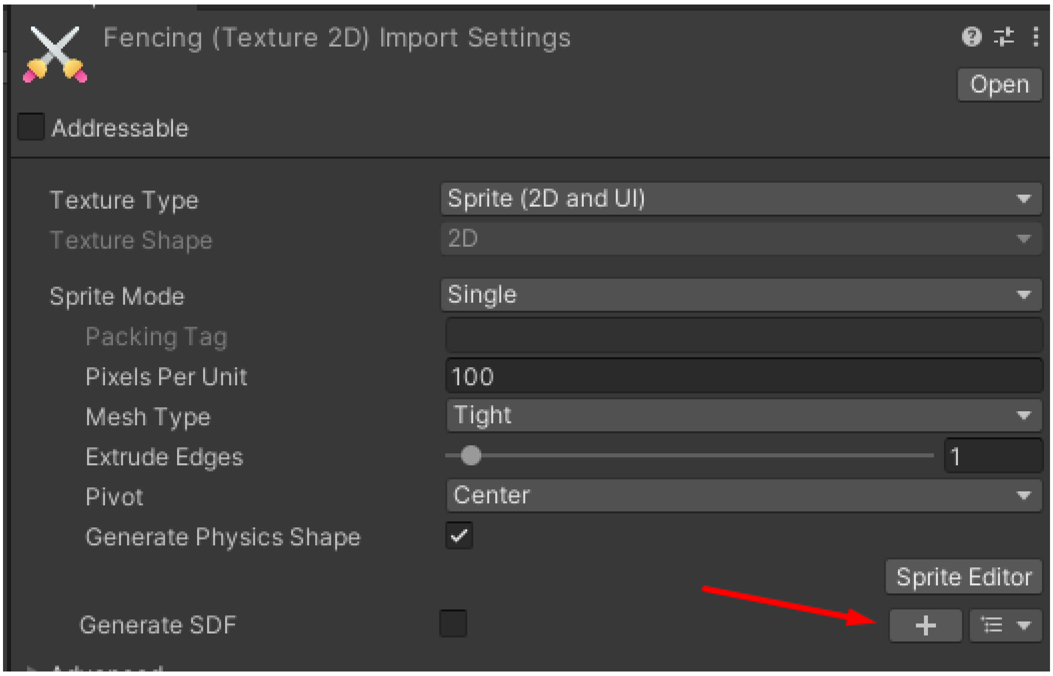
Sample files
SDFAsset
How to find which SDFAsset using Texture?
Why Decoupled Pipeline is useful?
Want to know more?

Version 1.2x introduced decoupled pipeline, this is new way to import SDF sprites.

`SDFAsset` is a file with extension *.sdfasset handled by custom importer.



It can be created in context menu.



Fencing (Texture 2D) Import Settings

Open

□ Addressable

Texture Type: Sprite (2D and UI)

Texture Shape: 2D

Sprite Mode: Single

Packing Tag:

Pixels Per Unit: 100

Mesh Type: Tight

Extrude Edges: 1

Pivot: Center

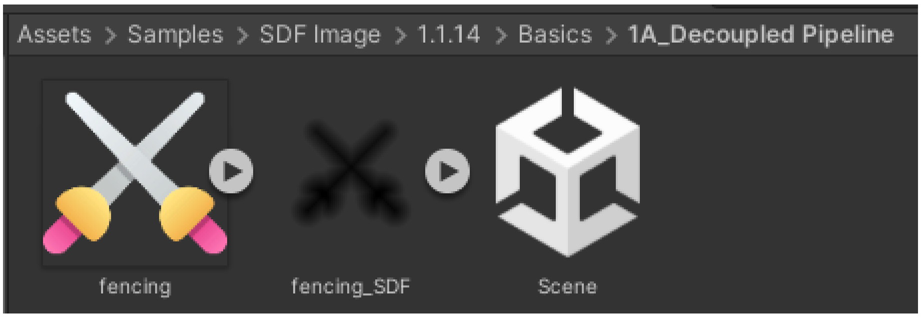
Generate Physics Shape: ☑

Sprite Editor

Generate SDF: □

Or in Texture Importer with button

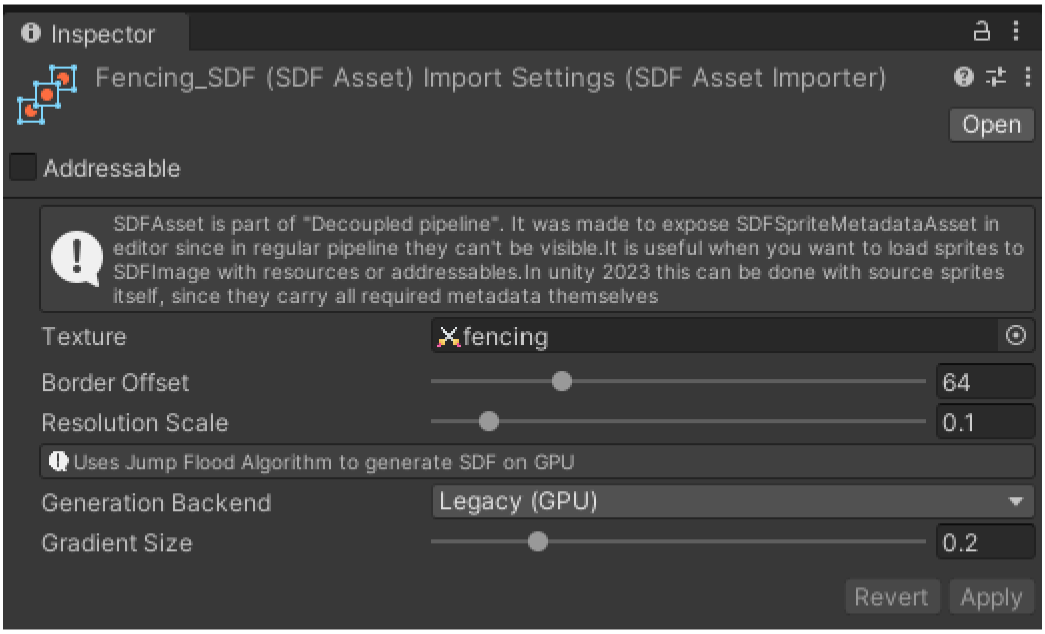
Sample files



Assets > Samples > SDF Image > 1.1.14 > Basics > 1A_Decoupled Pipeline

- **fencing.png** - source texture, it does not have Generate SDF checkbox
- **fencing_SDF.sdfasset** - `SDFAsset` that referencing fencing.png, all sdf contents is generated inside of it

SDFAsset



Fencing_SDF (SDF Asset) Import Settings (SDF Asset Importer)

□ Addressable

SDFAsset is part of "Decoupled pipeline". It was made to expose SDFSpriteMetadataAsset in editor since in regular pipeline they can't be visible.It is useful when you want to load sprites to SDFImage with resources or addressables.In unity 2023 this can be done with source sprites itself, since they carry all required metadata themselves

Texture: fencing

Border Offset: 64

Resolution Scale: 0.1

Uses Jump Flood Algorithm to generate SDF on GPU

Generation Backend: Legacy (GPU)

Gradient Size: 0.2

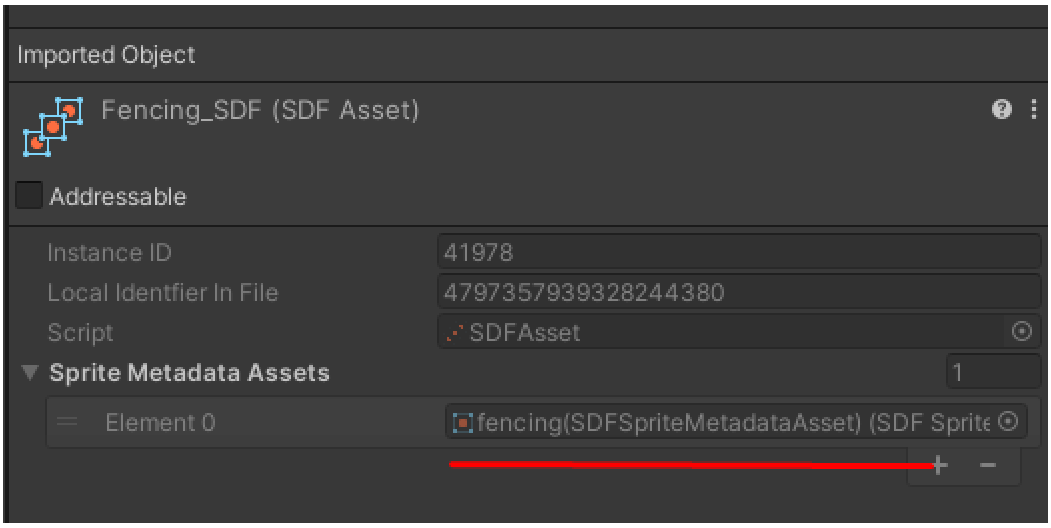
`SDFAsset` is product of SDFAssetImporter.cs, it works almost the same as sdf importer for regular pipeline, it generates same sub-assets but using different host/main object.

- Texture field - this is texture that sdf is generated from, texture should be in Sprite mode.
  Any re-import of the texture will re-import `SDFAsset` as well, so once you setup the reference you don't need to worry about manually updating `SDFAsset`.
- SDF Import settings - this is the same settings that regular texture has, its covered in SDF Import Settings

How to find which SDFAsset using Texture?

Since `SDFAsset` can be anywhere in the project and still reference texture its useful to know which asset using particular texture.

Switch Inspector to debug mode in order to see SDFAsset itself



Imported Object

Fencing_SDF (SDF Asset)

□ Addressable

Instance ID: 41978

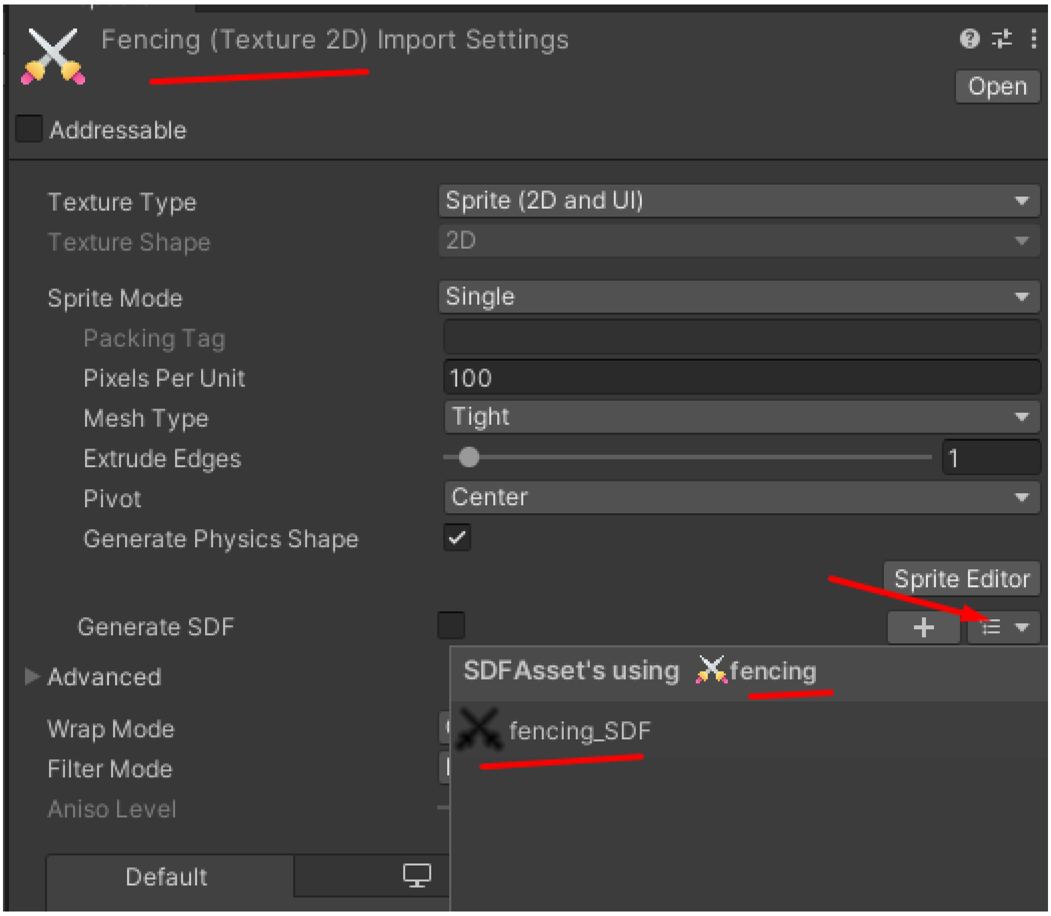
Local Identfier In File: 4797357939328244380

Script: SDFAsset

Sprite Metadata Assets: 1

Element 0: fencing(SDFSpriteMetadataAsset) (SDF Sprite

`SDFAsset` holds references to all `SDFSpriteMetadataAsset`'s, other then right now it has no other functionality.



Fencing (Texture 2D) Import Settings

□ Addressable

Texture Type: Sprite (2D and UI)

Texture Shape: 2D

Sprite Mode: Single

Packing Tag:

Pixels Per Unit: 100

Mesh Type: Tight

Extrude Edges: 1

Pivot: Center

Generate Physics Shape: ☑

Generate SDF: □

SDFAsset's using fencing

fencing_SDF

In TextureImporter there is dropdown button that shows all `SDFAsset`'s that is using this texture.
This is the similar menu to Prefab Family in unity 2022 and later.

In demo scene, currently only one `SDFAsset` is using this texture, but you can create multiple `SDFAsset`'s with different sdf import settings for single texture.

Why Decoupled Pipeline is useful?

- It decouples source textures and generated sdf, to store it anywhere in project. This also makes working with atlases easier (see more at 📸 Atlases)
- Exposes `SDFSpriteMetadataAsset`'s in project view, allowing them to be loaded as Resources or Addressables. I regular pipeline they are hidden, due to unity limitations.
- Intact import settings. In regular pipeline import settings is written to importer to AssetImporter.userData, this is not safe since other assets can override this settings. `SDFAsset`'s has separate importer therefore all its data is safely stored in them.

Want to know more?

Read detailed info of how decoupled pipeline working with scripting examples in [TODO] Decoupled Pipeline Scripting