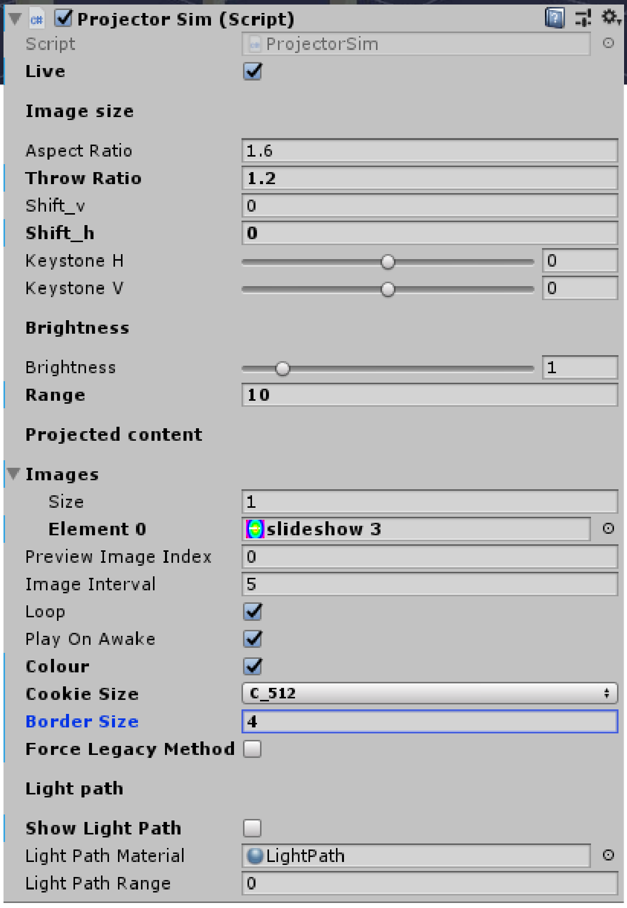
Projector Simulator

White Games

SOME TEXT
And some smaller text



Projector Sim (Script)

Script: ProjectorSim
Live: ✓

Image size

Aspect Ratio: 1.6
Throw Ratio: 1.2
Shift_v: 0
Shift_h: 0
Keystone H: 0
Keystone V: 0

Brightness

Brightness: 1
Range: 10

Projected content

Images
Size: 1
Element 0: slideshow 3
Preview Image Index: 0
Image Interval: 5
Loop: ✓
Play On Awake: ✓
Colour: ✓
Cookie Size: C_512
Border Size: 4
Force Legacy Method: ☐

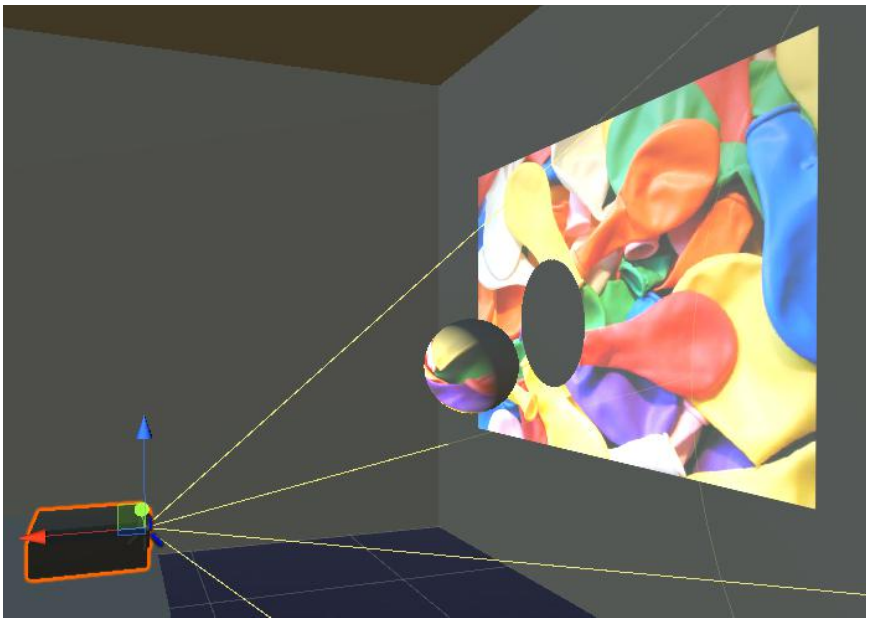
Light path

Show Light Path: ☐
Light Path Material: LightPath
Light Path Range: 0

Table of Contents

Overview

Projector Simulator is a Unity package which allows you to project any video, image, or series of 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

V1.41

- Improved performance when projecting multiple RenderTextures simultaneously

V1.4

- Implemented new shader-based method, massively improving both performance and quality
    - **Unity 2018.3.8f1** and later only
    - Keystoning is not yet supported under this new method. Keystoning can still be used, but the projector will drop down to the “legacy” pre-1.4 pixel-by-pixel CPU-based method
    - You can also choose to force the legacy method to be used should you see any problems with the new GPU-based method
    - New method requires images to be in RGBA32 format and RenderTextures in ARGB32 format (may not be supported on all platforms - more will be added later if needed)

V1.33

- Made RenderTextureProjectors and VideoProjectors aware of each other's processing, so two projectors will not be processed in the same update in order to maintain a steady framerate.
    - If a projector has already processed this frame, then any other projectors will wait for the next frame, and so on.
    - If you still experience frame rate issues with multiple projectors, you may need to reduce the projector's framerate (increase the *image interval*) to allow enough frames for all projectors to process before the first needs to update again.

V1.32

- Added light path geometry for simulating volumetric light
- Custom materials can be applied to the light paths
- Light paths do not currently take keystoning into account

V1.31

- Fixed a minor bug that prevented the new prefabs from working with default values
- Renamed the original “Projector” prefab to “ImageProjector” to be more in keeping with the new prefabs' naming conventions

V1.3

- Added ProjectorSim_RenderTexture script. This new type of projector is able to project RenderTextures (and by extension, video files) instantaneously, removing the need to manually extract a video's frames.
- Added RenderTextureProjector and VideoProjector prefabs

V1.23

- Added script to synchronise projected frames with an audio track (projected content will only progress when audio clip is playing - audio clip and projected content will have same length – see https://youtu.be/a1bsCE3JCPM)

V1.22

- Reduced cookie generation time by 10-30% depending on projector resolution/colour mode/number of images
- Memory optimisations

V1.21

- Added “Range” setting, as previously a projector's range was fixed at 10, and you would have to manually edit each underlying Spotlight's 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

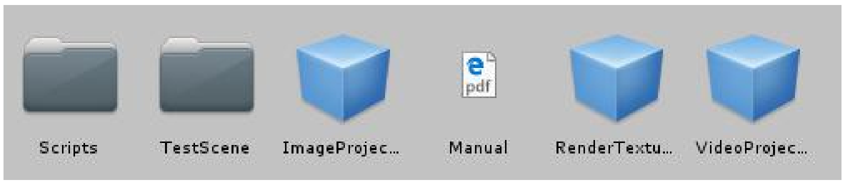
V1.0

- Initial release

How to use

Getting started

After importing the package, you will see 6 items:



The **Scripts** folder contains the 3 main scripts for the asset to work – *ProjectorSim.cs*, *ProjectorSim_RenderTexture.cs* and *CookieCreator.cs*.

It also contains other scripts:

- *ProjectorAudioSync.cs*, which can be used for synchronising projector playback with an audio track;
- *ProjectVideo.cs*, which allows the RenderTextureProjector to project a video file (used in the *VideoProjector* prefab).
- *ThrowBuilder.cs*, which builds light path geometry, for a visible "light cone" effect

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

Finally, there are the **Projector** prefabs which can be placed in your scene.

Positioning the image

1. To begin, drag and drop one of the Projector prefabs into your scene
2. Position the projector so that the light source is in the desired location. Rotate the projector 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

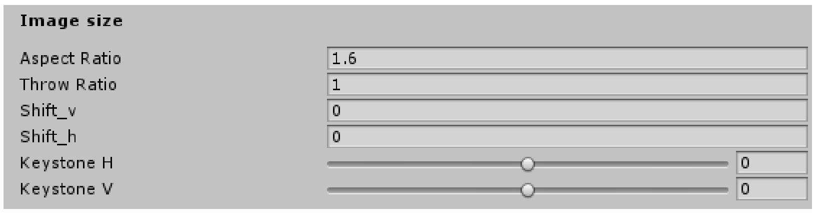


Image size

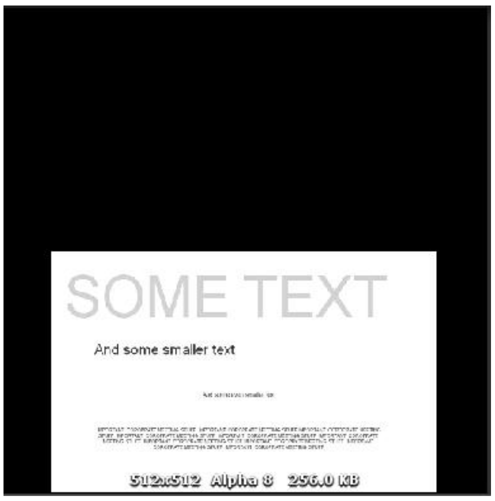
Aspect Ratio	1.6
Throw Ratio	1
Shift_v	0
Shift_h	0
Keystone H	0
Keystone V	0

Aspect Ratio is the aspect ratio of the projected image (width divided by height).

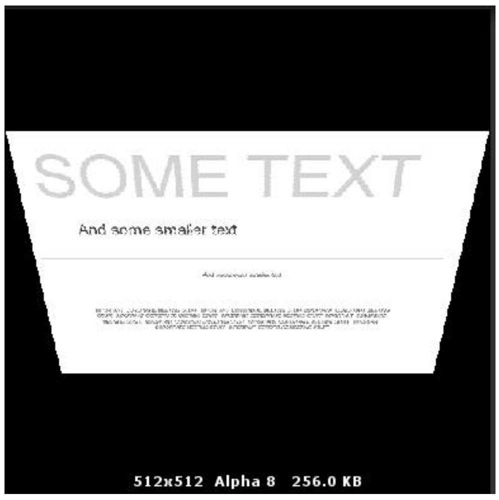
Throw Ratio is equal to the projector distance divided by the image width, and is a standard measurement used by real-world projection lenses. Smaller values result in larger images.

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:



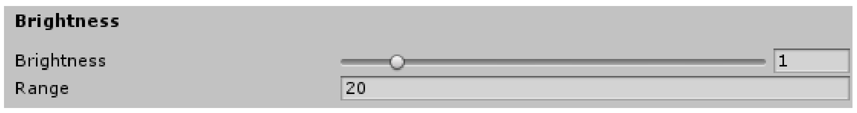
Lens shifted



Keystoned

We will aim to mitigate these in a future update by implementing the keystone adjustment in the shader-based method. Currently, applying keystone forces the projector to drop down to the slower legacy method.

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

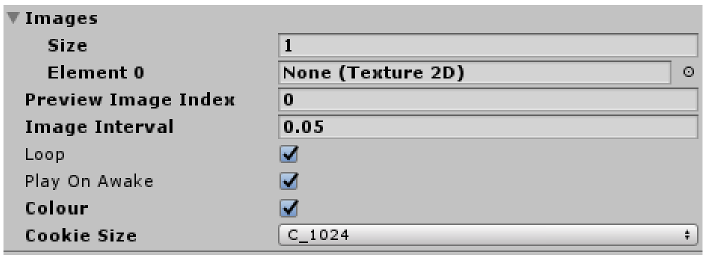


Brightness

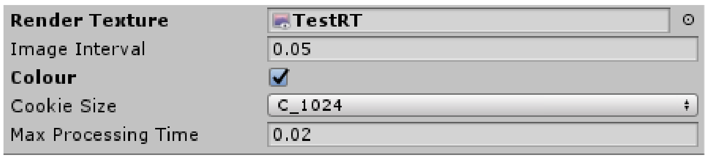
Brightness	1
Range	20

Projecting content

4. Once the image is in place, use the **Projected content** controls to control what the projector will display (controls differ between "ImageProjector" and "RenderTextureProjector" prefabs):



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

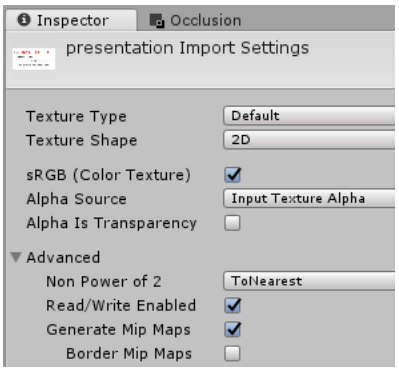


Render Texture: TestRT
Image Interval: 0.05
Colour: ✓
Cookie Size: C_1024
Max Processing Time: 0.02

The **Images** parameter is where you can drag and drop your image files. Several images can be added to an **ImageProjector** prefab 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 images, you can expect it to add several seconds to your level load time under the legacy method. Processing with the new shader-based method in v1.4 will be much faster.

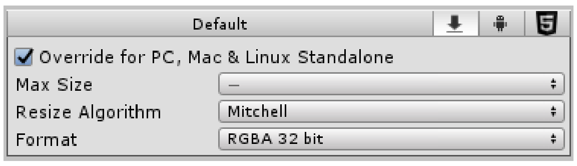
*Tip: You can use external tools such as **VirtualDub**, **VLC Media Player**, **Adobe Premiere**, or your preferred choice of software to convert a video's frames into a sequence of image files.*

The only prerequisites to dragging your images into the projector is that you have ticked **Read/Write Enabled** (to allow the fallback to Legacy method), and set the image format to **RGBA 32 bit** in the image's import settings:



Inspector | Occlusion
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: ☐



Default
✓ Override for PC, Mac & Linux Standalone
Max Size: –
Resize Algorithm: Mitchell
Format: RGBA 32 bit

Support for more image formats will be added in the future if there is a need (please get in touch if so). In the meantime, you should be able to drop down to the Legacy (pre-1.4) method and continue to use whichever format works for you.

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.*

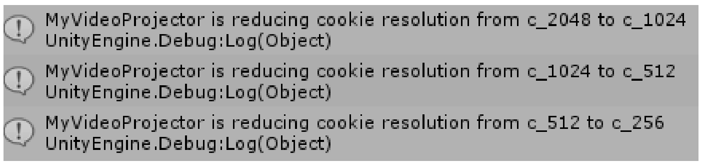
The **Colour** checkbox dictates whether the projected image is projected in full colour, or converted to greyscale. Greyscale projected images are quicker to generate than equivalent colour images.

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.

In a **RenderTextureProjector** or **VideoProjector** prefab, the **Images** parameter is replaced by the **RenderTexture** parameter. Here you can drag and drop the RenderTexture that you wish to project.

The other options in the **RenderTextureProjector** prefabs behave similarly to the variables described above for the standard Projector prefab. The only new addition is the **Max Processing Time** value.

As the cookies now have to be generated in real-time (in the standard Projector prefab they are pre-generated at the start of the level), we should limit the amount of processing time the projector is allowed to take each time it updates its cookie(s). The **Max Processing Time** value is the time allowed, in seconds. The projector will dynamically reduce its resolution at runtime in order to update the cookies at the given **image interval** whilst allowing your game to maintain a steady framerate.



```
MyVideoProjector is reducing cookie resolution from c_2048 to c_1024
UnityEngine.Debug:Log(Object)
MyVideoProjector is reducing cookie resolution from c_1024 to c_512
UnityEngine.Debug:Log(Object)
MyVideoProjector is reducing cookie resolution from c_512 to c_256
UnityEngine.Debug:Log(Object)
```

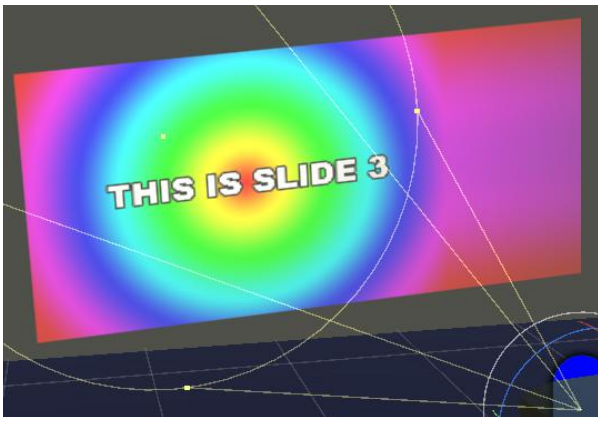
Shader-based method properties

Projector Simulator 1.4 sees the implementation of a new shader-based method which is much faster than the pre-1.4 “Legacy” method.

The Legacy method builds the light cookies pixel-by-pixel and is thus quite CPU-intensive and slow.

The new shader-based method is **much** faster but has one drawback: we no longer have control over every pixel in the resulting light cookies.

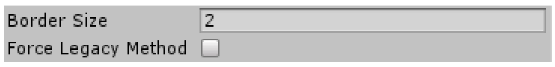
Spotlights require that the edge of a cookie has a black border, or else the light will “leak” out of the spotlight's cone where the cookie is not black at the edge. In the legacy method, we could manually ensure that our cookie was enclosed in a black border at all times, but with a shader method, some downsampling and interpolation can happen, resulting in light leakage:



Here you can see the right edge of the projected image is repeating outside of the spotlight's cone. This is due to the projector's low resolution (c_256) and lack of lens shift.

During processing we add a black border around the projected image and then scale it down to be put in the cookie. If the image is scaled down too far, or too much lens shift applied (which also scales the image down in the cookie), that black border can be lost in the downsampling process.

We therefore added the ability to control the size of the black border:



Border Size 2
Force Legacy Method ☐

Border Size is the size of the border around your projected image(s) before the downscaling occurs. Your projected image will shrink slightly when this is increased, but it usually only requires a few additional pixels. An alternative to increasing the border size is to increase the cookie resolution.

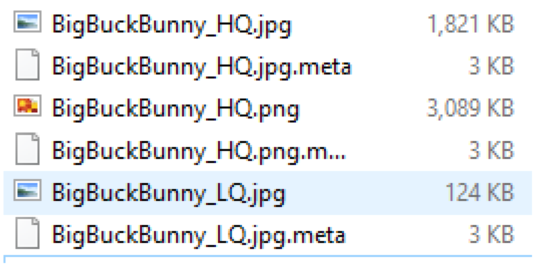
Force Legacy Method allows you to choose to fall back to using the pre-1.4 pixel-by-pixel approach. This should be left unchecked, and should only be enabled if you are having issues with the new shader-based approach. You will have to either play or reload your scene in order to apply the change.

Image Import Settings

Pre-1.4 “Legacy Method” import settings

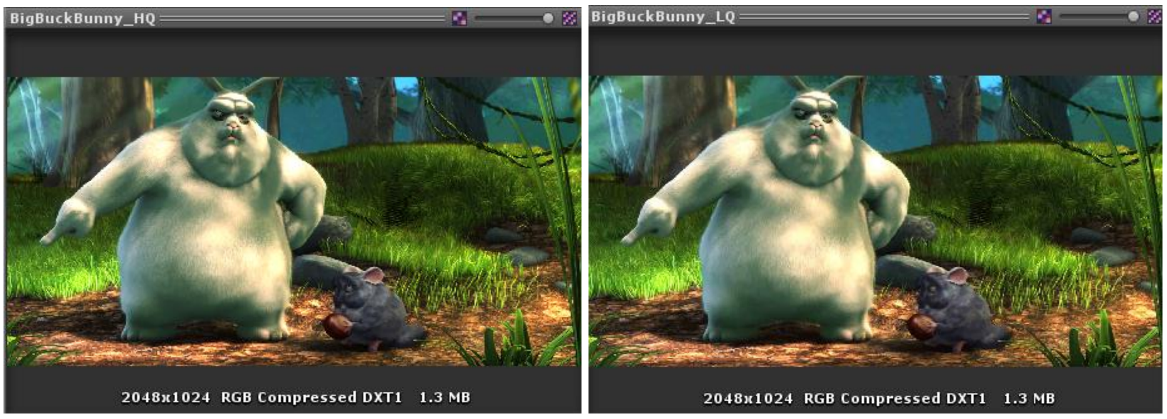
It is possible for an ImageProjector to contain hundreds (possibly thousands) of images to be used in a slideshow or quasi-video. These images will however increase your build size. The more images you add, the more benefit you will get out of properly compressing the images in question in Unity's import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



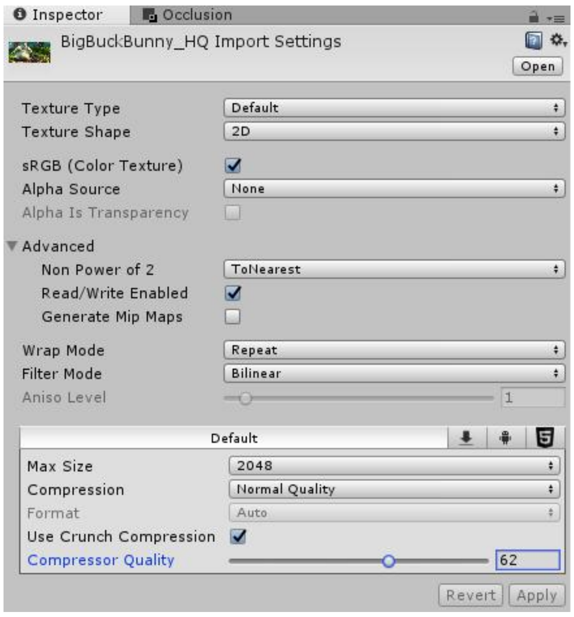
BigBuckBunny_HQ.jpg	1,821 KB
BigBuckBunny_HQ.jpg.meta	3 KB
BigBuckBunny_HQ.png	3,089 KB
BigBuckBunny_HQ.png.m...	3 KB
BigBuckBunny_LQ.jpg	124 KB
BigBuckBunny_LQ.jpg.meta	3 KB

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

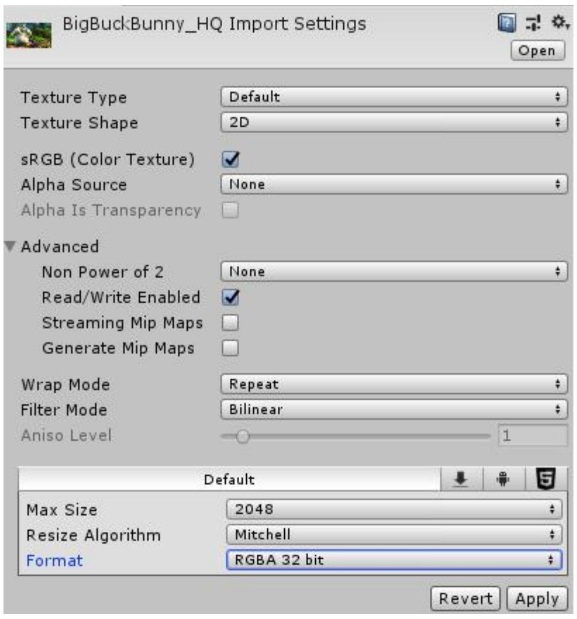
Open

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

Crunch Compression *reduces build size and is useful if an ImageProjector has hundreds of Images.*



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: None
Read/Write Enabled: ✓
Streaming Mip Maps: ☐
Generate Mip Maps: ☐
Wrap Mode: Repeat
Filter Mode: Bilinear
Aniso Level: 1

Default
Max Size: 2048
Resize Algorithm: Mitchell
Format: RGBA 32 bit

Revert | Apply

For stability reasons, the new Shader method requires imported images to be in ***RGBA 32 bit*** *format.*

First of all, ensure that **Read/Write Enabled** is ticked, as this is required for the ImageProjector to project the image under the Legacy method. It is not required for the v1.4 shader method, but it is good to be compatible with the old method should you need to fall back to the Legacy method.

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 Quality** 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

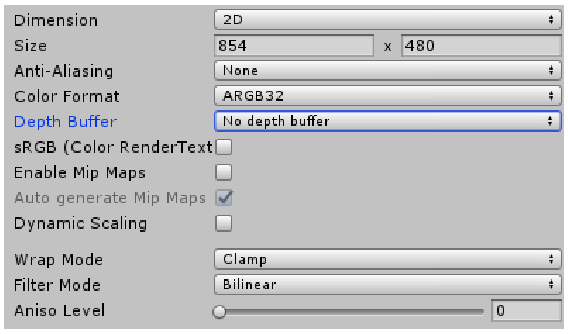
Note that applying crunch compression can take several seconds per image. If you have hundreds of images you can therefore expect crunch compression to take several minutes. Try experimenting with a single image before applying the changes to the whole set.

1.4 Shader Method import settings

Projected Images

Under the new Shader method, projected images must be in **RGBA 32 bit** format. This is to ensure compatibility with other textures used during processing of the cookies. The new Shader-based method therefore does not support Crunch Compression. If your build size is increasing too much due to having many 32-bit images, you may want to consider either forcing the use of the Legacy Method, or compiling your images into a video and using a VideoProjector.

RenderTextures



Dimension	2D
Size	854 x 480
Anti-Aliasing	None
Color Format	ARGB32
Depth Buffer	No depth buffer
sRGB (Color RenderText	☐
Enable Mip Maps	☐
Auto generate Mip Maps	✓
Dynamic Scaling	☐
Wrap Mode	Clamp
Filter Mode	Bilinear
Aniso Level	0

Under the new shader method, RenderTextures that are to be projected with either RenderTextureProjectors or VideoProjectors must be in **ARGB32** format. Again, this is to ensure compatibility with other textures used during processing.

If these formats do not work for you under the new Shader-based method, please get in touch. In the meantime, you should be able to fall back to the Legacy Method.

Quasi-video

It is possible to extract frames from a video and play them back on an ImageProjector 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

You can use the provided ProjectorAudioSync script to syncronise your projector with the audio track from your video. See an example of this on our YouTube channel.

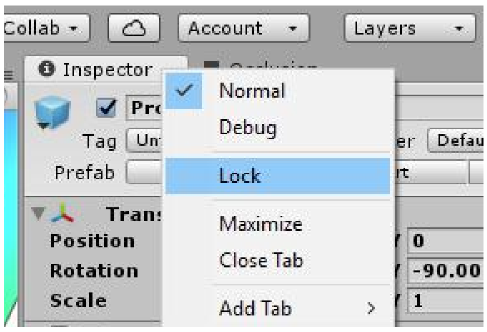
Note: *since v1.3, you may prefer to use a VideoProjector to project a video file, rather than extracting its frames.*

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.

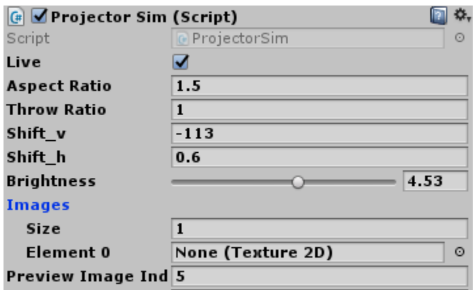
Luckily, you do not have to drag and drop each individual image.

To drag multiple images into the projector simultaneously, first select the ImageProjector prefab 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

Since v1.1, there are a few functions included to enable your own scripts to control the playback of the ImageProjector. Examples are included in the *ProjectorControl* scene and *ProjectorControl.cs* script.

```
ProjectorSim pj;
```

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

```
pj.enabled = true;
pj.enabled = false;
```

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

```
pj.PauseSlideshow();
pj.PlaySlideshow();
```

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

```
pj.AdvanceSlideshow();
```

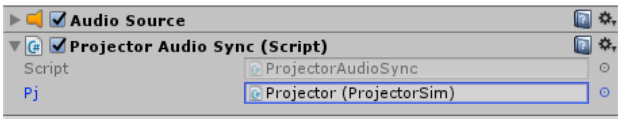
Or you can make the projector jump to a specific frame:

```
pj.SetSlideshowIndex(5);
```

RenderTextureProjectors similarly have *Pause()* and *Play()* functions. You can turn the RenderTexture Projector on and off in the same manner as a regular Projector – by setting the *ProjectorSim_RenderTexture*'s “enabled” property as above.

Synchronising projected content with an audio track

In v1.23 a new script was created, *ProjectorAudioSync.cs*. This script can be added to an object with an Audio Source component, and then given a reference to the ImageProjector you want to have synchronised to the audio:



Audio Source
Projector Audio Sync (Script)
Script: ProjectorAudioSync
Pj: Projector (ProjectorSim)

You should also turn off "Play On Awake" on the ImageProjector.

This will cause the projector to play through its frames at a constant rate whenever the audio is playing, such that the projected content and the audio clip will have the same length. If you want to pause the projector, just pause the audio.

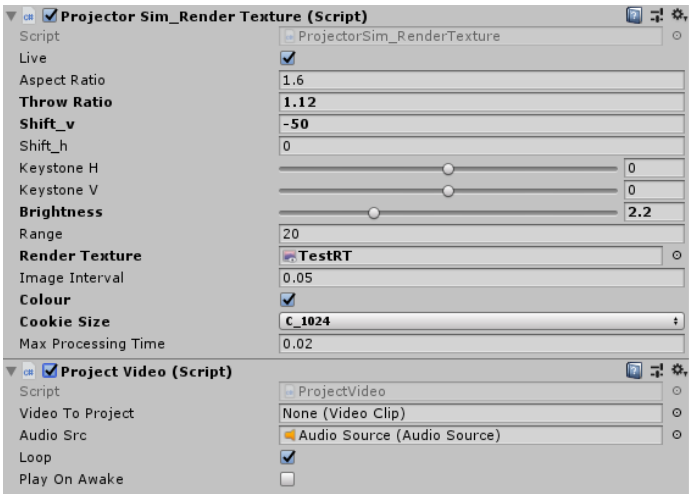
An example of this can be seen in our video: https://youtu.be/a1bsCE3JCPM

Note: *since v1.3, you may prefer to use a VideoProjector to project a video file, as the VideoProjector has its own AudioSource built in.*

Projecting a video file or RenderTexture

The optimisations released in v1.22 finally made it feasible to generate a Projector's cookies in real-time. In v1.3, 2 new prefabs were added; **RenderTextureProjector**, and **VideoProjector**.

RenderTextureProjector simply allows you to project a live RenderTexture instead of predefined static images. One big advantage of this new type of projector is that it does not require the lights' cookies to be generated at the start of the level, increasing level load time. Instead, the cookies are generated in real-time at a predefined framerate. The main disadvantage of this type of projector is that in order to assure that we maintain a smooth framerate, we have to limit the resolution of the projected image. The main cause for this is Unity's *Texture2D.Apply()* function, which cannot be made to work asynchronously.



Projector Sim_Render Texture (Script)

Script	ProjectorSim_RenderTexture
Live	✓
Aspect Ratio	1.6
Throw Ratio	**1.12**
Shift_v	**-50**
Shift_h	0
Keystone H	0
Keystone V	0
Brightness	**2.2**
Range	20
Render Texture	**TestRT**
Image Interval	0.05
Colour	✓
Cookie Size	**C_1024**
Max Processing Time	0.02

Project Video (Script)

Script	ProjectVideo
Video To Project	None (Video Clip)
Audio Src	Audio Source (Audio Source)
Loop	✓
Play On Awake	☐

A **VideoProjector** makes use of the **ProjectorSim_RenderTexture** script, and simply calls some functions to render your given video into the RenderTexture used by the projector.

The **VideoProjector** prefab has its own default AudioSource for the video, which you can modify to your need, but you can reference your own on the script if you prefer.

When projecting a video file, you should call *GetVideoPlayer()* and use that reference to play and pause the video, rather than calling the RenderTextureProjector's *Play()* and *Pause()* functions, otherwise the projected image will pause but the video will continue to play in the background.

The exception is when the projector's PlayOnAwake is unchecked.You should first call *ProjectorSim_RenderTexture.*`PlayVideoProjector()` to start the video, then use the VideoPlayer reference's *Play()* and *Pause()* functions thereafter. There is an example script on the VideoProjector in the demo scene.

Note: The resolution that is possible to achieve is highly dependant on processor speed. On a 4.0GHz processor, we had reliable results with a cookie resolution of 256x256. The code should run on mobile and other platforms but has not been tested and performance may not be sufficient. If you require a higher resolution then it is recommended to use the standard ProjectorSim prefab with predefined images.

Known Issues/Troubleshooting

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 (fixed* in v1.4, Unity 2018.3.8f1 and later)
    - 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.~~
    - For update 1.22, we experimented with using multithreading to optimise the generation times. Unfortunately, we were forced to conclude that it is not possible due to Unity's functions not being thread-safe (specifically the Texture2D's *SetPixels* and *Apply* functions). We also experimented with multithreading between each call to these functions (a single image would use multiple threads, rather than multiple threads each working on multiple images), but did not see any performance improvements. After finishing our multithreading experiments, we noticed a performance improvement even when multithreading was disabled, introduced by the required restructuring of the code. These optimisations are what make up V1.22.
    - ~~Multithreading may still be possible in the future, but until Unity's functions become thread-safe, it will require the use of native OpenGL/DirectX/other API code.~~

    * In update 1.4 we implemented a new shader-based method which should stupendously improve any previously seen issues with level load times, provided those projectors do not use keystoning and therefore are compatible with the new method.
3. The projector(s) introduce a strange coloured lighting effect on the projected image, or the rest of the scene

- This is likely caused by the **Pixel Light Count** being set too low – each colour projector adds 3 spotlights to your scene, and when using Forward Rendering the pixel light count setting may need to be increased.
- The **Pixel Light Count** setting can be found in the Quality settings. An example of this effect and fix can be seen in our Introduction and Tutorial video.

4. A projected image appears in the Game view, but not in the Scene view
   - As Projector Simulator makes use of real Unity spotlights, ensure you have lighting enabled in the Scene view.
5. Several RenderTextureProjectors in a scene can cause lag (fixed*)
   - Each RenderTextureProjector and Videoprojector must process its current image within a single frame, within the allowed time set on the projector.
   - Each projector is not aware of the other's processing time.
   - When two or more of these projectors try to process their image in a single frame, the total processing time for all projectors can be above the allowed time in the Projectors' settings. This can cause a stutter in the frame rate if the processing of all Projectors takes too long.

   * Functionality was added in v1.33 to ensure two projectors never try to process their image in the same frame (if one projector has already processed, the other projector will wait for the next frame)

   * Any lag should be dramatically reduced if using the new shader-based method added in v1.4 (only available in Unity 2018.3.8f1 or later).
6. SRP Support?
   - We plan to support SRP in a future release
   - We have a version of Projector Simulator which works under HDRP in the Unity Editor, but when we build the project, Unity crashes. The crash was reported to Unity and they have reproduced it, recognising it as a bug. We will look at SRP support again when they fix this bug in a future release of Unity.