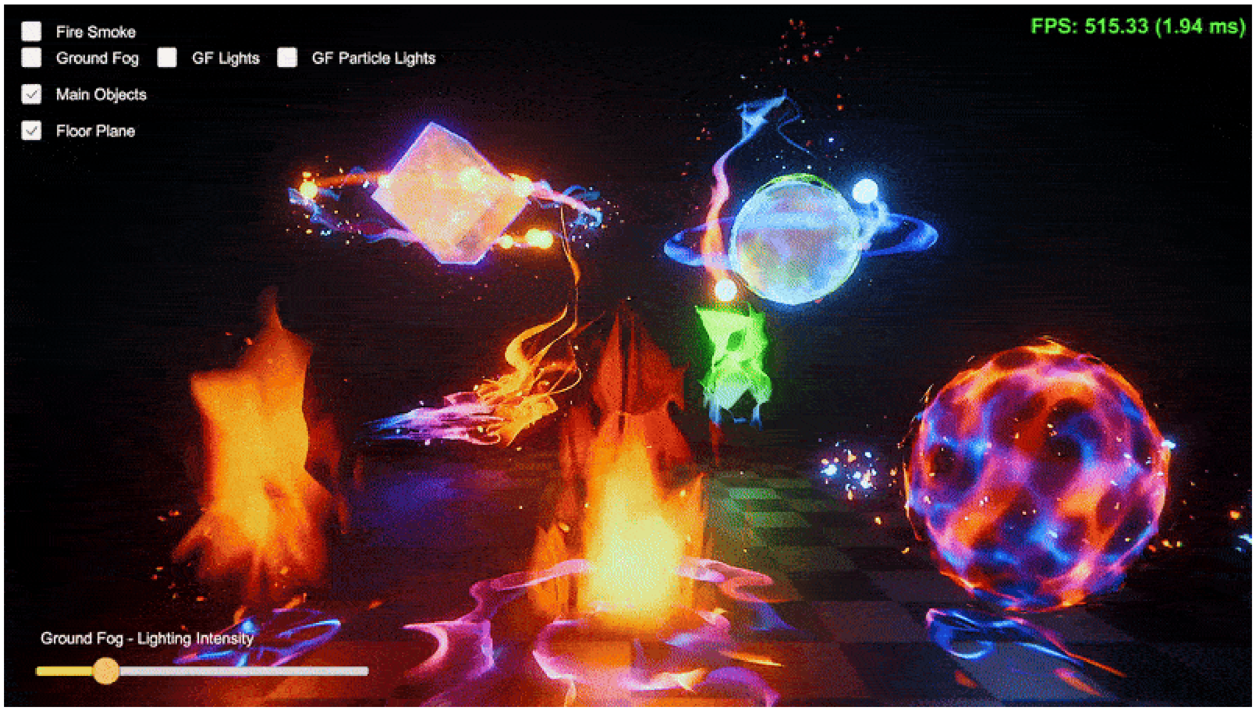
🔥

Solaris

Solaris is an advanced all-in-one shader for generating fully procedural fire, energy, and smoke-like effects without the use of any textures or complex mesh geometry. It works not only with static meshes, but also with Unity's particle system and trail renderer components, while supporting advanced custom transparent lighting + translucency. There are over 100 parameters available for adjustment in realtime, able to be tweaked and toggled for balancing performance against features.

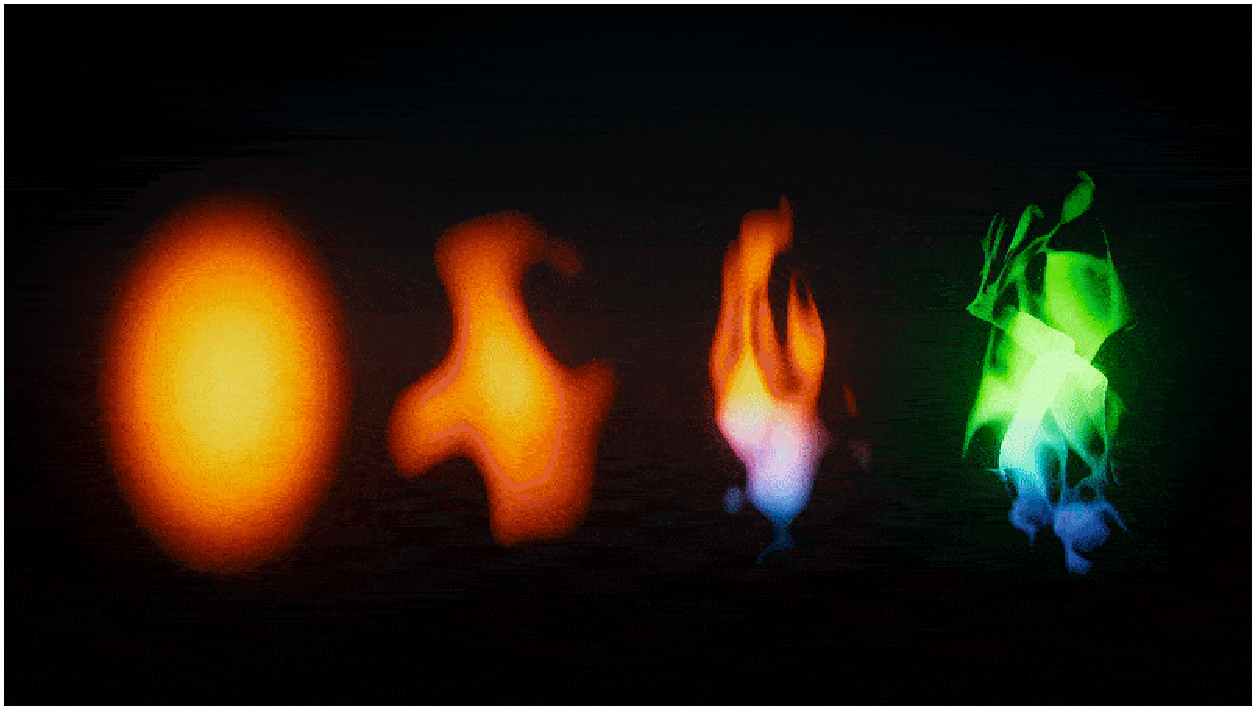


Main demo scene (included).

Despite it’s complexity, the basis of Solaris is simple: **masked fractal noise**.

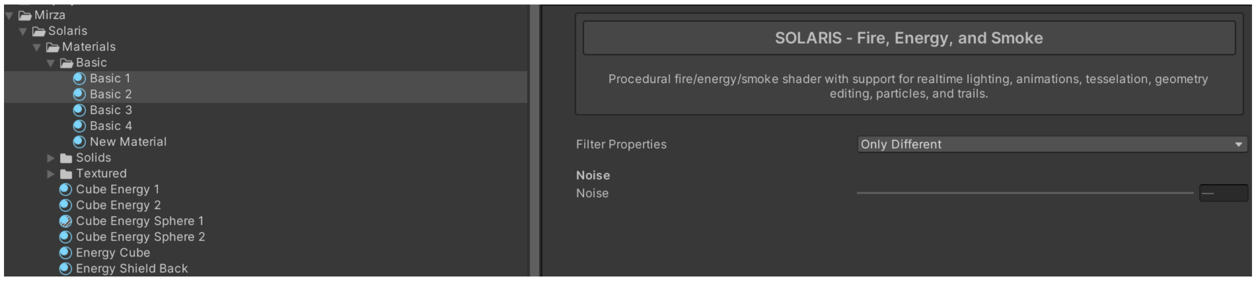
- Noise masked by an elliptical radial gradient, which can then be coloured in-shader, with options for geometry tessellation + editing, and caustic vertex animation.

To get an idea of the basics, open the aptly named *Basic* scene under Mirza/Solaris/Scenes/Basic/.


Basic materials scene (included).

Here's you'll see a straightforward layout of materials on Unity's default quad mesh, with increasing complexity from left → right. The first is simply the mask, following by introducing noise, then colour and some basic vertex editing, followed by the final animated green 'warp flame' on the right.

Solaris features an advanced, custom material editor which allows you to preview the differences between materials when multiple materials are selected. This makes it easier to identify the differences. The only difference between the first and second basic material is *Noise*.



Procedural fire/energy/smoke shader with support for realtime lighting, animations, tesselation, geometry editing, particles, and trails.

Filter Properties: Only Different

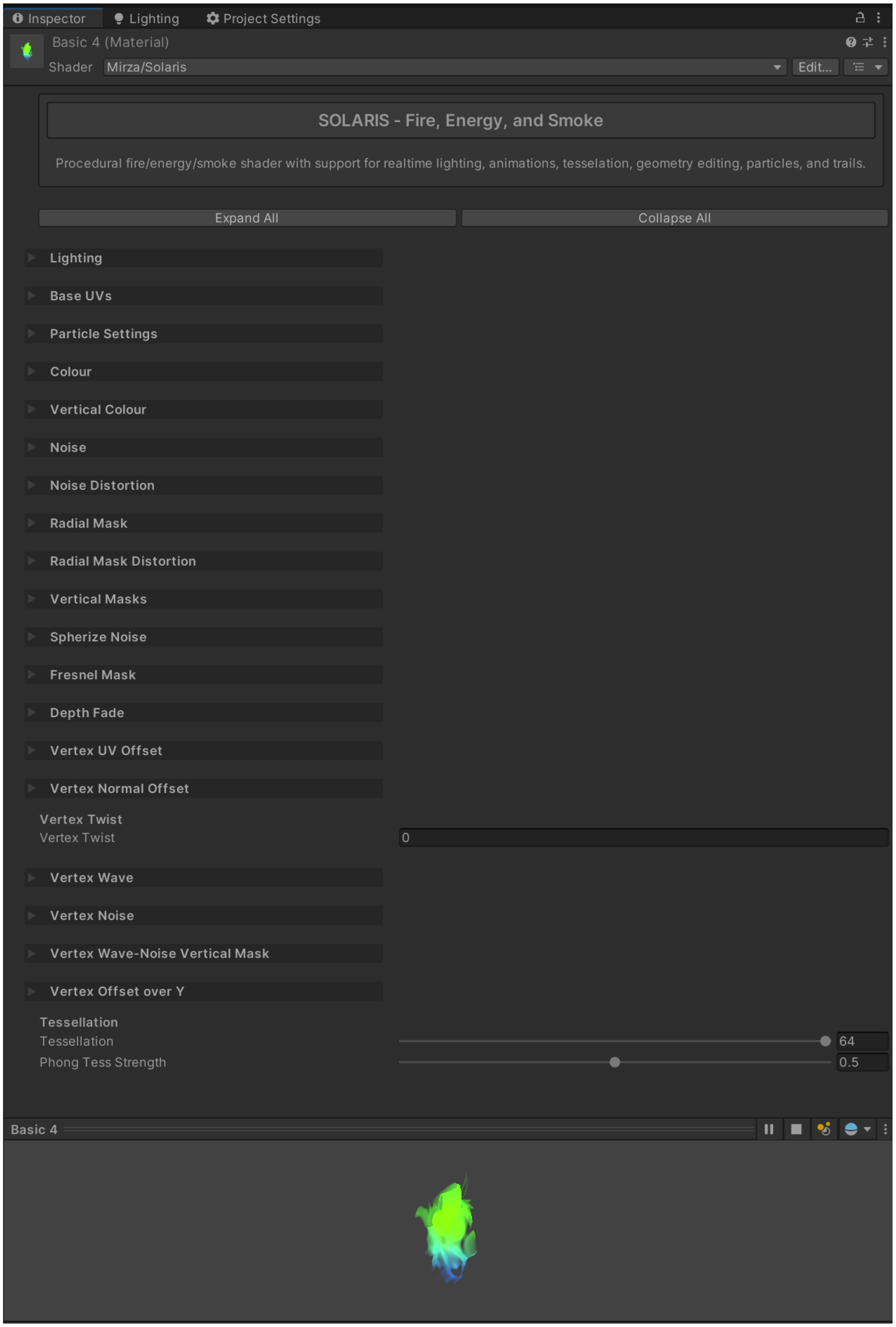
Noise

Noise: —

Multi-select materials and set **Filter Properties** to *Only Different* to preview the differences.

Material Editor

Solaris’ material editor organizes its properties into logical foldout groups, with a live preview.



Inspector
Lighting
Project Settings

Basic 4 (Material)

Shader Mirza/Solaris Edit...

SOLARIS - Fire, Energy, and Smoke

Procedural fire/energy/smoke shader with support for realtime lighting, animations, tesselation, geometry editing, particles, and trails.

Expand All
Collapse All

Lighting

Base UVs

Particle Settings

Colour

Vertical Colour

Noise

Noise Distortion

Radial Mask

Radial Mask Distortion

Vertical Masks

Spherize Noise

Fresnel Mask

Depth Fade

Vertex UV Offset

Vertex Normal Offset

Vertex Twist
Vertex Twist 0

Vertex Wave

Vertex Noise

Vertex Wave-Noise Vertical Mask

Vertex Offset over Y

Tessellation
Tessellation 64
Phong Tess Strength 0.5

Basic 4

Lighting

You can enable lighting and adjust the scale of various properties, such as the influence of the main directional light, and additional lights thereafter (point, spotlight...).

Base UVs

Swap XY to rotate the base UVs going from bottom to top → left to right.

Most UV sampling is done using the mesh surface (texture) coordinates, but you can choose to switch to world space or the transform's local object space UVs instead. The latter two are useful in cases where you want to sample noise volumetrically and/or when the actual texture coordinates aren't mapped to your liking on the mesh itself, etc... These are just example use-cases.

Particle Settings

Control *some* of the per-particle properties using by this shader/material. There are other properties that are applied on a per-particle basis, but they are grouped with other non-particle properties that they are directly related to for better organization, which you'll find in other foldouts.

💡 You only need these if you use Solaris materials for particles ***and*** want any per-particle customization. Otherwise, you can ignore **Custom Data** and **Custom Vertex Streams**.

Particle Randomization

- `0.0` = no randomization (no offsets applied per-particle).
- `1.0` = full randomization (large offsets applied per-particle).

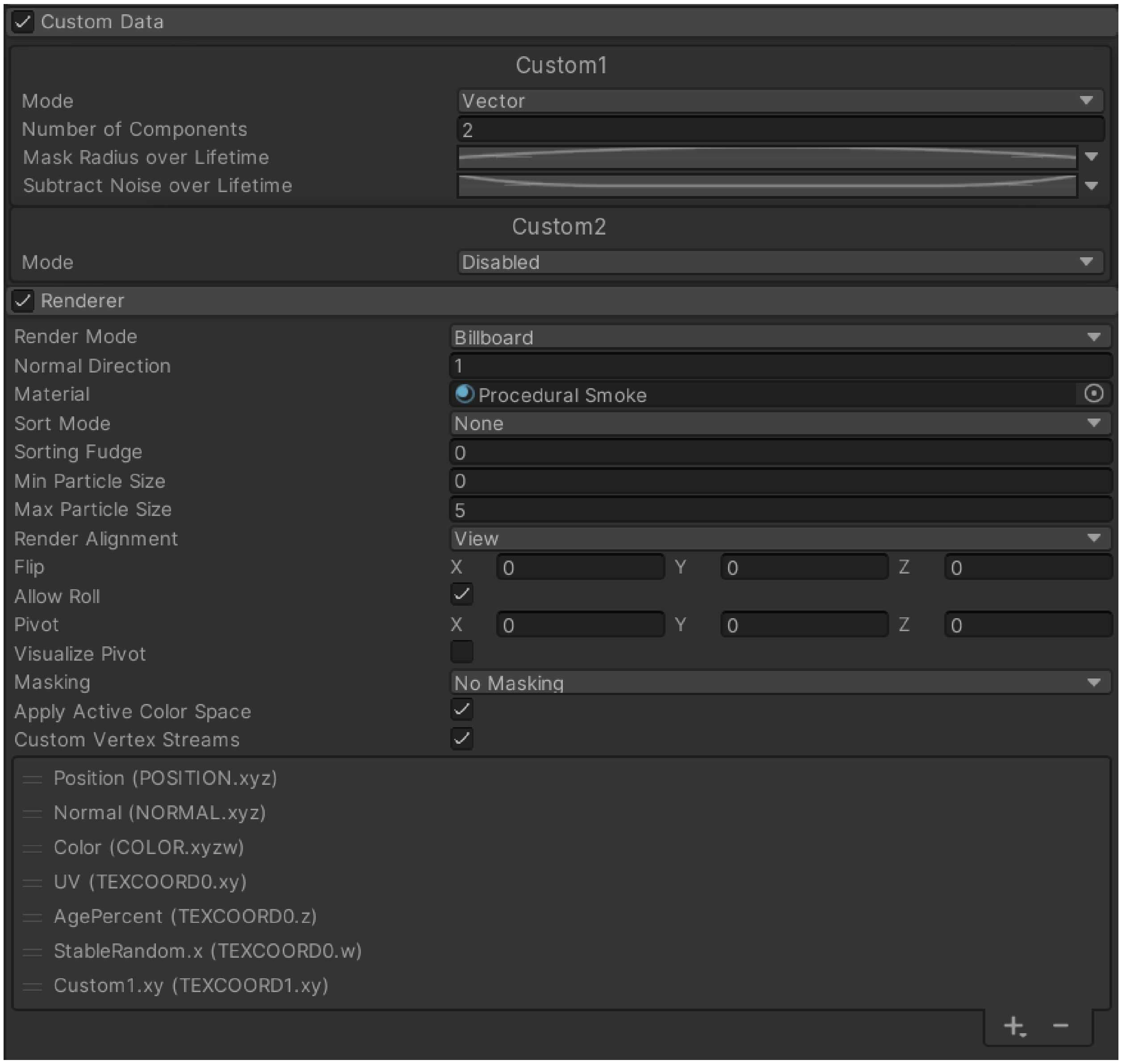
You can also scale the amount of noise subtraction (which is an alpha mask) over lifetime.

Custom Vertex Streams

Solaris can read custom vertex streams from Unity's particle system to apply per-particle randomization and animation/masking. **Order is important:**

- `AgePercent (TEXCOORD0.z)`
    - Required for noise animations.
- `StableRandom.x (TEXCOORD0.w)`
    - Required for randomization.
- `Custom1.xy (TEXCOORD1.xy)`
    - Enable the Custom Data module, where the X and Y of **CustomData1** are...
        - `X = Mask Radius over Lifetime`
            - The radial mask radius scale over lifetime.
        - `Y = Subtract Noise over Lifetime`
            - Noise erosion, which is itself an alpha mask (making it similar to alpha erosion).

💡 You can omit the vertex streams you don't need (including all of them), but they ***must*** be in order, so you can only remove streams from *bottom to top*. **Example:** You can choose to send only `Custom1.x` rather than both `.xy`, but can't remove `StableRandom.x` or `AgePercent` before that, as it would shift the target TEXCOORD element(s).



✓ Custom Data	
Custom1	
Mode	Vector
Number of Components	2
Mask Radius over Lifetime	
Subtract Noise over Lifetime	
Custom2	
Mode	Disabled
✓ Renderer	
Render Mode	Billboard
Normal Direction	1
Material	Procedural Smoke
Sort Mode	None
Sorting Fudge	0
Min Particle Size	0
Max Particle Size	5
Render Alignment	View
Flip	X 0 Y 0 Z 0
Allow Roll	✓
Pivot	X 0 Y 0 Z 0
Visualize Pivot	
Masking	No Masking
Apply Active Color Space	✓
Custom Vertex Streams	✓

```
Position (POSITION.xyz)
Normal (NORMAL.xyz)
Color (COLOR.xyzw)
UV (TEXCOORD0.xy)
AgePercent (TEXCOORD0.z)
StableRandom.x (TEXCOORD0.w)
Custom1.xy (TEXCOORD1.xy)
```

Colour

The main colour properties are here. A to B is a remap of the underlying grayscale output, which can be shaped using a power (exponential) curve. The alpha of the A/B properties is actually a multiplier for the RGB. You can apply additional adjustments to the hue and saturation, as well as scale the overall intensity and set the total alpha separately using the slider.

Vertical Colour

Use the toggle to enable this property block, which then applies a vertical gradient (based on the vertical mesh UV) mixing the base colour with the colours in this foldout/section, top to bottom. A/B colours in this foldout are the grayscale remap for the top, and otherwise function the same as the base colours. You can shape the vertical mask using the power and remap properties.

💡 **Useful trick:** When *remap min* is greater than *remap max*, the mask is **inverted**.

Noise

The core procedural 4D fractal caustic noise properties are here.

Basic controls include uniform scaling, XYZ tiling (W does nothing), XYZW animation (W = looping rotation), XYZW offset, and octaves. You can further enable and apply ‘dilation’ (using the gradient of the noise to expand itself, which creates a caustic or web-like effect).

Use the *Particle Animation* property to apply an offset over particle lifetime.

You can shape and remap the noise, apply a parallax offset (which makes it appear further ‘into’ the mesh), enable and twist it along the local XZ plane (for 3D models, not flat quads), and adjust/remap the flow of the length-wise UV (useful for changing the apparent speed of flow across the surface).

💡 Noise is used both for RGB and alpha. The grayscale of the noise is remapped into RGB colours, as well as serving as an overall mask for the alpha.

Noise Distortion

Another layer of noise that distorts the core noise.

Radial Mask

A radial gradient mask applied to the noise, which also has controls for mixing in the scaling of the radius by particle lifetime, if you’re using the material with a particle.

Radial Mask Distortion

Noise that distorts the radial mask.

Vertical Masks

Two additional vertical (length-wise) masks that can be enabled and applied with separate controls. They are multiplied in by default, but you can choose to apply them subtractive (simulating erosion).

Spherize Noise

Applies a bulging effect to the noise UVs.

- Useful for 2D geometry.

Fresnel Mask

Useful for applying a mask around the edges of **3D geometry**.

- Only applies to 3D geometry (not applicable to flat/2D meshes).

Depth Fade

Depth fade simulates a ‘soft particle’ effect, fading away (lowering alpha) where the mesh intersects with geometry. This is multiplicative by default, but you can toggle subtractive behaviour.

Camera depth fading blends the alpha away depending on proximity to the camera. This is especially useful to prevent clipping effects, for example, when moving through fog as the mesh will fade as you get closer, and reveal itself when you move farther away from it.

Vertex UV Offset

Applies a vertex offset depending on the vertical UVs of the mesh. You can use this to pinch or expand the geometry at the top and bottom of the mesh (based on UV texture coordinates).

- Useful for 2D geometry.

Vertex Normal Offset

Similar to **Vertex UV Offset**, except this applies offsets by the *normals* of the mesh.

- Useful for 3D geometry (has uses for 2D geometry: see **Vertex Twist**).

Vertex Twist

A non-foldout misc. property that twists the XZ vertices along the Y axis of the mesh.

- Useful for 3D geometry.
- Useful for 2D geometry that has been offset using **Vertex Normal Offset**.

Vertex Wave

Applies a sine wave to the vertices along the length of the mesh.

Vertex Noise

Applies a layer of noise distortion to the vertices of the mesh.

Vertex Wave-Noise Vertical Mask

A vertical mask applied to both **Vertex Wave** and **Vertex Noise**.

Vertex Offset over Y

Vertex offsets applied length-wise to the mesh. Useful for different types of mesh bending.

Two of these are applied using a power-curve, while one applies offsets based on a ‘circular’ (sine-based) curve. The former are useful for simulating gravity, momentum, and ‘jiggle physics’, whereas the latter is useful for rotation-based mesh bending.

The reason for multiple offsets is convenience and organization. You may want to apply some constant gravity to one of the offsets, using another for position-based momentum or jiggle physics, and the circular-based offset for rotation-based momentum. Examples are provided in the demos.

Tessellation

Mesh vertex sub-division.

While optimally you’ll want to use a custom mesh, this property is useful for (re-)using base geometry (such as Unity’s default quad) and being able to immediately take advantage of effects which would otherwise require a high-resolution (such as vertex noise, or twisting). For 3D geometry (non-quads, flat planes…) use *Phong Tess Strength* to adjust the curvature of subdivisions.