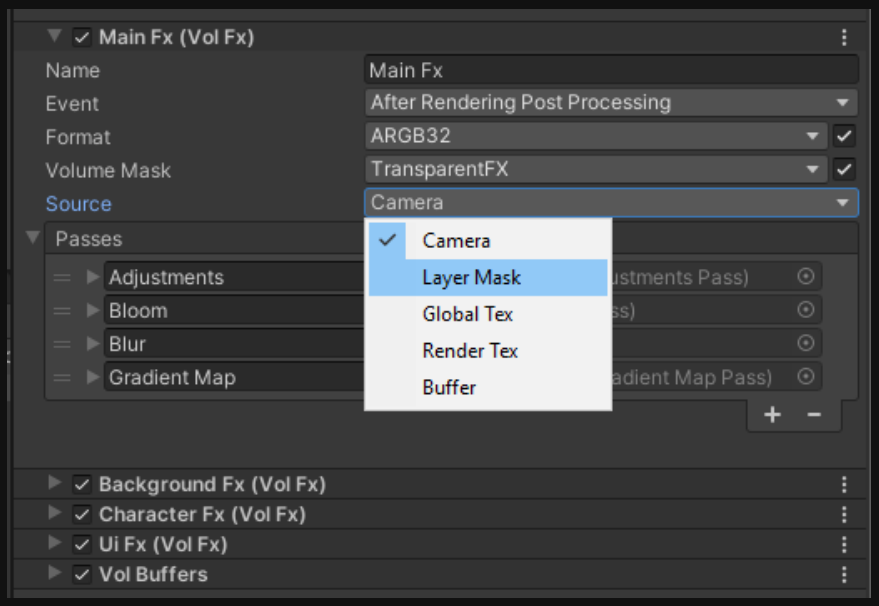
VolFx

VFX Toolkit
Quick Guide

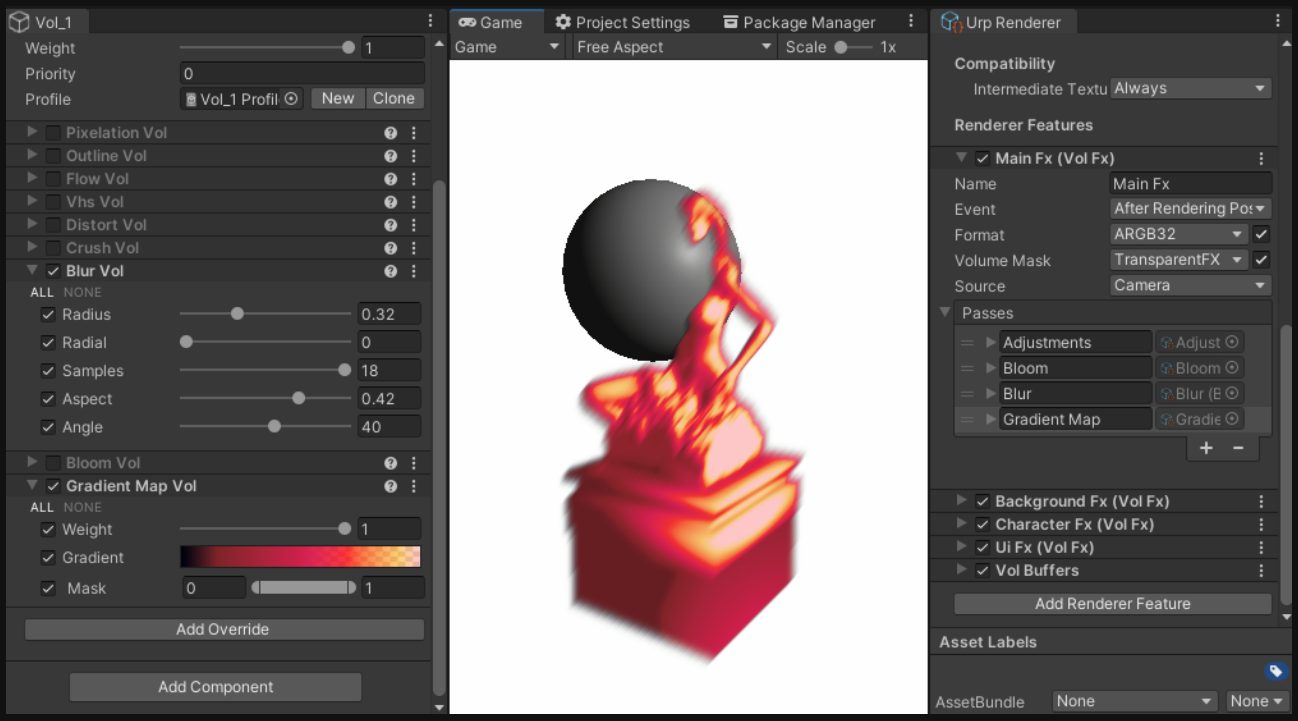
VolFx is customizable multy post-processing vis buffer system
that allows to build a custom scene processing for visual effects creation

It consists of modules that can process different sources, by LayerMask, GlobalTexture or Camera content



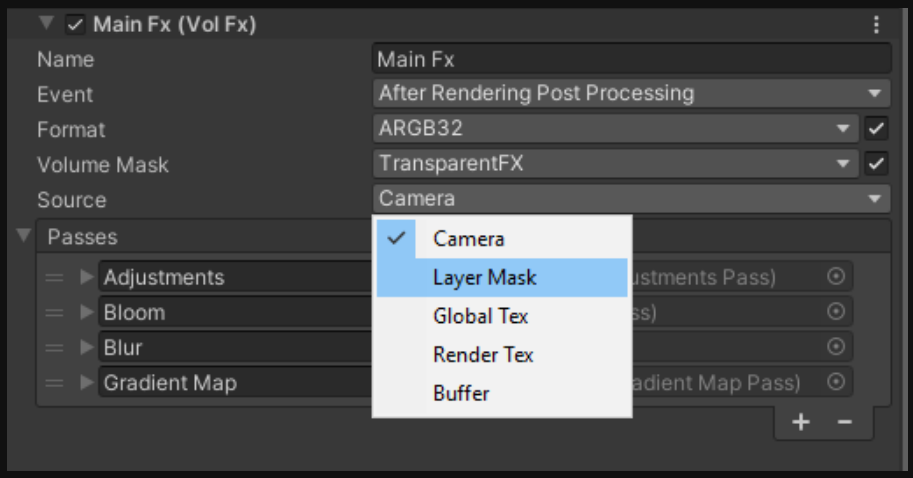
It can be used to control the scene and the display of groups of objects on it or to process textures for effects(like light maps, pattern animations, height etc)

To start using it you need to add VolFx RenderFeature to UrpRenderer and specify on which source will be processed.

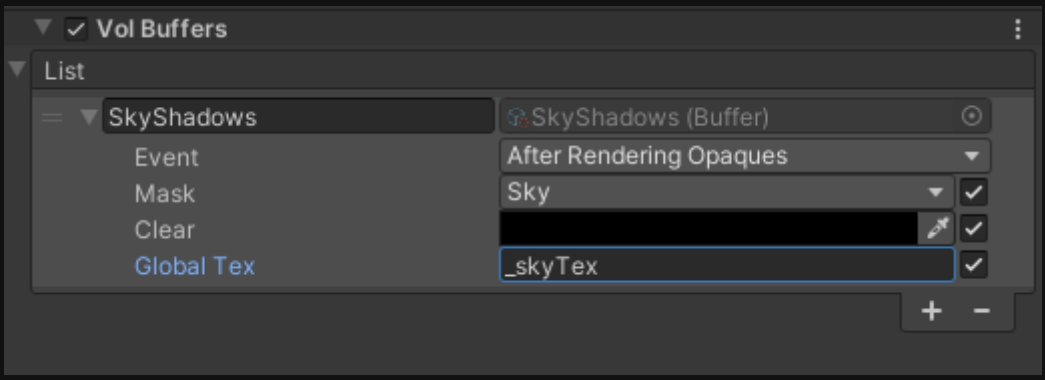


It can be a group of objects rendered by LayerMask, global texture or camera content.

Then add RenderPasses that will be applied to the source, in the order in which they are arranged in the queue and use it via Volume

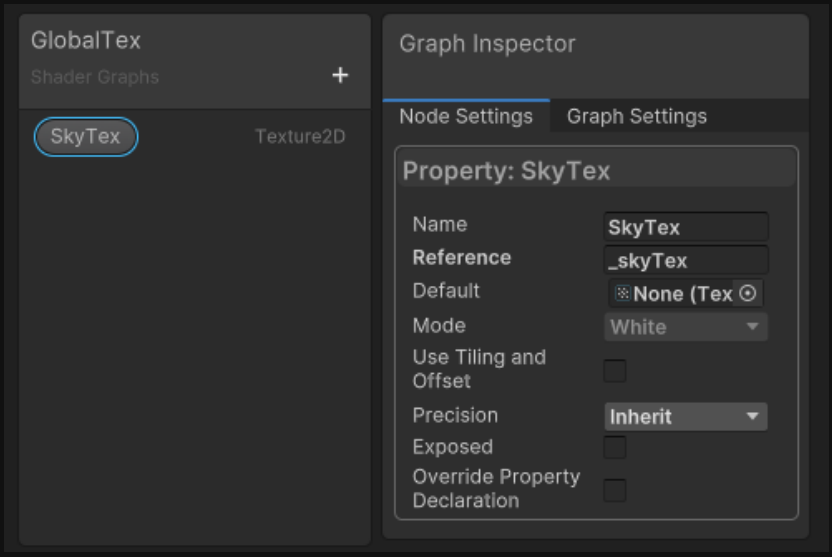


For some effects it may be useful to draw objects into a buffer in order to later apply its texture through a shader. (It can be light, fog of war or just a mask for some effect)

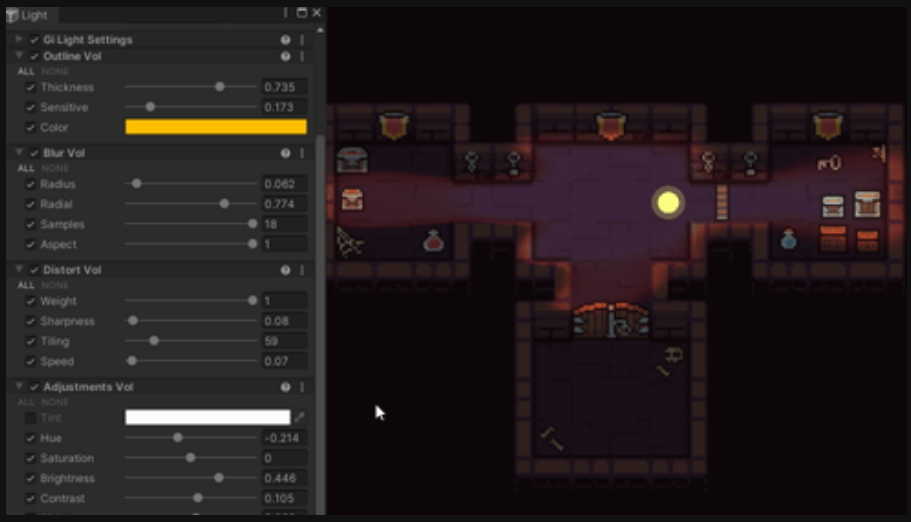


* Objects are collected by LayerMask and rendered into a separate texture

Texture can be processed with realtime and used throug a shader



* VolumeMask must be specified in order to select what settings to use for processing, example of post processing Light Texture



VolFx also allows you to create CustomPasses
* Example-template of a simple GrayscalePass can be found in ProjectSamples

To create a CustomPass, it must be inherited from VolFx.Pass and then it will appear in the list to be added.

```
[ShaderName("Hidden/VolFx/Grayscale")] // shader name for pass material
public class GrayscalePass : VolFx.Pass
{
    // =======================================================================
    public override bool Validate(Material mat)
    {
        // use stack from feature settings, feature use custom VolumeStack with its own
LayerMask
         var settings = Stack.GetComponent<GrayscaleVol>();

         // return false if we don't want to execute pass, standart check
         if (settings.IsActive() == false)
              return false;

         // setup material before drawing
         mat.SetFloat("_Weight", settings.m_Weight.value);
         return true;
     }
}
```

By default material is created automatically using path and **S**haderNameAttribute and is updated every time before processing is called.
But you can also overiider low-level to access additional functionality.

```
// called to perform rendering
public virtual void Invoke(CommandBuffer cmd, RTHandle source, RTHandle dest,
                           ScriptableRenderContext context,
                           ref RenderingData renderingData)
{
    Utils.Blit(cmd, source, dest, _material, 0, Invert);
}
```

In this way you can expand the engine and create dynamic effects controlled via VolumeProfile and scenes that have their own processing pipelines.