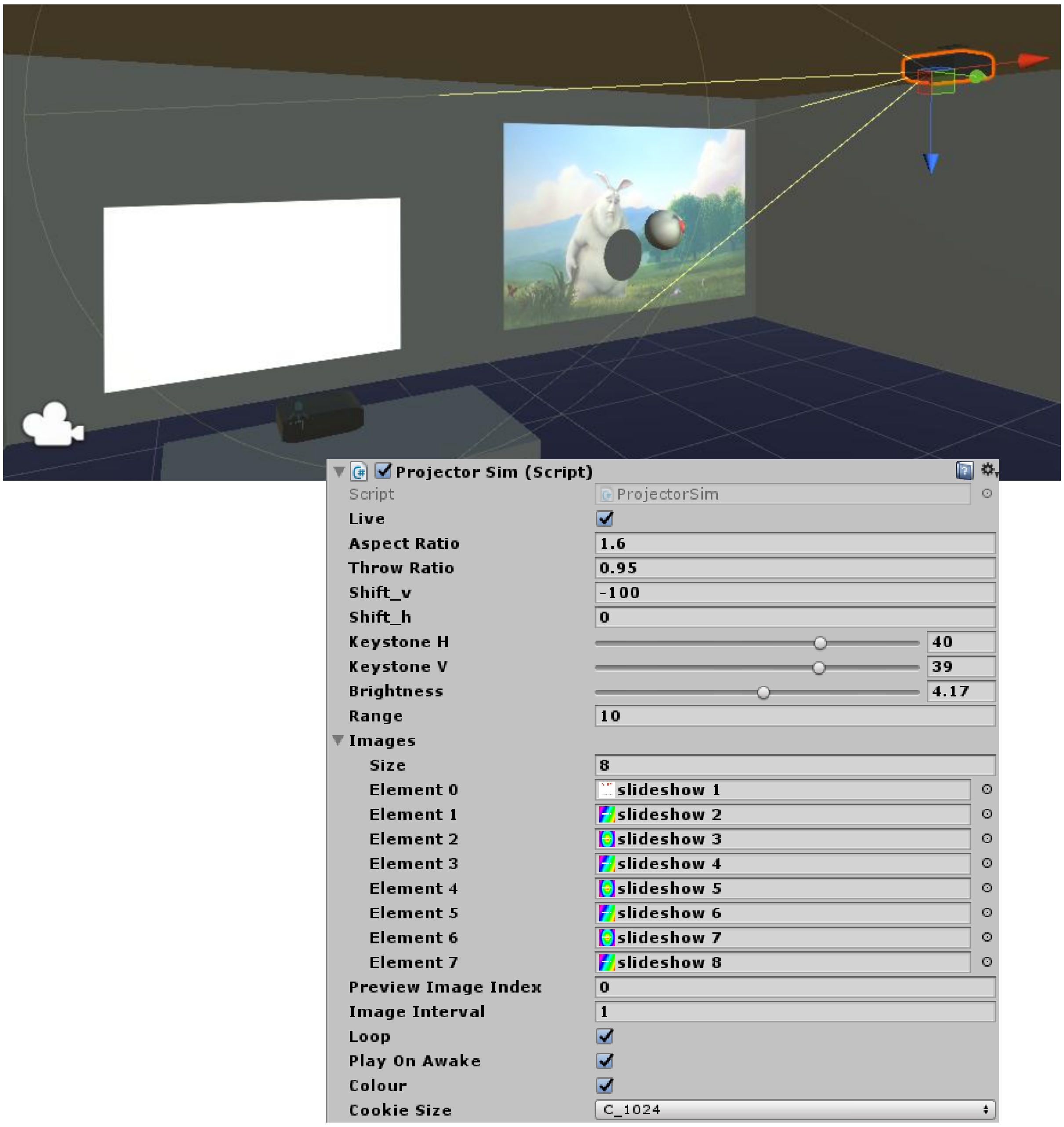
Projector Simulator

White Games



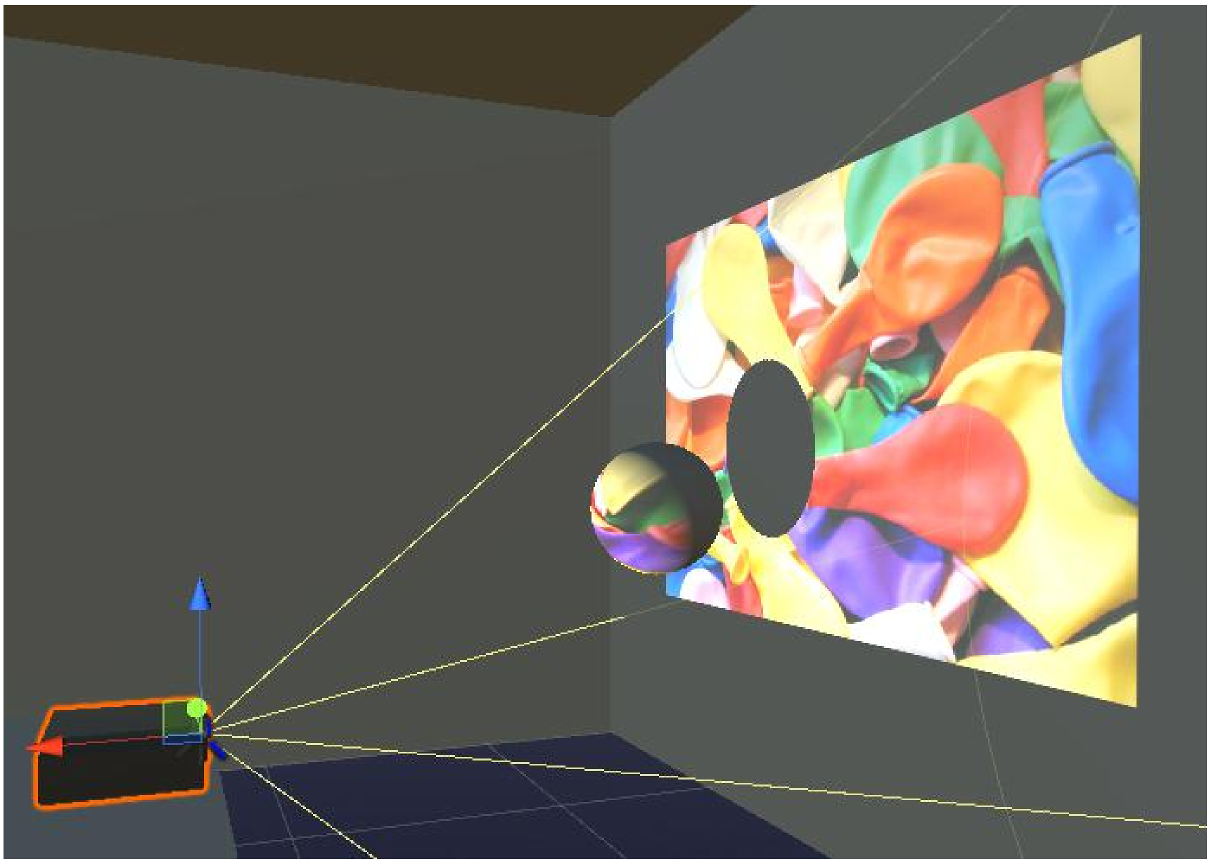
Projector Sim (Script)

Script	ProjectorSim
Live	✓
Aspect Ratio	1.6
Throw Ratio	0.95
Shift_v	-100
Shift_h	0
Keystone H	40
Keystone V	39
Brightness	4.17
Range	10
Images	
Size	8
Element 0	slideshow 1
Element 1	slideshow 2
Element 2	slideshow 3
Element 3	slideshow 4
Element 4	slideshow 5
Element 5	slideshow 6
Element 6	slideshow 7
Element 7	slideshow 8
Preview Image Index	0
Image Interval	1
Loop	✓
Play On Awake	✓
Colour	✓
Cookie Size	C_1024

Table of Contents

Overview

Projector Simulator is a Unity package which allows you to project any static image (or series of static images) in colour. It also simulates off-axis projection (lens shift) to create realistic shadowing effects and allowing projectors to be placed, for example, on a ceiling or table offset from the centre of the image.



Release Notes

V1.21

- Added “Range” slider, as previously a projector's range was fixed at 10, and you would have to manually edit each underlying Spotlight range in order to increase the reach of the light
- Increased default projector range to 20 instead of 10

V1.2

- Added keystone adjustment sliders
- Optimised generation of images after the first image in a projector (subsequent images now use the first image as a starting point)
    - On a colour 1.6 aspect projector, we saw between 5% and 66% improvement in the time it took to generate subsequent images, depending on projector resolution and amount of lens shift (more lens shift = more improvement, as there is more black space in the resulting cookies. Smaller resolutions also resulted in more improvement.)
    - Your results may vary depending on projector resolution, aspect ratio, lens shift amount, and processing speed.
- Fixed a bug where the slideshow would play when calling “SetSlideshowIndex”, even if the slideshow was paused
- Other small optimisations/code cleaning

V1.1

- Added support for multiple images per projector
    - Now possible to project slideshows and quasi-video
- Added C# projector control interface
- Fixed a bug which caused images to be projected at the incorrect size – images now make the most use out of the available pixel space

V1.01

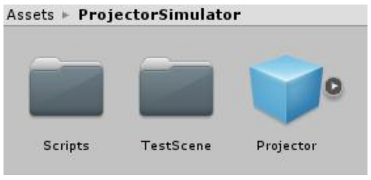
- Fixed a bug which caused all projectors to default to 256x256 resolution on level load

V1.0

- Initial release

How to use

After importing the package, you will see 3 items:

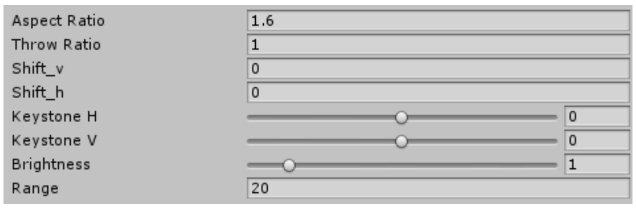


The **Scripts** folder contains the 2 main scripts for the asset to work – *ProjectorSim.cs*, and *CookieCreator.cs*.

The **TestScene** folder contains all the data used by the example scene. This is not needed for the asset to function.

Finally, there is the **Projector** prefab which can be placed in your scene.

1. To begin, drag and drop the "Projector" prefab into your scene
2. Position the projector so that the light source is in the desired location. Rotate the projector around the Up axis so that the forward direction is perpendicular to the display surface (if a square image is desired)
3. Use the **Aspect Ratio**, **Throw Ratio**, **Shift_v**, and **Shift_h** values to position the image in the desired location



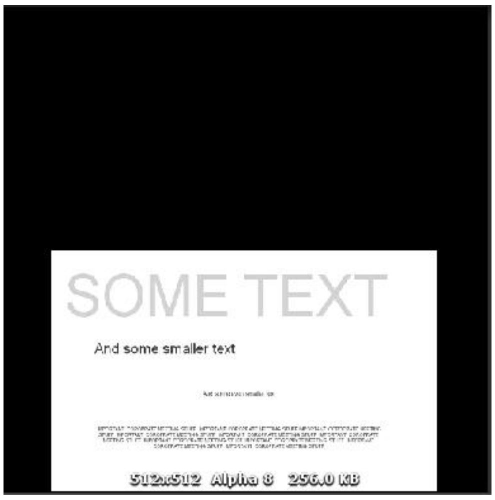
Aspect Ratio	1.6
Throw Ratio	1
Shift_v	0
Shift_h	0
Keystone H	0
Keystone V	0
Brightness	1
Range	20

Aspect Ratio is the aspect ratio of the projected image.

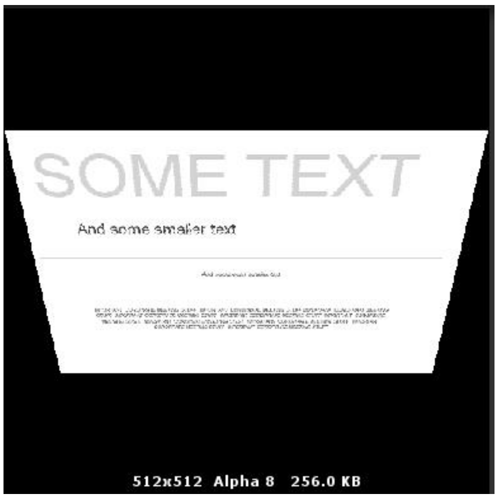
Throw Ratio is equal to the projector distance divided by the image width, and is a standard measurement used by real-world projection lenses. Smaller values result in larger images.

Shift_v and **Shift_h** correspond to the vertical and horizontal lens shift amount, respectively. Use vertical shift if you wish to achieve a square image with the light source above, below, or to the side of the image centre (e.g. a projector on a ceiling).

The **Keystone** sliders allow you to project a trapezoidal image. Usually used to make the image appear rectangular when your projector is not perpendicular to the screen surface. Note that keystoning can create unwanted artefacts in the projected image, as it is no longer an orthographic projection:



Lens shifted

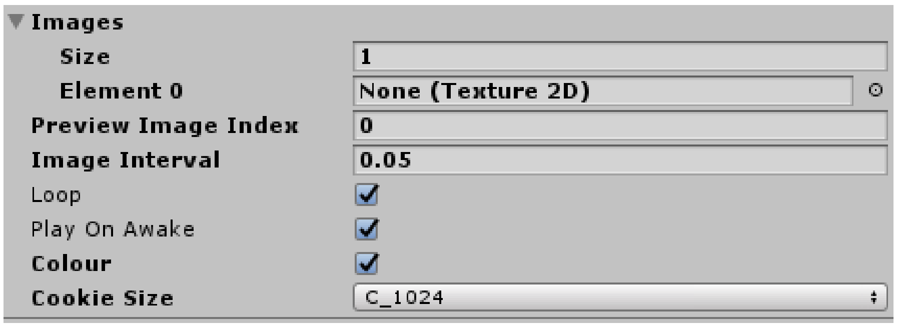


Keystoned

We will aim to mitigate these in a future update by interpolating the pixels samples from the projected image. When keystoning, you may have to adjust the projector's Aspect Ratio value, as keystoning squashes the image.

You can also use the **Brightness** and **Range** sliders to adjust the brightness and reach of the image. Depending on the image size, the image may appear too dull or too whited-out.

4. Once the image is in place, use the remaining controls to control what the projector will display:

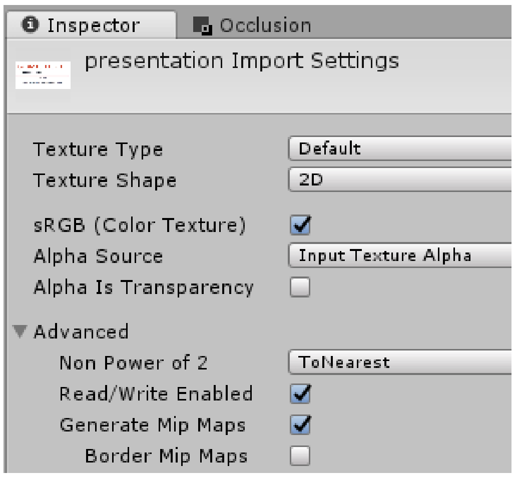


▼ Images	
Size	1
Element 0	None (Texture 2D)
Preview Image Index	0
Image Interval	0.05
Loop	✓
Play On Awake	✓
Colour	✓
Cookie Size	C_1024

The **Images** parameter is where you can drag and drop your image files. Several images can be added to create a slideshow or simple video. Be aware that each image takes a certain amount of time to process at the beginning of your level. If your projector contains hundreds of video frames, you can expect it to add several seconds to your level load time.

*Tip: You can use external tools such as **VirtualDub**, **VLC Media Player**, **Adobe Premiere**, or your preferred choice of software to convert a video's frames into a sequence of image files.*

The only prerequisite to dragging your images into the projector is that you have ticked **Read/Write Enabled** in the image's import settings:



presentation Import Settings

Texture Type	Default
Texture Shape	2D
sRGB (Color Texture)	✓
Alpha Source	Input Texture Alpha
Alpha Is Transparency	☐
▼ Advanced	
Non Power of 2	ToNearest
Read/Write Enabled	✓
Generate Mip Maps	✓
Border Mip Maps	☐

The **Preview Image Index** value is used to control which image is shown when the editor is in edit mode. When you have lots of video frames, you can scrub this value to find a suitable frame (the default preview frame may be solid black, for example, causing the projector to project no light).

Image Interval is the amount of time in seconds to show each image, if more than one has been added.

The **Loop** option allows the array of images to be cycled through continuously. Unchecking this option causes the projector to freeze on the last image when is it reached.

Play On Awake determines whether the slideshow will play as soon as the projector is turned on (usually at the start of the level). To start your level with the projector off altogether, simply disable the ProjectorSim component on the prefab, and re-enable it via a script when needed.

*Note: the public boolean variable **playOnAwake** also acts as an indicator for whether the slideshow is currently playing. Check this value when toggling between play/pause modes to know whether to call PlaySlideshow() or PauseSlideshow(). See the included **ProjectorControl** scene for an example.*

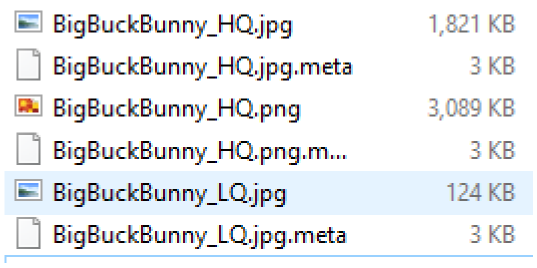
The **Colour** checkbox dictates whether the projected image is projected in full colour, or converted to greyscale. Greyscale projected images are about 3x quicker to generate than equivalent colour images.

The **Cookie Size** setting sets the size of the cookies used by the projector's light sources. This should remain at a low setting (e.g. C_256) while the image is being positioned, otherwise stuttering may occur during step 3. Once the image is in place, you can experiment with this property until a suitable image quality is achieved. Higher values will increase the time it takes to generate the projected image(s) at the start of the level.

Image Import Settings

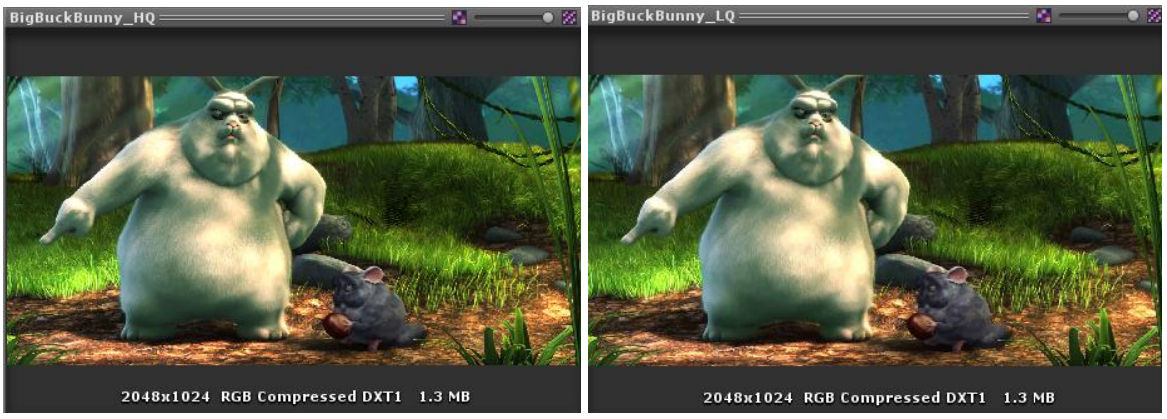
It is possible for a projector to contain hundreds (possibly thousands) of images to be used in a slideshow or quasi-video. These images will however increase your build size. The more images you add, the more benefit you will get out of properly compressing the images in question in Unity's import settings.

The format of the image in your assets folder has nearly no bearing over the size of the asset used by Unity. For example, here we have imported the same 1920x1080 image saved as a PNG (3089KB), high-quality JPG (1821KB), and low-quality JPG (124 KB).



BigBuckBunny_HQ.jpg	1,821 KB
BigBuckBunny_HQ.jpg.meta	3 KB
BigBuckBunny_HQ.png	3,089 KB
BigBuckBunny_HQ.png.m...	3 KB
BigBuckBunny_LQ.jpg	124 KB
BigBuckBunny_LQ.jpg.meta	3 KB

Although the asset image files are drastically different sizes, Unity processes all images through the same import settings, and the actual images used by the engine all end up more or less the same size in the Inspector:



BigBuckBunny_HQ

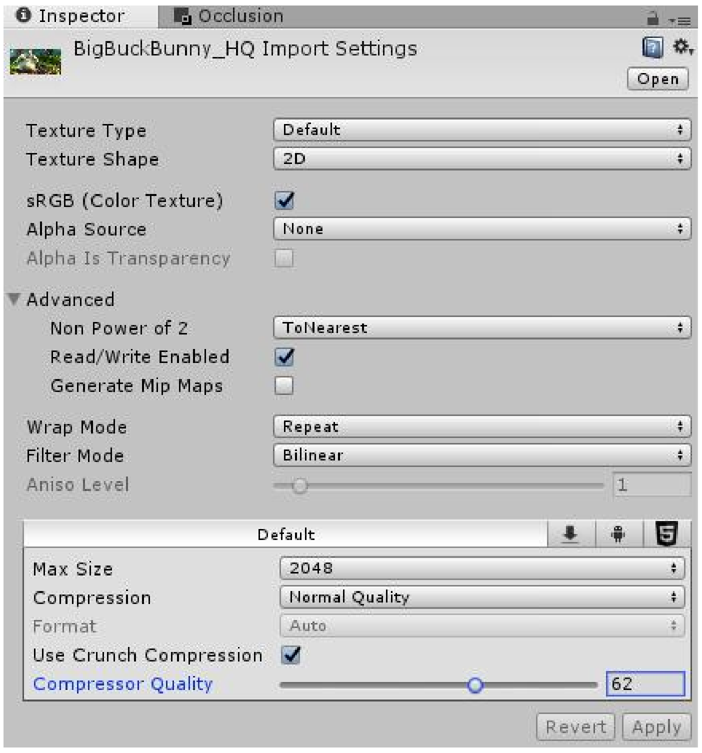
2048x1024 RGB Compressed DXT1 1.3 MB

BigBuckBunny_LQ

2048x1024 RGB Compressed DXT1 1.3 MB

If we had 100 of these 1.3MB images in our projector, our application size would grow by ~130MB.

We can however modify the images' import settings to reduce their size. You can select all of your images simultaneously in the Project window to change them all at the same time and ensure they are all using the same settings.


Inspector | Occlusion

BigBuckBunny_HQ Import Settings

Open

Texture Type: Default
Texture Shape: 2D
sRGB (Color Texture): ✓
Alpha Source: None
Alpha Is Transparency: ☐
Advanced
Non Power of 2: ToNearest
Read/Write Enabled: ✓
Generate Mip Maps: ☐
Wrap Mode: Repeat
Filter Mode: Bilinear
Aniso Level: 1

Default
Max Size: 2048
Compression: Normal Quality
Format: Auto
Use Crunch Compression: ✓
Compressor Quality: 62

Revert | Apply

First of all, ensure that **Read/Write Enabled** is ticked, as this is required for the projector to project the image.

If the images are not going to be used as textures anywhere in your scene, you can uncheck **Generate Mip Maps**. This reduced the Big Buck Bunny images from 1.3MB to 1.0MB.

You should also set **Alpha Source** to “None”.

Non Power of 2 may also be experimented with. Depending on your image's initial resolution and aspect ratio, you may or may not see a drop in Unity's reported image size. Try “None”, “ToNearest” and “ToSmaller”. Note that any value other than “None” may degrade the projected image quality due to the likely change in aspect ratio, but it should hardly be noticeable.

Max Size can of course potentially decrease your image's size, but may reduce the quality of the image too drastically. Feel free to experiment with this value.

The **Compression** setting can be used to find a balance between image quality and image size. A lower quality setting will result in a smaller size, but may degrade image quality too much. Experiment with this value.

Use Crunch Compression can drastically reduce your image's size without affecting the quality too noticeably. With our Big Buck Bunny image, the reported file size reduced from 1.0MB to:

	Compressor Quality = 100	Compressor Quality = 0
PNG	348.2KB	144.7KB
High-quality JPG	350.2KB	144.9KB
Low-quality JPG	286.6KB	129.5KB

Note that applying crunch compression can take several seconds per image. If you have hundreds of images you can therefore expect crunch compression to take several minutes. Try experimenting with a single image before applying the changes to the whole set.

Quasi-video

It is possible to extract frames from a video and play them back on a projector at a desired framerate. Do not expect to achieve flawless 4k 60fps video – although we can't stop you from trying. We recommend a frame rate between 15-30fps, depending on your source video's frame rate.

Although the Big Buck Bunny example above shows a 1920x1080 image, we recommend a lower resolution when working with quasi-video (e.g. 1280x720 or 720x480) as this will drastically reduce your build size compared to using full HD images.

You should pick a framerate which is a factor of the video's source framerate. For example, if I have a 60fps video I could extract every 4th frame for a 15fps projector, every 3rd frame for a 20fps projector, or every 2nd frame for a 30fps projector.

We *could* also export every single frame for a 60fps projector, but this will double the increase in build size due to having twice the number of frames, and the cost likely outweighs the benefit.

Some video tools may allow you to define a custom framerate when exporting frames. For example, 25fps on a 30fps video. In this instance you may see some stuttering or blended frames in the projector playback, depending on the technique used by the exporter.

When exporting, we recommend to use a naming convention which will put the frames in the correct order when sorted alphanumerically in the Unity inspector (for example, "*<VideoName>_<FrameNumber>.jpg*":

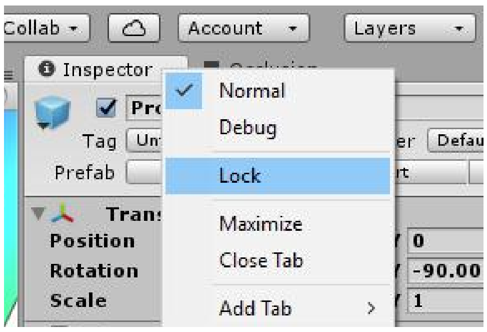
BigBuckBunny_001.jpg
BigBuckBunny_002.jpg
...
BigBuckBunny_423.jpg

Loading multiple images into the projector

Once you have your images imported into Unity using the desired import settings, you can drag and drop them from the Project window into the Images array on the projector.

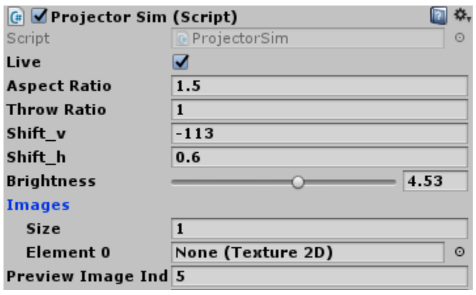
Luckily, you do not have to drag and drop each individual image.

To drag multiple images into the projector simultaneously, first select the Projector object you wish to work on. Right-click on the Inspector tab and click “Lock”. This allows you to select other items while the Inspector stays on the properties of the locked item.



Ensure that the projector has no images currently loaded by right-clicking on the word “**Images**” and selecting “Revert Value to Prefab”. Otherwise the images will be added at the end of the images currently in the array.

You can then select multiple images by first selecting the first image, and holding shift while clicking the final image (assuming your images are in the correct order when sorted alphanumerically). Drag them onto the word “**Images**” on the projector:



Projector Sim (Script)

Script	ProjectorSim
Live	✓
Aspect Ratio	1.5
Throw Ratio	1
Shift_v	-113
Shift_h	0.6
Brightness	4.53
Images	
Size	1
Element 0	None (Texture 2D)
Preview Image Ind	5

The images have likely been added after the empty Image entry. Right-click on any undesired array entries and select “Delete Array Element” until only the desired images are left.

Adjust the **Preview Image Index** value to select which image the projector will preview when in edit mode.

Set the **Image Interval** to the desired time to show each image. When working with video, this is 1/framerate. For example, a 20fps video has an image interval of 1/20 = **0.05**.

Now you can play your scene to see the effect.

Playback Control

There are now a few functions included to enable your own scripts to control the playback of the projector. Examples are included in the *ProjectorControl* scene and *ProjectorControl.cs* script.

```
ProjectorSim projector;
```

To turn the projector on and off, simply enable/disable the ProjectorSim component respectively.

```
projector.enabled = true;
projector.enabled = false;
```

To pause or resume the slideshow playback, call the functions:

```
projector.PauseSlideshow();
projector.PlaySlideshow();
```

You can manually force the slideshow to advance to the next frame with the function:

```
projector.AdvanceSlideshow();
```

Or you can make the projector jump to a specific frame:

```
projector.SetSlideshowIndex(5);
```

Known Issues

1. Occasionally, the projector may stop responding to updates in the editor's Inspector.
   - The cause of this is unclear, but it seems to only sometimes occur on project load, or when Unity has been in the background for an extended period of time, or when scripts are compiled.
   - To rectify the issue, simply play and stop your scene.
2. Longer level load times
   - When a projector has several images in its *Images* array, it will take longer to generate the projected images at the start of the level.
   - To reduce the level load time, you should consider reducing the resolution of your projector, or else projecting in greyscale by unchecking the *Colour* property.
   - If you are projecting a quasi-video with several hundred frames, you may want to consider reducing your sample rate of the source video in order to reduce the number of frames in the projector.
   - In update 1.2 we reduced the time needed to generate subsequent images after the first projected image, by using the first image as a starting point and removing the need to perform the same calculations on every image. In the future we will look at multithreading the generation process, using multiple CPU cores in parallel to cut the required time into fractions of what it is needed for the current implementation.