Note: this document is accurate at the time of writing, but behaviors may change in Unity 2020.1.b1 and/or 2020.2.a1.

Streaming Virtual Texturing - User Guide

Unity version: 2020.1 (expected from beta 1)

System requirements

GPU requirements

- GPU compute
- Texture2DArrays
- AsyncReadBack.

Supported platforms

- Windows
- Mac
- Playstation 4
- Xbox One

Supported graphics APIs

- DirectX 11
- DirectX 12
- Metal
- Vulkan

Render pipeline support

In Unity 2020.1, Virtual Texturing is supported in the High Definition Render Pipeline version 9.0.0-preview and above, in Shaders created using Shader Graph.

> Note for alpha users
>
> At the time of writing, the PR adding support for Virtual Texturing hasn't landed in HDRP and Shader Graph yet. You can track the PR here. A SRP version with Virtual Texturing support is embedded in the sample project.

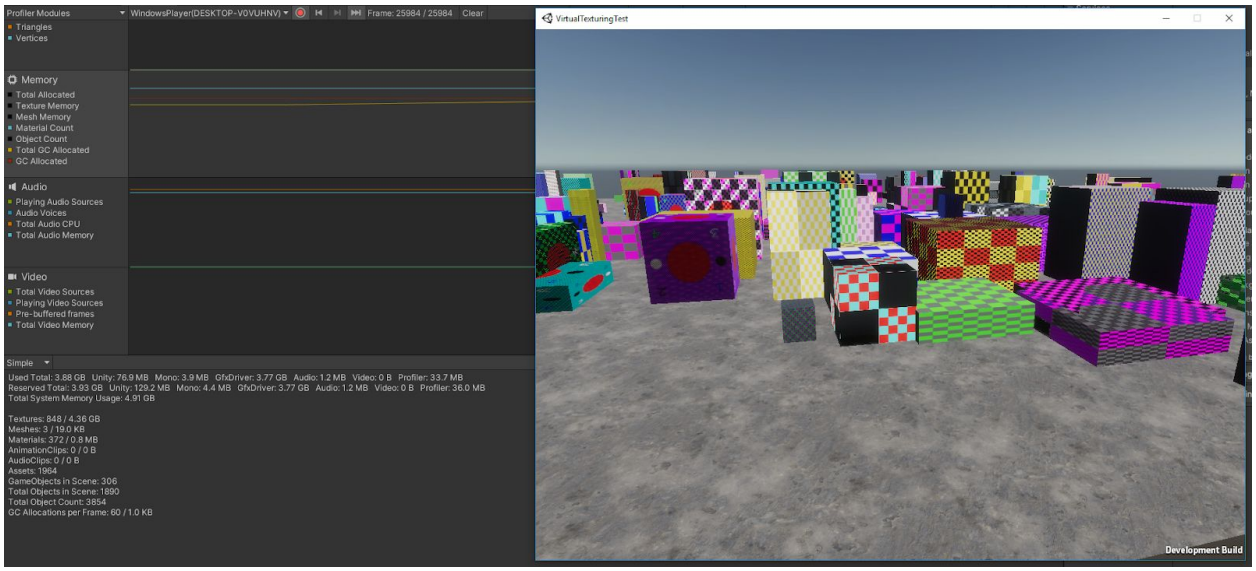
What is Streaming Virtual Texturing?

Streaming Virtual Texturing is a feature that reduces GPU memory usage and Texture loading times when you have many high resolution Textures in your scene. It works by splitting Textures into tiles, and progressively uploading these tiles to GPU memory when they are needed.

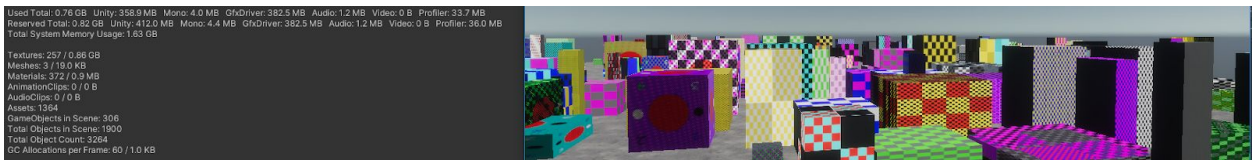
Streaming Virtual Texturing has a fixed memory cost that mostly depends on your frame resolution. The more high resolution Textures you have in your Scene, the more GPU memory you can save with Streaming Virtual Texturing.

Streaming Virtual Texturing uses the Granite SDK runtime. The workflow requires no additional import time, no additional build step and no additional streaming files. You work with regular Unity Textures in the Unity Editor, and Unity generates the Granite SDK streaming files when it builds your Project.

Streaming Virtual Texturing is part of a larger feature called Virtual Texturing, which will include additional functionality in future releases of Unity.



Scene with 800+ 4K textures using 3.77GB of video memory without virtual texturing or other streaming.



The same scene with 800+ 4K textures using 382.5MB of video memory (instead of 3.77GB) when using Streaming Virtual Texturing with all textures set to "Virtual Texturing Only" in the texture importer.

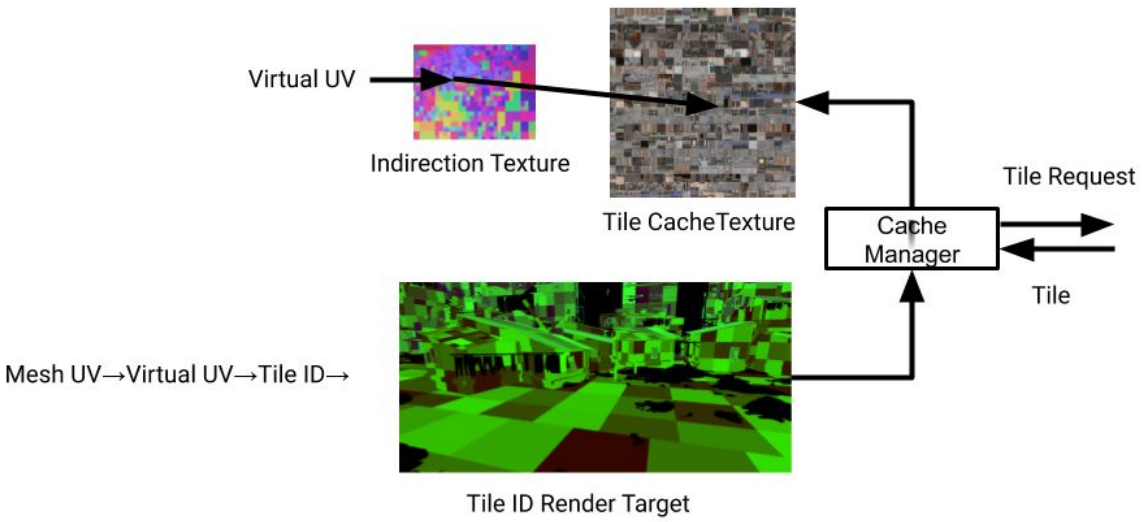
How Streaming Virtual Texturing works

The Streaming Virtual Texturing system divides Textures into tiles.

At runtime, when the Streaming Virtual Texturing system samples a Texture:

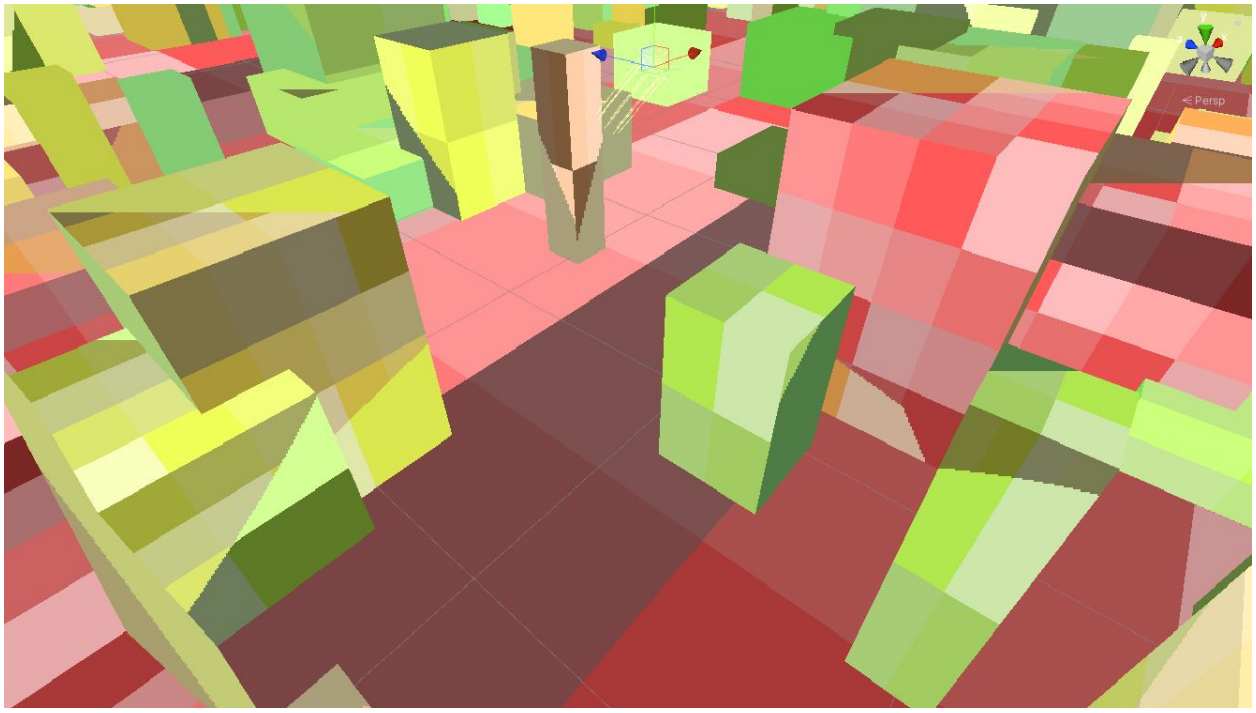
- The system samples an indirection texture, calculates the non-virtual UVs, and samples a cache texture with these UVs.
- At the same time, the virtual UVs are translated to a tile ID. The system binds an additional render target that receives these tile IDs, which is copied back to main memory asynchronously and processed by the CPU on a separate thread to create requests to load these tiles into a cache of GPU memory (if they are not already present).

The cost of these additional runtime operations mean that it is more efficient to group Textures together, and to sample them all at the same time. Grouping Textures together in this way is called stacking Textures. A group of Textures that are sampled at the same time using the same UV coordinates is called a Texture Stack.



Because the Virtual Texturing system translates UV coordinates to tile IDs while a frame is being rendered, it can take from milliseconds to seconds until the requested tiles are loaded into the GPU cache, and it is not guaranteed that these tiles will even be loaded into the cache.

When the desired tile is not in the cache, the Streaming Virtual Texturing system has an automatic fallback mechanism; it samples tiles from a higher mipmap level until the requested tile is in the cache. This results in a lower level of detail until the tile has loaded.



A debug view using distinct colors for every tile id. The view clearly shows how many neighboring screen pixels request the same tile (same color). The color hue (greenish, redish, …) represents the mipmap level that is requested.

Enabling Streaming Virtual Texturing in your Project

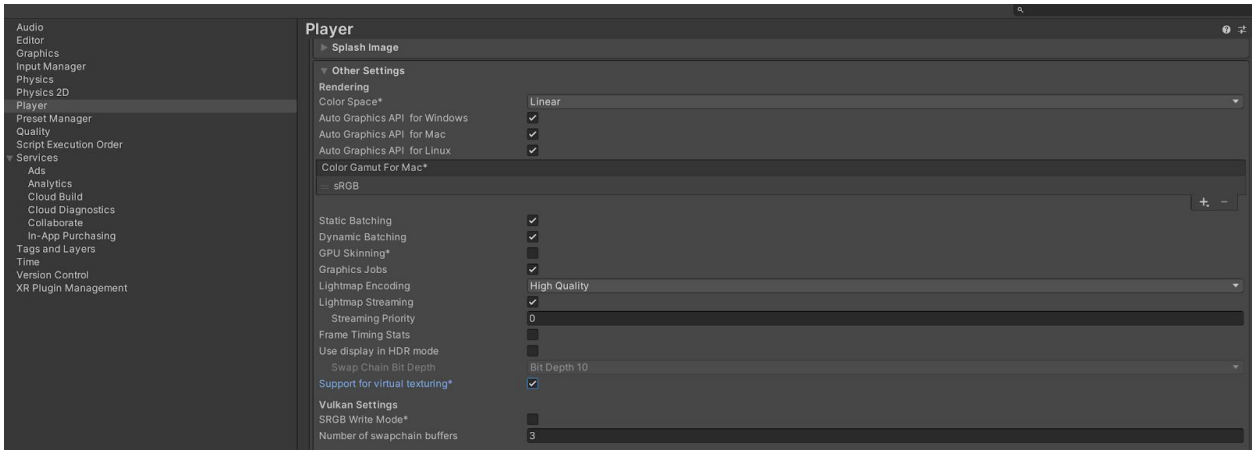
Note for alpha users

In Unity 2020.1.b1:

- The Virtual Texturing checkbox is at the top of the Player Project Settings (this has since moved, see below)
- You need to enable the built-in module “Virtual Texturing” in the Package Manager (this will always be enabled and hidden in the future)

To use Streaming Virtual Texturing in your Project, you need to enable the Virtual Texturing feature. Virtual Texturing is a project wide setting that is shared for all platforms.

You will not be able to build a player for unsupported platforms and graphics APIs. The Virtual Texturing system might allocate resources (buffers, etc.) when the setting is enabled, even if you do not use the feature in your Project. Do not enable Virtual Texturing if you are not using it.



When you enable Virtual Texturing in your Project, Unity adds the following compiler directives:

- ENABLE_VIRTUALTEXTURES: C# define that evaluates to True if Streaming Virtual Texturing is supported and enabled in the project
- ENABLE_VIRTUALTEXTURING: C++ (and C# editor) define if Streaming Virtual Texturing is possible on the current build target
- UNITY_VIRTUAL_TEXTURING: Shader version of ENABLE_VIRTUALTEXTURES

Note that these might be renamed in future versions of Unity.

Enabling Streaming Virtual Texturing in your Shader

You can use Streaming Virtual Texturing with Shaders created in Shader Graph. The default HDRP Shaders (Lit, Unlit, and so on) do not support Streaming Virtual Texturing.

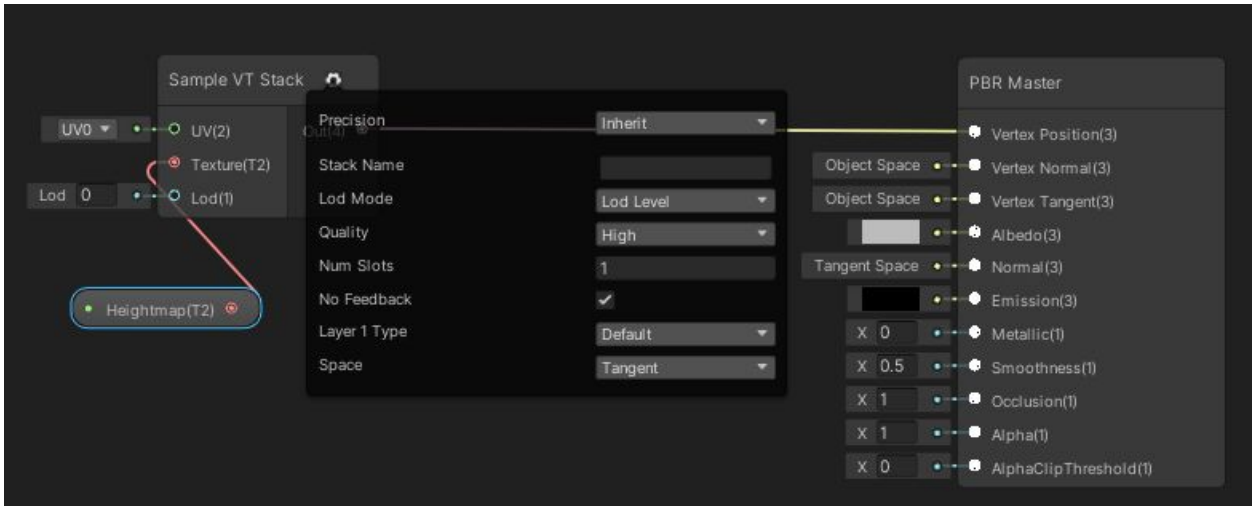
To stream a Texture using Streaming Virtual Texturing, you must add it to a **Sample VT Stack** node. A **Sample VT Stack** node defines a Texture Stack; each Texture assigned to the node will be sampled using the same UV coordinates. Because it is more efficient to sample multiple Textures that are part of the same Texture Stack than it is to sample multiple Texture Stacks, you should combine Textures into the same **Sample VT Stack** node where possible .

When you use a **Sample VT Stack** node in a pixel Shader, the Virtual Texturing system automatically streams in Texture tiles based on the UVs sampled in the VT stack nodes. This is called automatic screenspace requesting.

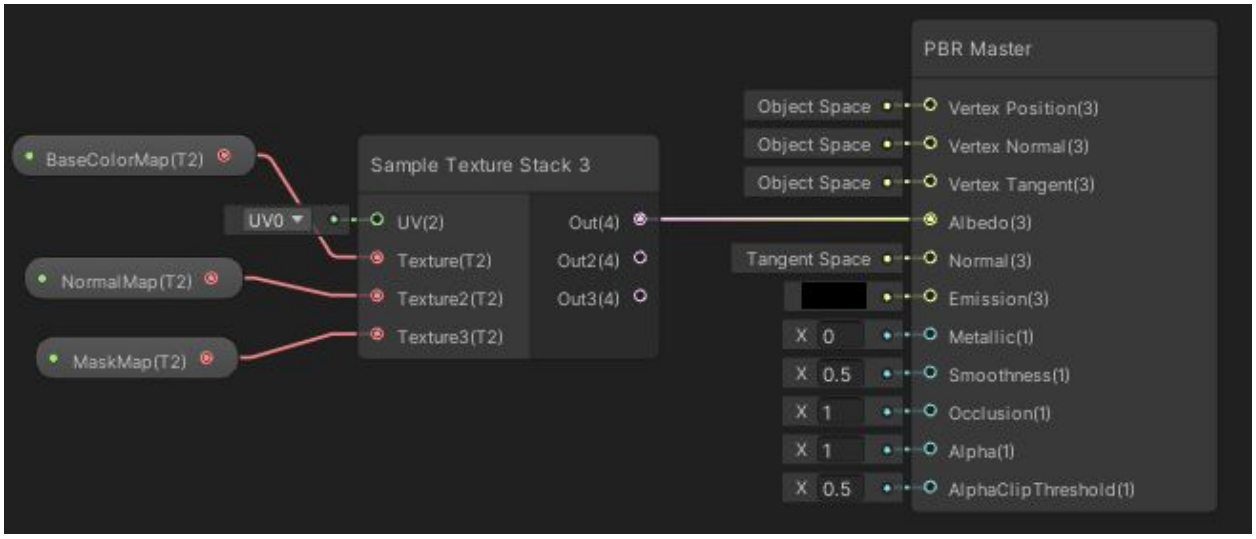
When you use a **Sample VT Stack** node in a vertex Shader, you first need to enable **No Feedback** on the **Sample VT Stack** node, and then select the **Lod Level** mode. Then you must manually request that the system loads the tiles, using a script. You do this using the *VirtualTexturing.System.RequestRegion()* API, like this:

VirtualTexturing.System.RequestRegion(Material, Stack ID, Rect, mipmap, numMips)

You can use this API to load any tiles that are not yet visible. This allows you to build a prefetching system for example. You need to call this API every frame or the system will assume that the tiles are not needed anymore and potentially evict them if other tiles are streamed.




Create Node
Input
Basic
Geometry
Gradient
High Definition Render Pipeline
Lighting
Matrix
PBR
Scene
Texture
Cubemap Asset
Sample Cubemap
Sample Procedural VT Texture Stack
Sample Texture 2D
Sample Texture 2D Array
Sample Texture 2D LOD
Sample Texture 3D
Sample VT Stack


BaseColorMap(T2)
NormalMap(T2)
MaskMap(T2)
UV0
Sample Texture Stack 3
UV(2)
Texture(T2)
Texture2(T2)
Texture3(T2)
Out(4)
Out2(4)
Out3(4)
PBR Master
Object Space
Vertex Position(3)
Object Space
Vertex Normal(3)
Object Space
Vertex Tangent(3)
Albedo(3)
Tangent Space
Normal(3)
Emission(3)
X 0
Metallic(1)
X 0.5
Smoothness(1)
X 1
Occlusion(1)
X 1
Alpha(1)
X 0.5
AlphaClipThreshold(1)

Cache Management

Virtual Texturing uses fixed sized texture caches in GPU memory. There is one cache per graphics format. You can configure the default size of a cache and overwrite its default size.

The Virtual Texturing system allocates a cache of a given graphics format when it renders a Material that has a Texture Stack with a Texture that uses that format. The cache size (in MB) is configured on the HDRP Asset. This means that you can have different configurations for each quality level.

You can also set the cache size in a script, using the *VirtualTexturing.System.ApplyVirtualTexturingSettings()* API.

> **Note for alpha users**
>
> At the time of writing, *VirtualTexturing.System.ApplyVirtualTexturingSettings()* does not recreate existing caches, so calling this api might have no effect. This should change by 2020.1 final.

Changing the cache size results in costly CPU and GPU operations, and blocks both the main thread and the render thread. The length of time it takes to change the cache size depends on the size and number of caches. To avoid noticeable freezes or stuttering, you should change the cache size at a time when frame rate is less important, such as when loading a level.

The optimal size of a given cache is mostly determined by the output screen resolution and the number of layers in each Texture Stack that use that graphics format. The optimal size is less influenced by the number of Materials in the Scene or the resolution of the Textures. For a full HD screen resolution on the highest quality setting, the combined total of GPU cache sizes is typically 700MB.

Cache thrashing occurs when the size of a cache is too low to accommodate its contents, and it must load and unload tiles in the same frame. The Virtual Texturing system automatically prevents cache thrashing by reducing Texture quality as needed. It works by monitoring cache usage and automatically managing a mipmap bias for the Texture Stack sampling in pixel

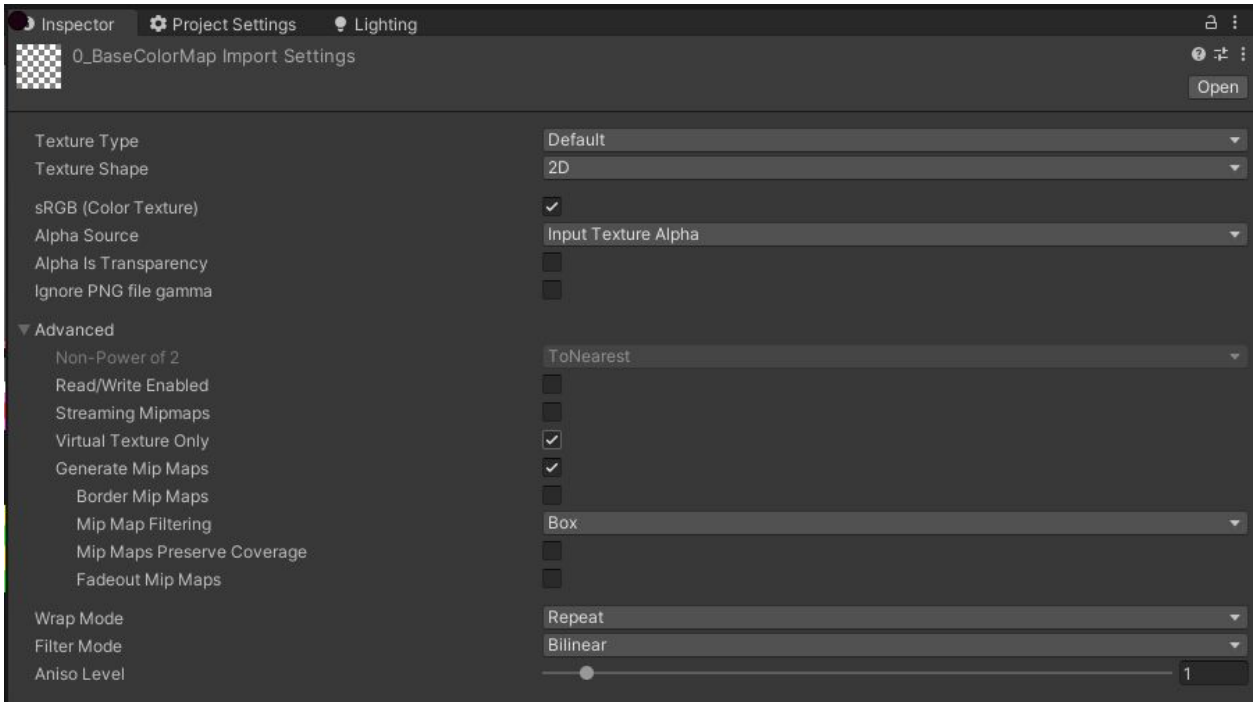
Shaders. If the cache is overused, then the mipmap bias will increase. This ensures that tiles from higher mip levels will be requested, which results in fewer tiles. The cache can therefore contain all of the requested tiles.

If you find that your Textures look blurry, try increasing the size of your caches. If the caches are already large enough to render frames at the highest quality, increasing the cache size further means that the cache can retain tiles for longer before evicting them. This can reduce Texture popping when moving or turning the Camera, as tiles that were previously visible are more likely to still be in the cache.

Marking Textures as "Virtual Texturing Only"

In the Unity Editor, you can mark Textures that are used only by the Virtual Texturing system as **Virtual Texturing Only**. This optimizes memory usage and Scene loading times in the Editor, and reduces the size of the build.

You do this by enabling the **Virtual Texturing Only** property in the Texture Importer of each Texture. Note that if you mark a Texture as **Virtual Texturing Only**, you cannot use it with a regular Texture Sampler in your project.



When you load a Scene in the Unity Editor, the Editor loads all referenced Textures into both CPU and GPU memory by default. The Unity Editor cannot stream Textures into GPU memory from disk, but the Streaming Virtual Texturing can stream them from CPU memory. Marking a Texture as **Virtual Texturing Only** means that the Editor loads it into CPU memory only on Scene load, and then streams tiles extracted from this Texture into GPU memory on demand. For a Project with many high resolution textures, this significantly reduces both GPU memory usage and Scene loading times in the Unity Editor.

By default, Unity includes all Textures that are sampled in Shader Graph in the build as standard Texture artifacts. In addition to this, the Virtual Texturing system imports all Textures that are used in Texture Stacks into a tiled streaming file, which Unity also includes in the build. If a Texture is not marked as **Virtual Texturing Only**, it is therefore included twice in the build;

once as a standard Texture artifact, and once in the tiled streaming file. If a Texture is marked as **Virtual Texturing Only**, Unity does not include it in the build as a standard Texture artifact, and includes it only in the tiled streaming file.

Important Notes

- The Virtual Texturing system is built on top of Unity Texture2D. Virtual Texturing Textures go through the same importer, which limits the maximum Texture size. Currently, no Textures larger than 16K x 16K are supported. Also, there is no support for UDIM Textures, or merging tiled image files into one large Texture.
- AssetBundles are not supported, including Adressables and Live Link.
- Not all Texture graphics formats are supported. The supported formats are:
    - GraphicsFormat::kFormatRGBA_DXT1_SRGB,
    - GraphicsFormat::kFormatRGBA_DXT1_UNorm,
    - GraphicsFormat::kFormatRGBA_DXT3_SRGB,
    - GraphicsFormat::kFormatRGBA_DXT3_UNorm,
    - GraphicsFormat::kFormatR_BC4_UNorm,
    - GraphicsFormat::kFormatRG_BC5_UNorm,
    - GraphicsFormat::kFormatRGB_BC6H_SFloat,
    - GraphicsFormat::kFormatRGB_BC6H_UFloat,
    - GraphicsFormat::kFormatRGBA_BC7_SRGB,
    - GraphicsFormat::kFormatRGBA_BC7_UNorm,
    - GraphicsFormat::kFormatR8_SRGB,
    - GraphicsFormat::kFormatR8_UNorm,
    - GraphicsFormat::kFormatR8G8_SRGB,
    - GraphicsFormat::kFormatR8G8_UNorm,
    - GraphicsFormat::kFormatR8G8B8_SRGB,
    - GraphicsFormat::kFormatR8G8B8_UNorm,
    - GraphicsFormat::kFormatR8G8B8A8_SRGB,
    - GraphicsFormat::kFormatR8G8B8A8_UNorm,
    - GraphicsFormat::kFormatR16_SFloat,
    - GraphicsFormat::kFormatR16_UNorm,
    - GraphicsFormat::kFormatR16G16_SFloat,
    - GraphicsFormat::kFormatR16G16_UNorm,
    - GraphicsFormat::kFormatR16G16B16A16_SFloat,
    - GraphicsFormat::kFormatR16G16B16A16_UNorm,
    - GraphicsFormat::kFormatR32_SFloat,
    - GraphicsFormat::kFormatR32G32_SFloat,
    - GraphicsFormat::kFormatR32G32B32A32_SFloat,
    - GraphicsFormat::kFormatA2B10G10R10_UNormPack32,
    - Not supported: R16G16B16
    - GraphicsFormat::kFormatR16G16B16_UNorm,
    - GraphicsFormat::kFormatR16G16B16_SFloat,

    - GraphicsFormat::kFormatR16G16B16_SNorm,
    - GraphicsFormat::kFormatR16G16B16_UInt,
    - GraphicsFormat::kFormatR16G16B16_SInt,
    - Not supported: R32G32B32
    - GraphicsFormat::kFormatR32G32B32_UInt,
    - GraphicsFormat::kFormatR32G32B32_SInt,
    - GraphicsFormat::kFormatR32G32B32_SFloat,
    - Not supported: channel transform SNorm
    - GraphicsFormat::kFormatR8_SNorm,
    - GraphicsFormat::kFormatR16_SNorm,
    - GraphicsFormat::kFormatR8G8_SNorm,
    - GraphicsFormat::kFormatR16G16_SNorm,
    - GraphicsFormat::kFormatR8G8B8_SNorm,
    - GraphicsFormat::kFormatR8G8B8A8_SNorm,
    - GraphicsFormat::kFormatR16G16B16A16_SNorm,
    - GraphicsFormat::kFormatR_BC4_SNorm,
    - GraphicsFormat::kFormatRG_BC5_SNorm,
    - Not supported: channel transform UInt
    - GraphicsFormat::kFormatR8_UInt,
    - GraphicsFormat::kFormatR16_UInt,
    - GraphicsFormat::kFormatR32_UInt,
    - GraphicsFormat::kFormatR8G8_UInt,
    - GraphicsFormat::kFormatR16G16_UInt,
    - GraphicsFormat::kFormatR32G32_UInt,
    - GraphicsFormat::kFormatR8G8B8_UInt,
    - GraphicsFormat::kFormatR8G8B8A8_UInt,
    - GraphicsFormat::kFormatR16G16B16A16_UInt,
    - GraphicsFormat::kFormatR32G32B32A32_UInt,
    - GraphicsFormat::kFormatA2B10G10R10_UIntPack32,
    - Not supported: channel transform SInt
    - GraphicsFormat::kFormatR8_SInt,
    - GraphicsFormat::kFormatR16_SInt,
    - GraphicsFormat::kFormatR32_SInt,
    - GraphicsFormat::kFormatR8G8_SInt,
    - GraphicsFormat::kFormatR16G16_SInt,
    - GraphicsFormat::kFormatR32G32_SInt,
    - GraphicsFormat::kFormatR8G8B8_SInt,
    - GraphicsFormat::kFormatR8G8B8A8_SInt,
    - GraphicsFormat::kFormatR16G16B16A16_SInt,
    - GraphicsFormat::kFormatR32G32B32A32_SInt,
    - GraphicsFormat::kFormatA2B10G10R10_SIntPack32,
- Crunch compression is not supported

- Textures in the Virtual Texturing system don't have mipmaps smaller than the tile size (128x128 pixels). The sampling is clamped to this mip so it is possible to see aliasing for certain content at a distance.
- No trilinear support in the Editor (the players do support trilinear)
- Mirror wrapping mode is not supported
- Per-axis clamping (e.g., repeating horizontally, clamping vertically) is not supported
- Aspect ratios of stack layers must match
- Non-Pow2 textures are not supported
- Max anisotropic filtering is 8 (borders are 8)
- All the Texture slots of the **Sample VT Stack** node need to be assigned in ShaderGraph
- A Texture assigned to different **Sample VT Stack** nodes (that have a different combination of Textures assigned) will be stored once per node with that configuration in the Player. Using Virtual Texturing can therefore increase the build size on disk.
- There is no streaming from disk in the Editor (Player only for now)
- Setting textures that are part of a stack dynamically in the Player is not supported
    - Material.SetTexture, Renderer.SetPropertyBlock
- There are some limits on the use of the **Sample VT Stack** node in shadergraph
    - Not supported in "Decal Graph" shaders (Decail, Projectors).
    - Not supported "Surface Type" = "Transparent" shaders
    - Subgraphs that export texture properties attached to a vt stack sample node can only be instanced one in the material.
    - Texture (Texture parameters, "Texture Asset" node,..) to Vt stack sample node connections have certain limitations. E.g. the same texture cannot be attached to two nodes.
    - Using the node in vertex shaders (connected to the "Vertex Position" output) is only possible if "Lod Mode" is "Lod Level" and "No Feedback" is enabled.