- Show Lasers -

Asset Version **1.0 – 2019-02-12**

Thank you or downloading **Show Lasers**! We hope you find this asset useful and wish you every success in creating your own laser shows and other laser-based effects. Below, you will find an overview of how to use Show Lasers. If you still have questions after reading this document or want to report a bug or request a feature, please let us know.

Show Lasers is developed by Sourcenity GmbH. Contact information and support:

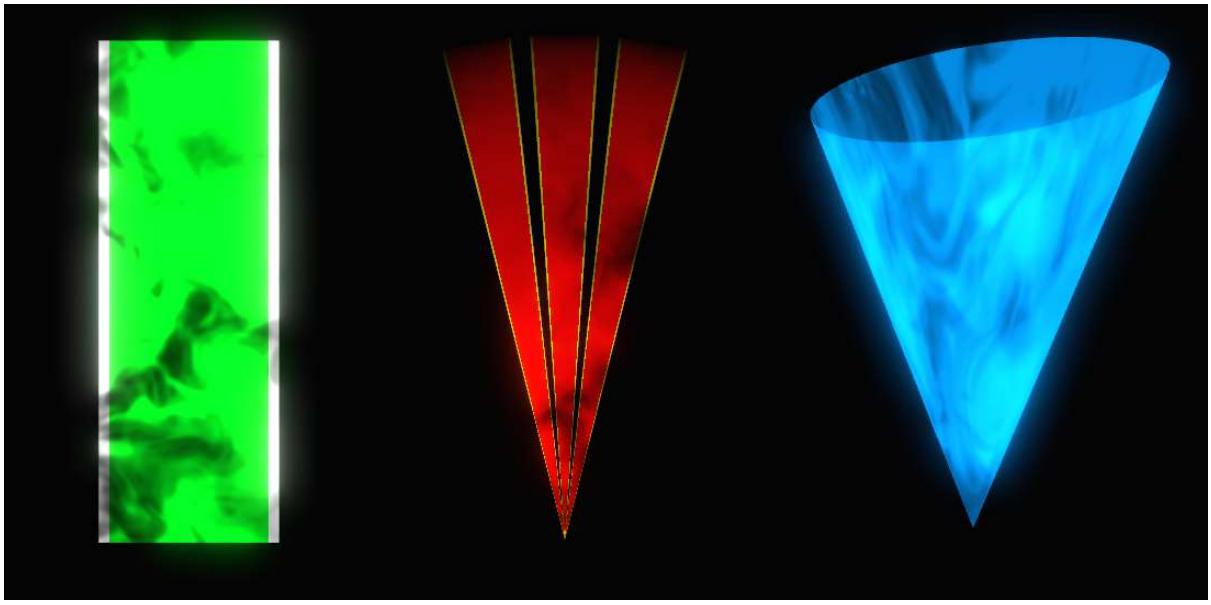
- Web: https://www.sourcenity.com/assets/showlasers/
- E-mail: contact@sourcenity.com

Show Lasers

About Show Lasers

Show Lasers is a script+shader for displaying "virtual laser beams" modelled after Laser Show eams on screen. Each beam originates at a certain origin (the pivot point of the beam), may be partially transparent, and give the appearance of cutting through clouds or smoke. Various settings allow changing the width, length, and radius of the beam, as well as rims or borders of different colors on both sides. The visual appearance of each beam is configurable to show different colors, different simulated smoke, types of transparency, and so on.

There are three basic types of beams – rectangular, circular (fan-like), and conic (cylindrical), as shown in the three screenshots below.



Which of these basic types you use depends on your use case. You can have a look at the three demo scenes to see examples on how to use each type. Each of the demo scenes also comes with animations to make it easier to understand the capabilities of each laser type.

To create a new effect, simply drop the `ShowLaserEffect` script that on a new empty game object. The script will install its own `MeshRenderer` and `MeshFilter` and the accompanying ShowLaserEffect shader. All configuration is done through the ShowLaserEffect script parameters. For more information about usage, see the sections **Basic Usage** and **Configuration** below.

To animate `ShowLaserEffect` objects, you can use either the Unity animation system (i.e. create animations for the object), or change the `ShowLaserEffect` public fields via scripts. To orchestrate the animations of multiple `ShowLaserEffect` objects, we recommend using Unity's timeline functionality. For more information, see the section on **Animation** below.

Show Lasers comes with a plain visualization as well as two types of simulated clouds or smoke. If you are not satisfied with either, you may provide your own texture or your own shader. For more information, see the **Custom Textures & Materials** section below.

Show Lasers does not require, but works well with the **Bloom** postprocessing effect. Although you can use any Bloom implementation you like, we provide you with a basic Bloom

implementation which can be dropped on the Camera of the scene as-is. For more information on these settings, see the **Post Processing** section below.

Supported Platforms

Show Lasers does not have any specific platform requirements. It has been tested on the PC platform in non-VR mode as well as in VR mode, including Single Pass Stereo, as well as on Android (plain) and Android (Oculus Go). Other platforms *should* work as well, although this has not been tested and we cannot give any guarantees (but would love your feedback!).

Both Linear and Gamma modes work fine, as well as Forward and Deferred rendering.

Performance

Show Lasers performance should be adequate for most use cases. Running the demo laser show on the following hardware configuration produces the following frame rates. All runs are in forward mode, 8xMSAA, "Ultra settings" in Unity, and with the Bloom post processing effect attached.

- **PC/Non-VR** on a "Standard PC" (i5-4590, GTX970), non-VR (2D), Full HD (1920x1080)
  ⇨ Averages **1200 fps** without vSync (about **0.8ms/frame**)
- **PC/VR** on the "Recommended VR Minimum Level PC" (i5-4590, GTX970), VR using OpenVR with the **HTC Vive** (2160x1200) in Single Pass Stereo mode
  ⇨ Frame rate stays at a **solid 90 fps** (capped). SteamVR performance graph shows a GPU time of about **2ms/frame**.
- **Android/non-VR** on a "Samsung Galaxy S7", non-VR (2D), Full HD (1080x1920)
  ⇨ Frame rate stays at a **solid 60 fps** (capped).
- **Android/VR (Oculus Go)**, VR mode, native resolution (2560 x 1440)
  ⇨ Frame rate stays at a **solid 60 fps** (capped).

Installation & Update

To install Show Lasers, simply download it from the asset store. If you have a .unitypackage file available, you can also choose Assets > Import Package > Custom Package in the main menu.

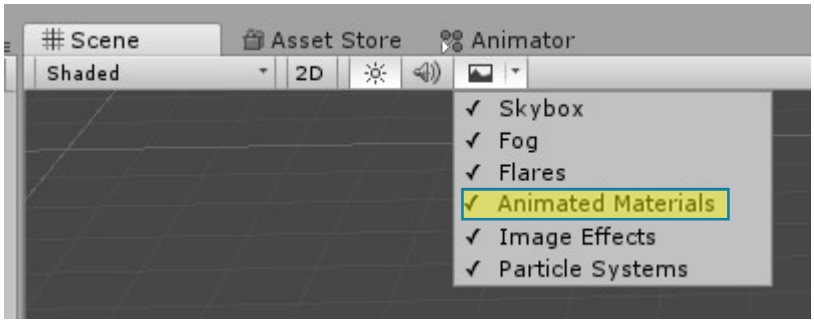
For updates, simply install the newer package.

You can move the Show Lasers folder wherever you would like.

Scene Settings

There are no required settings to be able to use Show Lasers. However, the effects will benefit from **HDR colors** that can be enabled on the main camera. Furthermore, the lasers also benefit from post processing effects such as Bloom. This is discussed in its own section below.

Changing any parameters of the `ShowLaserEffect` script in the editor will be **immediately** reflected in the generated mesh and material in the editor. However, to be able to actually **see** the changes (including the animations), you must enable the "Animated Materials" settings in the scene view, like this:



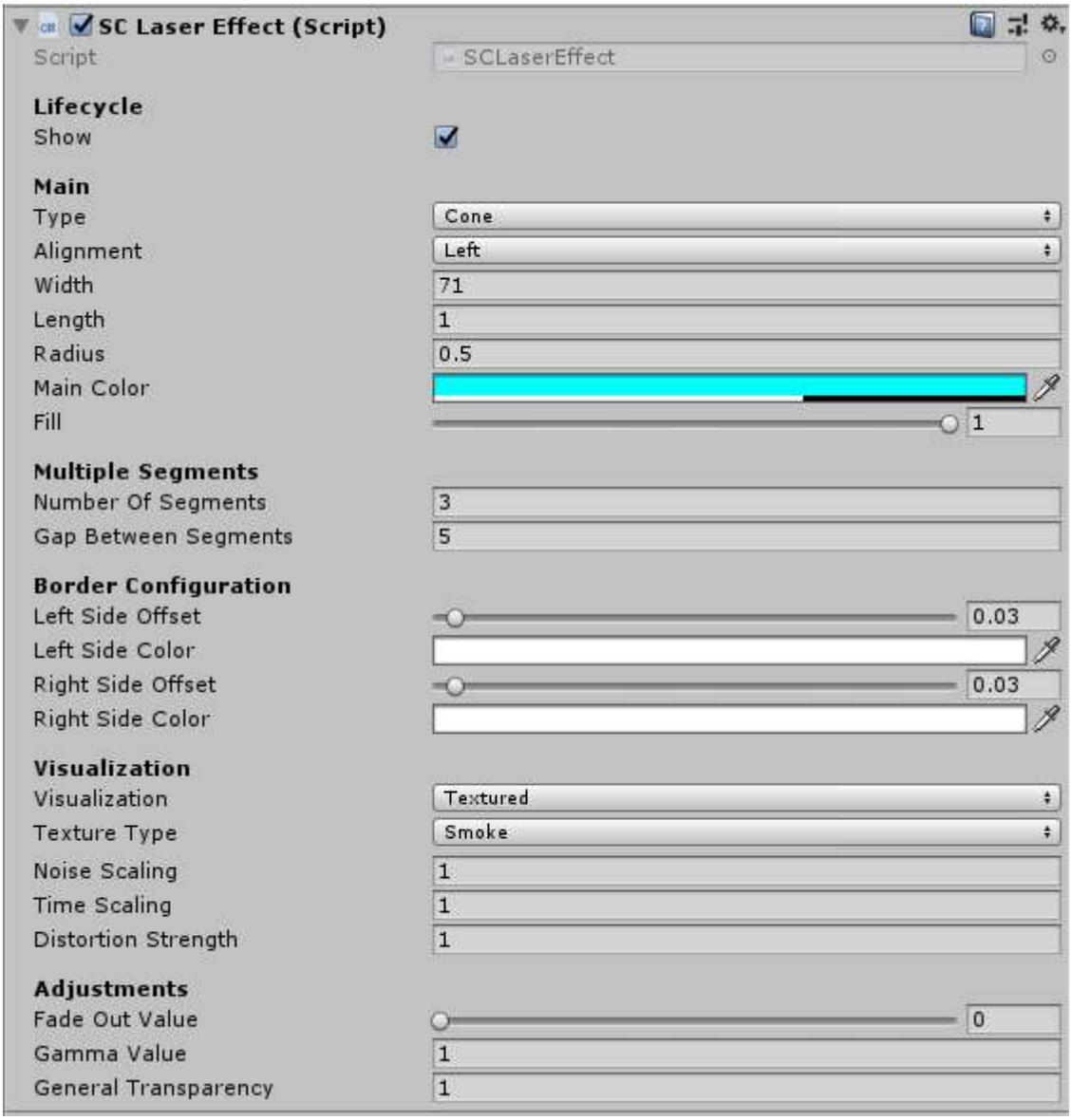
Basic Usage

There are two ways of creating a `ShowLaserEffect`:

1. Create an empty in your scene, and use the Add Component button to add the `ShowLaserEffect` script. The effects should be immediately visible in the scene view.

2. Drag one of the prefabs from the Prefabs folder onto your scene. Again, the effect should be visible immediately.

You can play around with the parameters to get a first feel for what you can do with each type of effect. Bear in mind that some parameters are mostly useful when animating and are thus not particularly interesting in static mode.



Configuration

In this section, we will explain each of the parameters of the `ShowLaserEffect` script. Refer to the adjacent screenshot for reference; note however that depending on your selections, some fields may not be visible at all times.

Lifecycle Part

In the lifecycle part, you will find exactly one Boolean value called **Show**.

Show. If show is false, the `ShowLaserEffect` script refrains from showing anything on the screen, but can still be configured in the background and re-enabled. **Show** is thus useful to start and stop displaying a particular laser, for example when animating.

Main Part

The main part covers the overall type and extent of the laser effect, as well as the color.

Type. There are three basic types of effects.

- ➔ **Plane** generates a rectangular area of size **Width** x **Length** in world units.
- ➔ **Fan** generates a segment of a circle with a certain (forward) **Length** in world units. The **Width** parameter, in this case, is in degrees. Using a width of 360 generates a complete circle.
- ➔ **Cone** generates a cone with an end radius of **Radius** (this field will only show up when Cone is selected). As before, **Length** specifies the (forward) length in world units, while Width specifies the angle in degrees. 360 generates a closed cone.

Alignment. Alignment can be set to **Left** or **Center**. In the first case, the generated laser beam extends towards the right from the origin. In the second case, it extends towards both sides from the origin. This can be particularly useful when animating the **Width** parameter. Note that in Left mode, a negative **Width** will allow you to build up the effect to the right.

Width. As mentioned above, **Width** configures the extent of the effect in the Left/Right direction (Unity X axis). In the case of a plane, **Width** is specified in world units; otherwise, in degrees. **Width** may be negative, in which case the effect grows into the other direction (note that this affects other parameters, see below).

Length. As mentioned above, **Length** configures the extent of the effect in the Forward direction (Unity Z axis). This parameter is always given in world units.

Radius. If the type is set to Cone, Radius specifies the radius of the end cone of the effect in world units.

Main Color. The main color determines the color of the effect. Note that it is permissible and often useful to change the transparency value of this color; note also that transparency is also affected by other visualization parameters (see below).

Fill. The fill value is a percentage (0 to 1, and 0 to -1) and denotes how much of the **Width** is actually filled. **Fill** is useful when animating: Slowly raising fill from 0 to 1, for example, results in a "built-up" animation. For a single effect, the same can be achieved with animating Width; however, when using multi-elements (see below), fill affects all elements at once. A negative value of fill "fills" the width from the other direction.

Multiple Segments

Each `ShowLaserEffect` can be split into multiple segments. This offers an easy way to create effects that are more interesting without having to create multiple game objects. In essence, the effect is "split" into multiple segments that each share the same values.

Note: For the **Fan** and **Cone** types, the overall size of the effect may not exceed 360° - thus, `NumberOfSegments * Width + (NumberOfSegments-1)* GapBetweenSegments` must be `<=360`.

Number of Segments. Indicates the number of segments used. The default is 1, i.e. only one segment. Use a number greater than 1 to split the effect into multiple parts. Note that the split may not be visible unless you also set a gap and/or rim colors.

Gap Between Segments. If there is more than one segment, the gap value denotes the space between the segments. This value is in units for the **Plane** type, and in degrees for the other types.

Border Configuration

In many laser shows, the left/right borders of the laser fan are using different colors and/or transparency values to "delimit" the effect more clearly. Using the border configuration values, you can achieve a similar effect. Note that these settings apply to all segments of the effect individually.

Left/Right Side Offset. The left and right offsets are a percentage of the **Width** and are used to specify how much of the border area is given a new color.

Left/Right Side Color. This field is visible if the left/right offset is not zero, and is used to give the border a different color from the main effect area.

Visualization

The visualization section allow you to configure the look & feel of the laser effect.

Visualization. Basic configuration of the visualization.

- ➔ **Plain**: The shader uses only the color(s) (main and border), along with their transparency. Useful for a simple look, or if you want to add noise using a post processing effect such as dirt.
- ➔ **Textured**: The shader uses a specific 4-channel "flowing" texture for visualization. Two such textures are built-in, and you can also supply your own. See the **Custom Textures & Materials** section below on how to create your own texture.
- ➔ **Custom Material:** In this case, the Show Lasers shader is not used at all. You have to supply your own material to take care of visualization; only the effect mesh is provided by Show Lasers.

Texture Type. This field is only visible when the **Textured** visualization option has been selected above. You can choose between two built-in textures (**Cloud** and **Smoke**) which can be further configured with the next fields, or you can supply your own texture. Note that this texture needs to conform to a certain layout, which is defined below in the **Custom Textures & Materials** section.

Noise Scaling. Noise scaling is the basic scaling effect in the XY-Plane for the texture used. The higher the value, the higher the repetition. Go below 1 for a zoom-like effect on the texture used. Note also that you can choose negative values, in which case the flow direction is reversed. A value of 0 removes the noise.

Time Scaling. Since the texture flow is animated, this value controls how fast the animation is going. Again, negative values reverse the animation. A value of 0 stops the animation.

Distortion Strength. The given texture is programmatically distorted using the given flow field by the shader. The strength of this effect can be controlled with this field; a higher value leads to a more "oily" effect. A value of 0 disables distortion.

Adjustments

The final section allows you adjust the effect globally.

Fade Out Value. In some cases, you might want to slowly fade the effect towards the forward (Z) axis. A fade out value of 0 disables fading, a value of 1 fades out nearly the entire effect. Use a value of near-to-0 to fade out the end of the effect.

Gamma Value. This value acts as a general multiplier for all color output. Essentially, this value controls the brightness of the lasers. Since the Bloom post processing effect allows you to define a lower threshold on brightness, you can adjust both Bloom and Gamma value in order to get a proper saturation effect.

General Transparency. This is the final transparency setting of ShowLaserEffect which supersedes the color transparencies as well as the fade out values. General transparency is useful in animations to fade in/ fade out the entire laser effect.

Animation

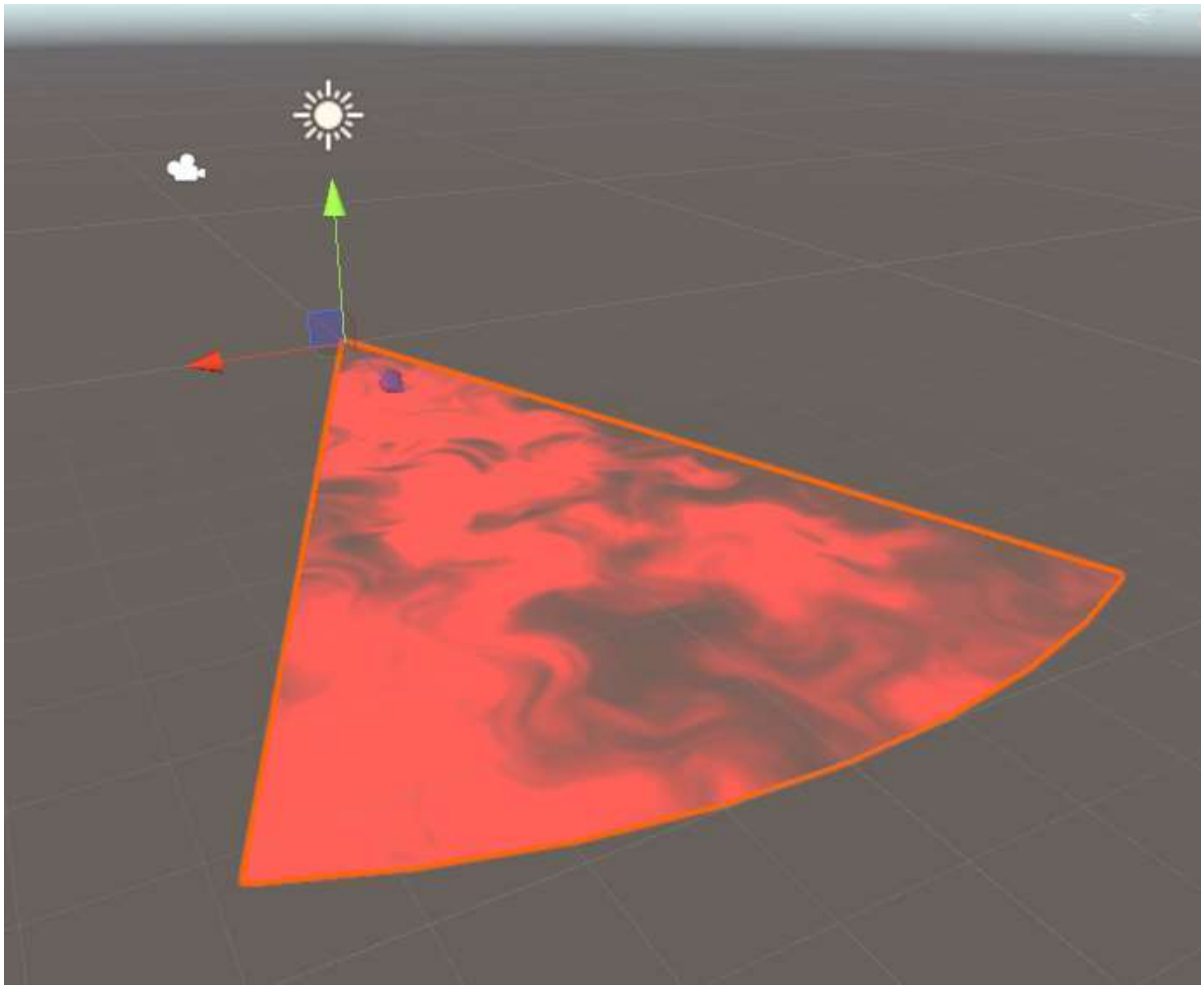
Show Lasers is made for animation. While the package itself does not contain any animations (except for the demo), you can easily create animations for each laser effect using any technique that is available in Unity. Two popular ones are using Meccanim animations (i.e. the default Unity animations) and combine them using the Timeline, or using your own scripts to control the `ShowLaserEffect` properties.

Note that the pivot point (origin) of each laser effect lies in the projection point, i.e. where the (imaginary) laser would emit the photons. This makes it easy for you to achieve animations that are similar to those generated by actual laser hardware.

Using Unity Animations and Timeline

Unity has powerful animation features that are fully documented in the Unity help pages, starting from here. We will not repeat these instructions here, but merely give a very simple example of how to create your first animation for an Show Laser effect.

First, create a laser effect by creating an empty game object at 0,0,0 with rotation 0,0,0, call it "My First Laser", and add the ShowLaserEffect script. Change type to **Fan**, alignment to **Center**, Width to **45**, set Visualization to **Textured/Smoke**, and change noise scaling to **0.2**. The result should like this:

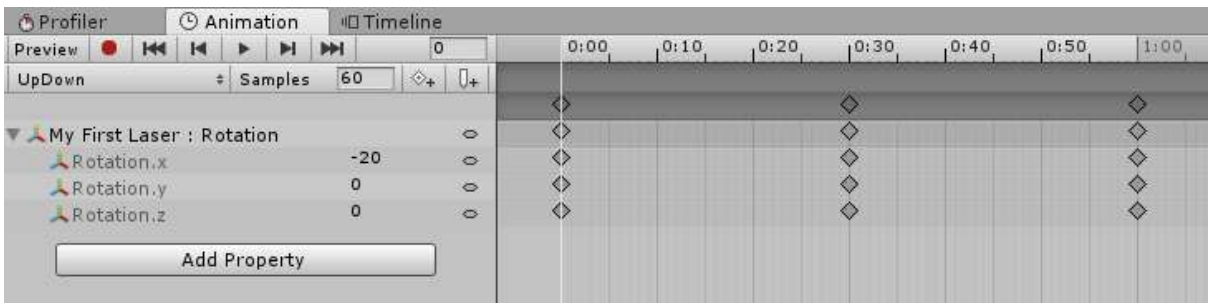


Now, we will animate this fan to go up and down. With your laser game object still selected, open the Animation view (Window > Animation > Animation). Within that window, click "**Create**" and select a file name (for example, "**UpDown.anim**").

To animate the fan up and down, we need to change the rotation. Click on the red record button on the left to start recording.

- First, we will set the start: With the time slider still at frame 0, change the X rotation of the game object to -20.
- Second, we will choose the end: Set the time slider to frame 30 (0:30 seconds), and change the X rotation of the game object to +20.
- Finally, the return path: Set the time slider to frame 60 (1:00 seconds), and change X rotation back to -20.

Stop recording by pressing the red record button again. The result should look like this.



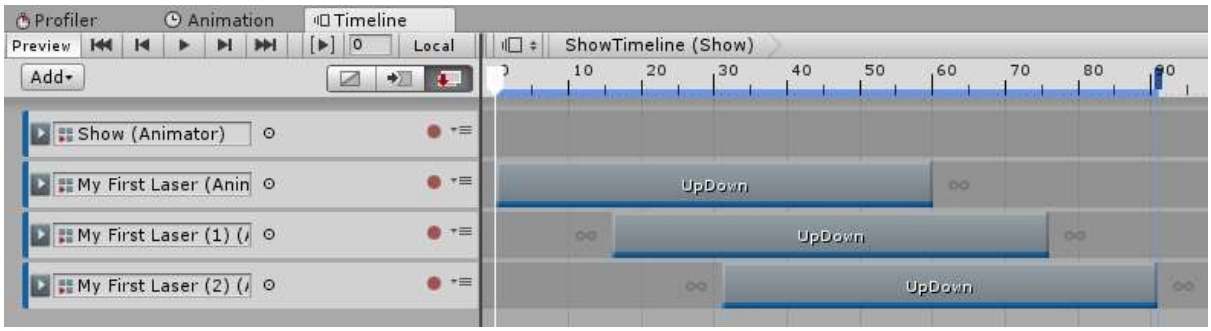
If you now press the Play button, you will see the fan animating up and down.

To animate multiple elements simultaneously or as part of a show, you should use Unity's timeline, which is documented here. Again, we will simply give a short introduction into its usage here.

First, create three copies of the fan game object you created; set their X position to -3, 0, and +3. Next, create another empty game object called "Show" which will house the timeline. With the "Show" object selected, open up the timeline view (Window > Sequencing > Timeline) and select **Create** to create a new timeline.

In the timeline, each track references a different game object. Drag&drop your three fans into the left-hand side of the timeline view. With each drop, you will be asked which kind of track you would like to add. Select "Animation track".

Now, you can add the "Up Down" animation we've created earlier to each of the three fans by simply dragging & dropping the animation file from the project window to the desired position. For example, to animate them in a slightly overlapping fashion, you can add the animations like this:



Clicking play shows all three fans being animated as expected.

In this example, we merely animated the rotation of the laser. You can of course also change any of the properties of the ShowLaserEffect script itself, such as the width, length, border, transparency, and so on.

The combination of animations and timeline gives you incredible creative and expressive power over what to animate and how. We recommend having a look at the demo laser show scene, which is set up in the same way, for some inspirations on how to design you own shows.

Using Scripts

Another way of animating `ShowLaserEffect` scripts is by code. The `ShowLaserEffect` script will automatically react to changes to its public properties, thus the only thing you need to do is change these public properties.

As an example, the following script, which needs to be placed on the same game object as the `ShowLaserEffect` script, fades an effect in and out in a ping-pong fashion.

```
using sourcenity;
using UnityEngine;

public class Fadeout : MonoBehaviour {

    public void Update()
    {
        float t = Mathf.PingPong(t:Time.time, length:1);
        GetComponent<SCLaserEffect>().generalTransparency = t;
    }
}
```

You can change any of the public parameters of the ShowLaserEffect script at will. There are only three things to note:

- For fan and cone settings, the overall width may not exceed 360 (or -360), otherwise, you will get visual artifacts. This means that `Width x NumberOfSegments + GapBetweenSegements x (NumberOfSegments -1)` must be `<=360` or, for negative width, `>=-360`. In the editor, those values are checked for you, but this is not the case if you change values using the API.
- Setting the field **Show** to false will remove the entire mesh from the mesh renderer for performance reasons. You can still change other parameters while show is false, but they will not be visible until show is set back to true, even in the editor.
- For some changes, `ShowLaserEffect` needs to re-create the mesh of the effect. This will happened automatically in the next frame and will incurs a (very) slight performance overhead. The mesh needs to be regenerated if changes occur in any of the fields in the **Main** and **Multiple Segments** parts of the script. For all others, only the material (shader settings) is changed.

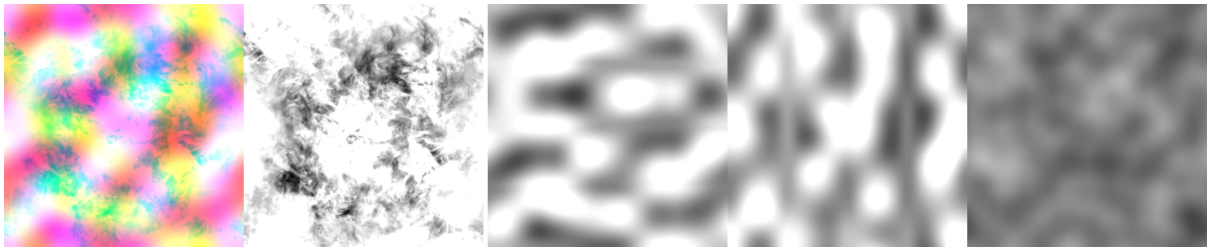
Custom Textures & Materials

Show Lasers comes with two types of "simulated smoke": **Clouds**, which are modelled after Perlin noise, and **Smoke**, which uses a vector-based smoke representation. The textures are setup in a way in which the plain 2D texture is distorted, or **warped**, to achieve a moving effect.

The visual appearance of Show Lasers can be changed in two ways. First, you can supply your own 4-channel texture. This is the recommended way as it still uses the Show Lasers shader, which means that you can still control the colors, borders, transparency, and so on through the Show Laser script.

Custom Texture

The two textures which are supplied with Show Lasers can be found in the Resources folder. These are TIFF format files with four channels, each of which contains a grayscale image which has a different function in the shader. The following picture shows the full RGBA view, followed by the Red, Green, Blue, and Alpha channels.

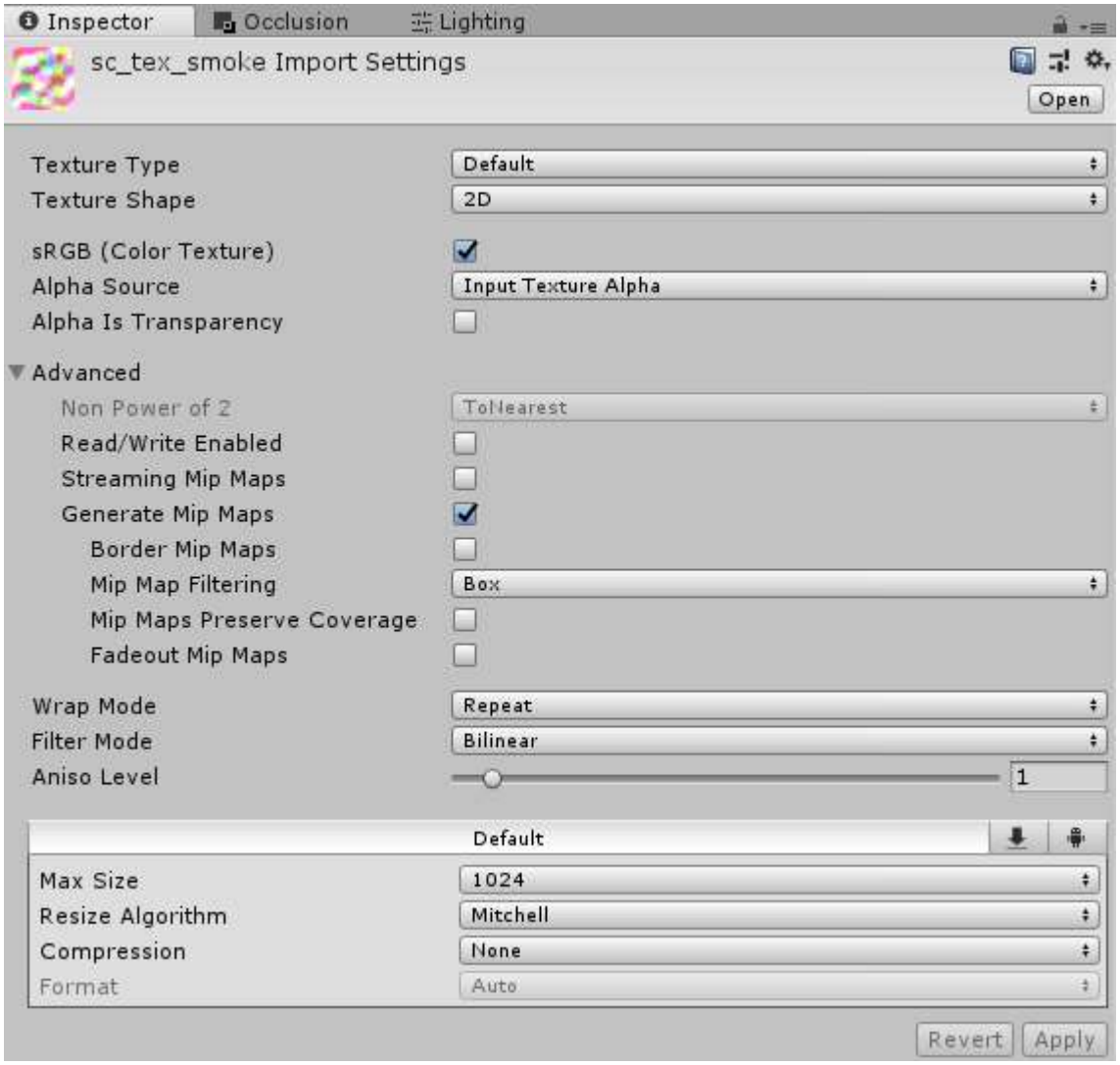


The shader works with these channels as follows.

- **Red Channel.** The red channel contains the actual image to be distorted, which is why, in this example, it looks like smoke. This is **most likely the channel you want to replace** with your own variant of "clouds" or "smoke". Note that the image should be **seamless**. The image is (obviously) grayscale; the tint color is supplied with the setting in ShowLaserEffect.
- **Green and Blue Channel.** These two channels, together, contain the flow texture, which indicates, for each point, the vector UV that indicates the distortion flow direction. You can replace both channels with your own implementation of a flow map.
- **Alpha Channel.** Finally, to add some more randomness, the alpha channel contains a noise texture that is applied to the final output. This adds another layer of randomness to the visualization.

We recommended starting with a new red channel if you want to experiment with custom textures.

Note that you need to be careful with the import settings of the texture files as well: Be sure to set the max size to the actual size (1024 in our case), and disable compression (None) as compression interferes with the distortion algorithm. Below are the import settings for the included textures.



Custom Material

The second option is to not use the Show Laser shader at all. In this case, however, the ShowLaserEffect will only provide the mesh of the effect without any visualization.

If you want to go that way, please have a look at the `ShowLaserShader.shader` file in the Resources folder to help you get started writing your own shader.

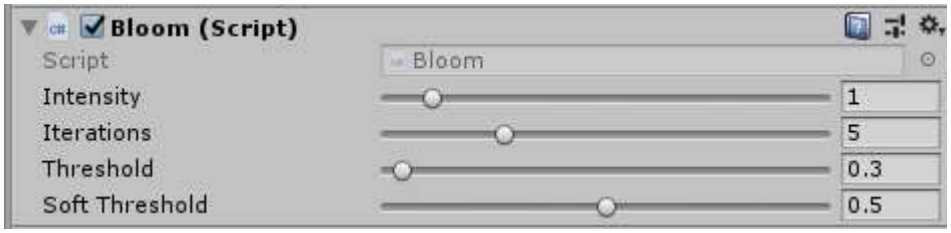
Post Processing

Show Lasers can be used as-is without any post processing options. However, many feel that adding post processing (that is, image screen effects) enhances the experience and thus we have made sure that it is possible to use Show Lasers with the typical post processing affects, the foremost of which is the **Bloom** effect.

The Bloom image effect essentially creates a "glow" around bright objects.

Using the included Bloom effect

Show Lasers comes with a Bloom effect that consists of a (post-processing) script and a shader. The script needs to be attached to the camera, and has the following settings:



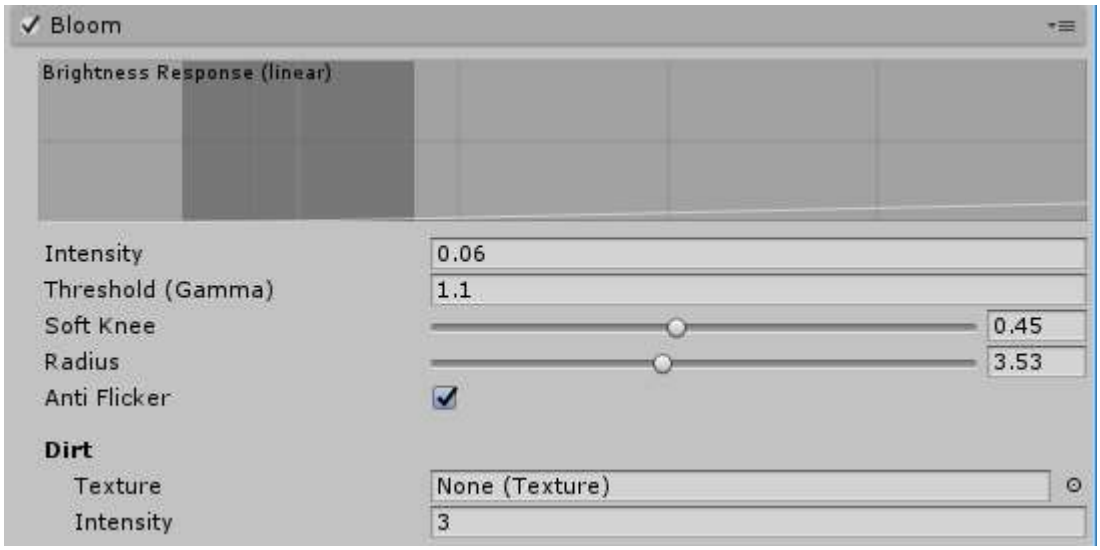
The most important parameter to control is **Threshold**, which indicates the cutoff gamma value from which on the effect will be applies. If you want to raise this threshold to avoid "normal" objects in the scene to be affected, be sure to raise the gamma value in `ShowLaserEffect` as well to achieve the same visual effect.

To raise the overall "explosive effect", you can increase intensity. To change the glow effect, try playing with the Iterations setting.

Using Unity's Post Processing Stack

A common choice for Unity developers is to use Unity's post processing stack which includes a Bloom effect. To install the Unity post processing stack, you need to first install it from the asset store.

Once installed, please follow the instructions in the Unity documentation. The Bloom effect is one of several which are included in the stack, and has the following settings:



The settings are similar to those of the Bloom effect that comes with Show Lasers. We recommend that you start playing with **Threshold** and **Intensity**.

Final Words

We again thank you for downloading and using **Show Lasers.** For creative help when creating your own laser shows, we recommend you check out some of the laser show recordings on YouTube for inspiration. Creating good-looking shows is not easy and will take some time, but we feel it is definitely worth it. All the best!

The Sourcenity team