Manual: Piglet glTF Importer 1.3.8

- Online Manual: https://awesomesaucelabs.github.io/piglet-manual/
- Asset Store Page: https://assetstore.unity.com/packages/slug/173425
- Support Email: “awesomesaucelabs” (at gmail)

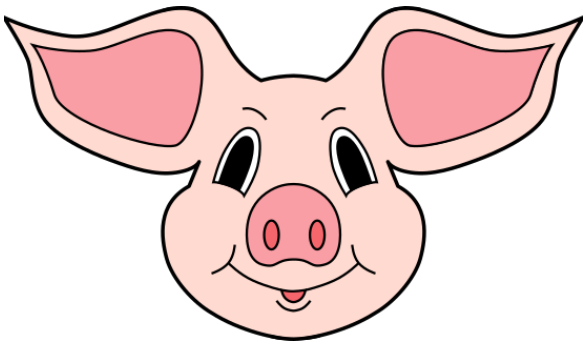


Table of Contents

- Introduction
- Features
    - Supported glTF Extensions
- Known Issues
- Installation
- Editor Imports
    - Importing glTF Models into your Unity Project
    - Selecting Materials Variants (KHR_materials_variants)

    - Editor Animation Tutorial
        - Previewing Animations in the Editor
        - Playing (Mecanim) Animations at Runtime
    - Editor Import Options (Piglet Options Window)
- Runtime Imports
    - Runtime Import Tutorial
    - Runtime Materials Variants Tutorial (KHR_materials_variants)
    - Runtime Animation Tutorial
    - Runtime Import API
        - Overview
        - Creating a GltfImportTask
        - Configuring Callbacks on a GltfImportTask
        - Executing a GltfImportTask
        - Runtime Import Options
- Optimizing glTF Files
    - Supercompressed Textures (Unity 2019.3+)
        - Overview
        - Installing KtxUnity
        - Supercompressing Your glTF Textures
    - Draco Mesh Compression (Unity 2019.3+)
        - Overview
        - Installing DracoUnity
        - Draco-compressing Your glTF Meshes
- URP Support (Unity 2019.3+)
- Sample Application: PigletViewer
- Changelog
- Footnotes

Introduction

Piglet is a Unity asset that allows you to load 3D models from glTF files, both in the Editor and at runtime. This gives you the ability to import 3D models while your game is running, and also gives you access to a huge collection of free/paid glTF models from Sketchfab.

Visit the Web Demo[1] to try Piglet before you buy it.

Features

- import glTF models in the Editor or at runtime
- import glTF models from `.gltf`, `.glb`, or `.zip` files, using file paths or HTTP URLs
- import and play glTF animations (including skins and blendshapes)
- import glTF textures and materials, for use with your own models
- supports supercompressed textures via KtxUnity (Unity 2019.3+)
- supports Draco mesh compression via DracoUnity (Unity 2019.3+)
- tested with glTF models from Sketchfab and Blender
- supported render pipelines: built-in (Unity 2018.4+), URP (Unity 2019.3+)
- supported platforms: Windows, Mac, Android, iOS, WebGL, UWP
- full source code provided

Supported glTF Extensions

The glTF format supports “extensions”, in order to provide functionality that goes beyond the standard capabilities of glTF. For sample models that test/demonstrate specific glTF features and extensions, see glTF 2.0 Sample Models.

The table below shows the most common glTF extensions, and indicates which ones are currently supported by Piglet. For a more comprehensive list of glTF extensions, see the glTF Extension Registry.

This table was last updated in June 2022, for Piglet version 1.3.7.

Extension	Supported?	Notes
KHR_draco_mesh_compression	YES	Requires DracoUnity (see Installing DracoUnity).
KHR_lights_punctual	NO	
KHR_materials_clearcoat	NO	
KHR_materials_ior	NO	
KHR_materials_sheen	NO	
KHR_materials_specular	NO	
KHR_materials_transmission	NO	
KHR_materials_unlit	YES	
KHR_materials_variants	YES	See Selecting Materials Variants and Runtime Materials Variants Tutorial
KHR_materials_volume	NO	
KHR_mesh_quantization	NO	
KHR_texture_basisu	YES	Requires KtxUnity (see Installing KtxUnity).
KHR_texture_transform	YES*	*Only scale and offset transformations are supported (no rotations).
KHR_xmp_json_ld	NO	

Extension	Supported?	Notes
EXT_lights_image_based	NO	
EXT_mesh_gpu_instancing	NO	
EXT_meshopt_compression	NO	
EXT_texture_webp	NO	
KHR_materials_pbrSpecularGlossiness	YES	

Known Issues

- **Runtime imports may stall the main Unity thread**. I have done my best to minimize interruptions to the main Unity thread during runtime glTF imports. However, one significant issue is that Unity uploads textures to the GPU **synchronously**, and this can cause FPS drops when importing high resolution textures at runtime. For now, the best way to work around this issue is to either: (1) convert your textures to KTX2/BasisU format (see Supercompressed Textures (Unity 2019.3+)), or (2) lower the resolution/quality of your textures.
- **Many glTF extensions are not supported (yet!).** I am steadily chipping away at this. See Supported glTF Extensions for details about which extensions are currently supported.
- **Piglet does not create humanoid avatar mappings (yet!)**. For the time being, there is no easy way to retarget humanoid animations (e.g. Mixamo) onto glTF models. This is a highly requested feature and I do plan to implement it.
- **Piglet cannot export glTF files (yet!)**. So far, Piglet does not have any glTF export capabilities, neither in the Editor nor at runtime. This is a highly requested feature and I do plan to implement it.

Installation

To set up Piglet in your project, purchase and install Piglet from the Asset Store page. Piglet works with Unity 2018.4 or later, and does not require installation of any third-party assets/dependencies.

Piglet bundles the following libraries:

Library	Author	License	Path
Newtonsoft.Json-for-Unity	Newtonsoft/jilleJr@github	MIT License	`Assets/Piglet/Depend`
SharpZipLib	icsharpcode@github	MIT License	`Assets/Piglet/Depend`
UnityGLTF	Khronos/Sketchfab	MIT License	`Assets/Piglet/Depend`

Editor Imports

Importing glTF Models into your Unity Project

Once you have installed Piglet from the Unity Asset Store, you can import glTF models into your Unity project by either:

1. Dragging-and-dropping a `.gltf` / `.glb` / `.zip` file from a file browser application (e.g. Windows File Explorer) into the Unity Project Browser (Figure 1), **OR**
2. Saving a `.gltf` / `.glb` / `.zip` file directly into your Assets folder from an external program (e.g. Blender).

For a video demonstration of the drag-and-drop method, see the Editor Import Demo video.

In general, saving glTF files directly into your Assets folder is a more efficient workflow, especially if you are working on a model in Blender and frequently re-exporting it to glTF. However, one important caveat is that the updated glTF file will not be reimported until you switch focus back to Unity (e.g. Alt-Tab). Unfortunately, Unity does not respond to project file changes while it is in the background.

After an Editor glTF import completes, an output folder is created in your project with the same name as the input `.gltf` / `.glb` / `.zip` file, minus the file extension. This folder contains a Unity prefab for the imported model, as well as Unity assets for the individual textures, materials, meshes, and animation clips (Figure 1). To use the imported model in your game, you can simply drag-and-drop the prefab into your Unity scene(s).

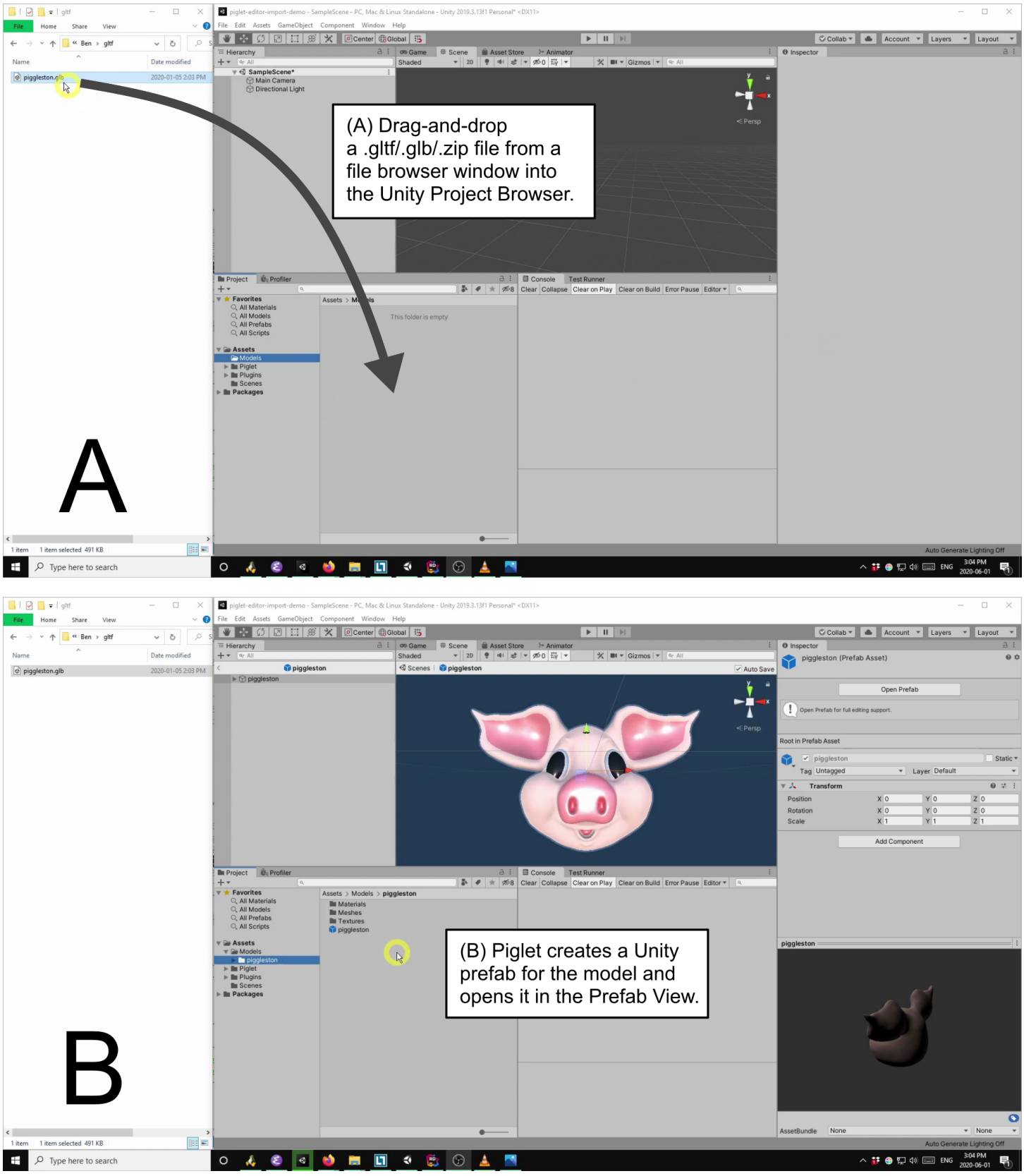


Figure 1: Importing a glTF file via drag-and-drop. (A) The user drags-and-drops a .gltf/.glb/.zip file from Windows File Explorer to the Unity Project Browser window. (B) Piglet creates a Unity prefab for the model and opens it in the Scene View.

There may be circumstances where you want copy a `.gltf` / `.glb` / `.zip` file into your project without automatically converting it to a Unity prefab. You can bypass/disable Piglet's automatic glTF imports by either:

- Holding down the `Control` or `Command` key while dragging-and-dropping the `.gltf` / `.glb` / `.zip` into the Unity Project Browser, **OR**
- Unchecking `Enable glTF imports in Editor` in the Piglet Options window, located under `Window => Piglet Options` in the Unity menu.

Selecting Materials Variants (KHR_materials_variants)

Piglet supports the KHR_materials_variants extension, which allows a glTF file to provide multiple visual styles ("variants") for the same model. The mesh data is shared among all variants, but each variant applies a different set of materials (textures) to the mesh.

Whenever a glTF model uses the KHR_materials_variants extension, the user can switch between the available variants using the drop-down menu on the `MaterialsVariantsSelector` component, attached to the root GameObject of the model (Figure 2). The last item of the drop-down menu is always the special "default" element, which resets all materials to their initial state when the model was first loaded. (The default materials used by a model does not necessarily correspond to any particular variant.)

It is also possible to switch materials variants at runtime. For instructions, please see the Runtime Materials Variants Tutorial.

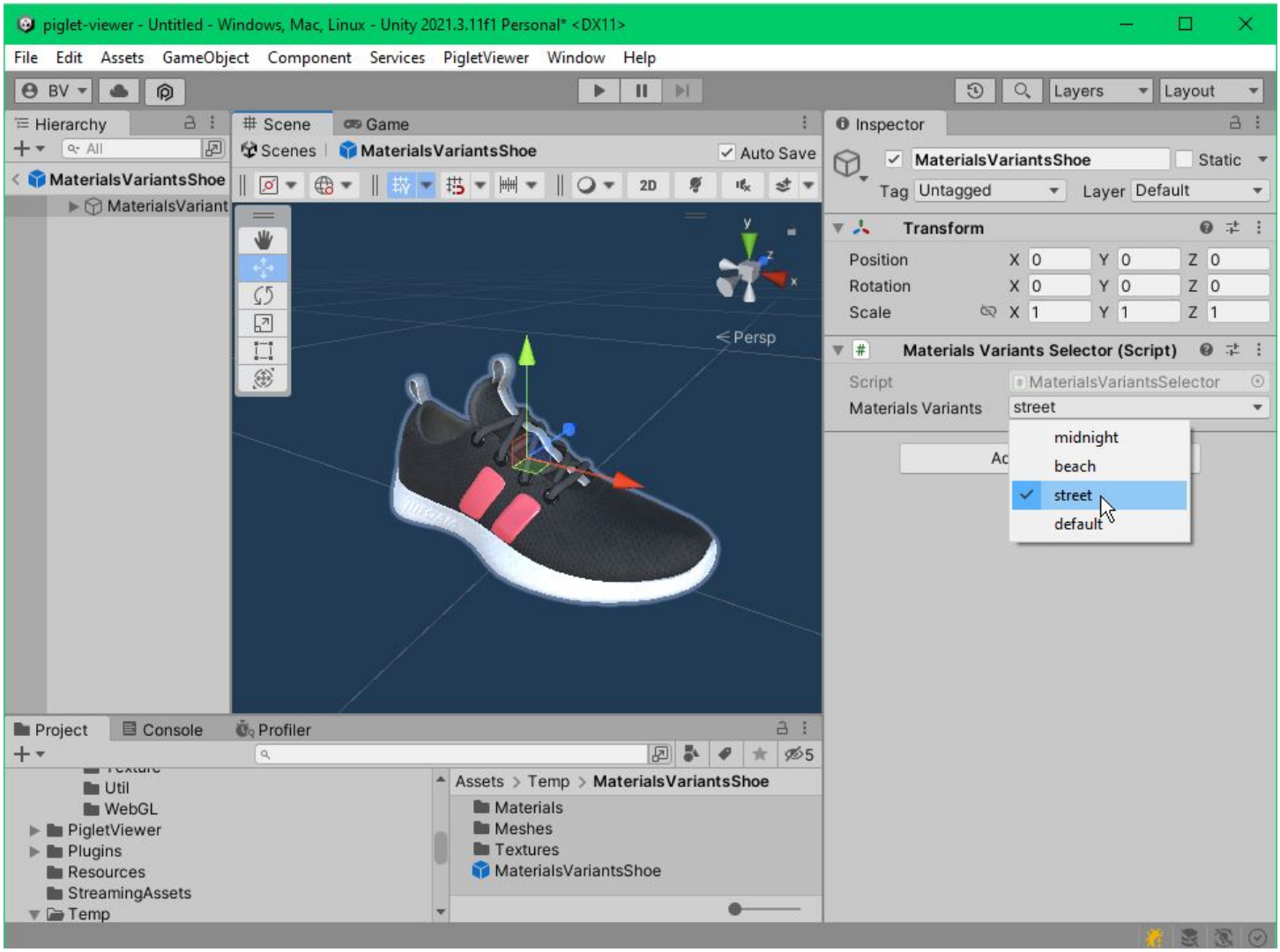


Figure 2: Users can select different materials variants for a model using the drop-down menu on the `MaterialsVariantsSelector` component, on the root GameObject of the model. This component/menu will only be present if the glTF model uses the `KHR_materials_variants` extension.

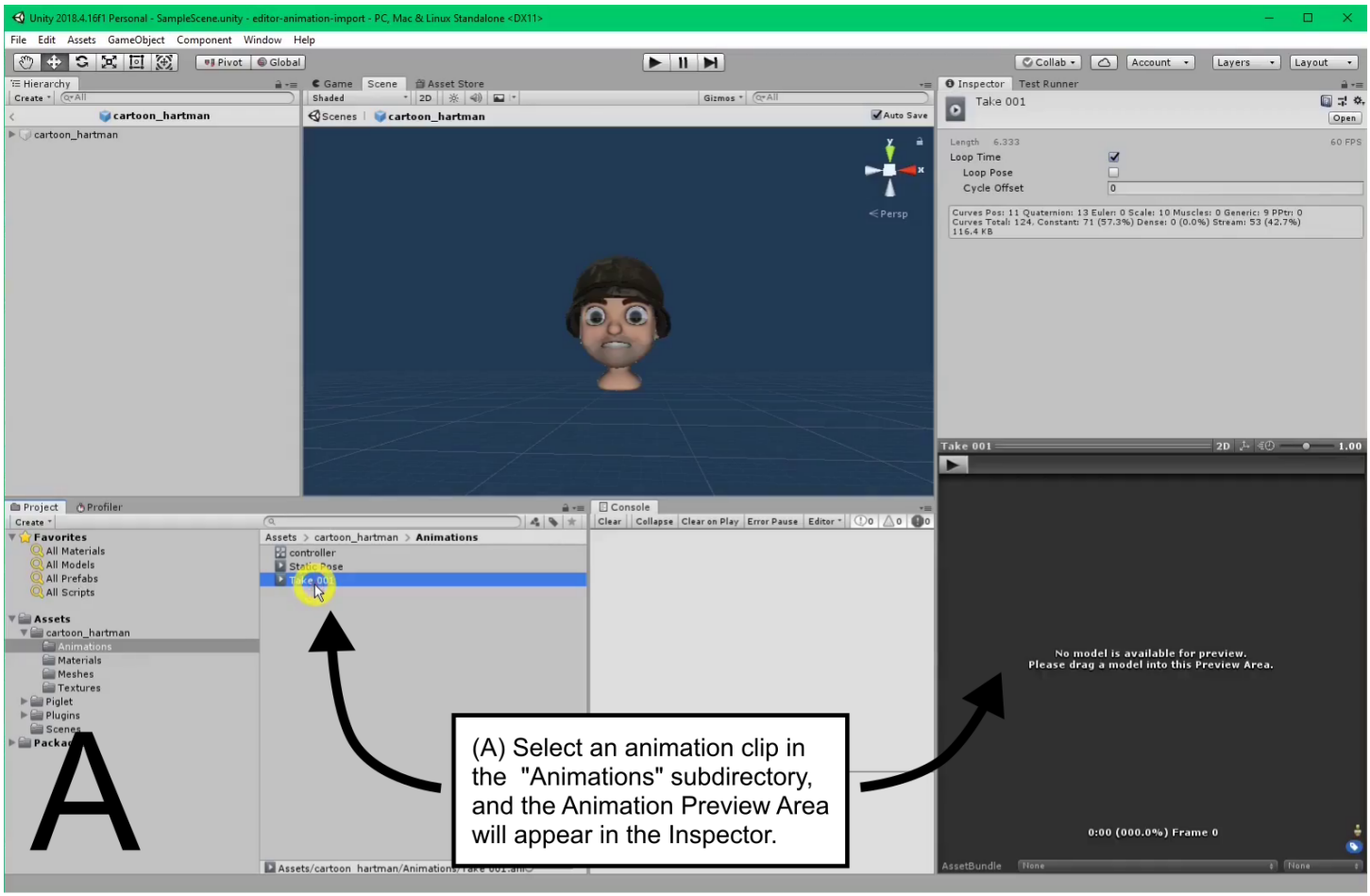
Editor Animation Tutorial

Piglet can import and play animations from glTF files, both in the Editor and at runtime. This section demonstrates how to preview animation clips created during Editor imports, and how to play them back from runtime scripts. For a video version of this section, see the Editor Animation Tutorial video.

Previewing Animations in the Editor

If a glTF file contains animations, Piglet will create an `Animations` subdirectory under the main import directory containing: (1) an `AnimatorController` asset (“controller”) for playing the animations at runtime, (2) a “Static Pose” `AnimationClip` for resetting the model to its default pose, and (3) an `AnimationClip` asset for each animation from the glTF file (Figure 3). The “controller” asset is not needed for previewing animations but is further explained in Playing (Mecanim) Animations at Runtime.

To preview an glTF animation in the Editor, first select the AnimationClip asset in the Project Browser window (Figure 3). This will cause a blank Animation Preview Area to appear in the Inspector window with the message `No model is available for preview. Please drag a model into this Preview Area.` Next, drag the prefab for the imported glTF model (located one level up from the `Animations` folder) onto the Animation Preview Area. You should then be able to click the Play button to view the animation in the Editor.


A
(A) Select an animation clip in the "Animations" subdirectory, and the Animation Preview Area will appear in the Inspector.
No model is available for preview.
Please drag a model into this Preview Area.

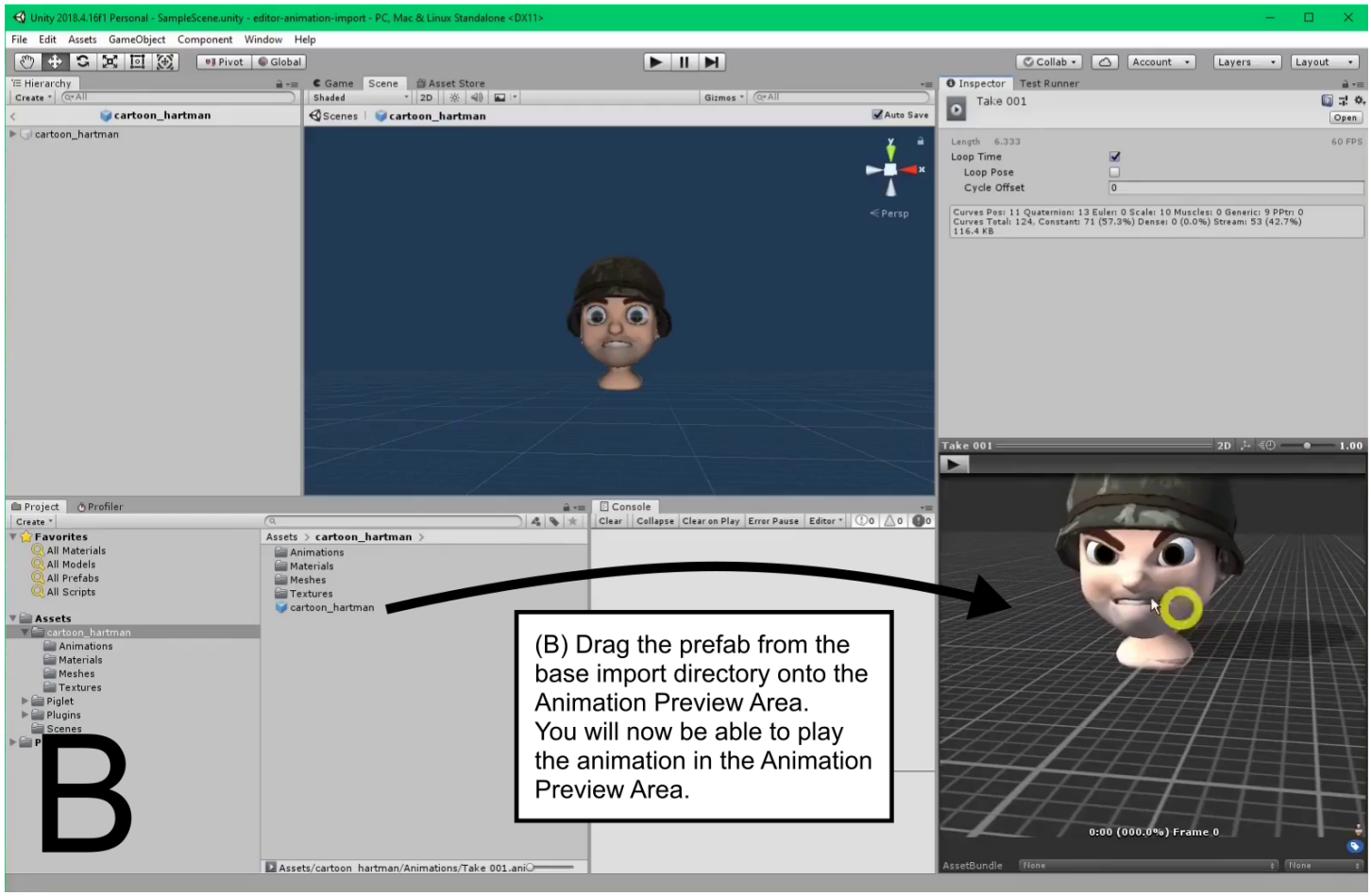

B
(B) Drag the prefab from the base import directory onto the Animation Preview Area.
You will now be able to play the animation in the Animation Preview Area.

Figure 3: Previewing an animation clip in the Editor. (A) The user selects an animation clip in the “Animations” subdirectory, causing the Animation Preview Area to appear in the Inspector (bottom right). (B) The user drags the prefab for the model from the main import directory onto the Animation Preview Area, in order to associate the model with the animation clip. Having established this link, the user is able to preview the animation by clicking the Play button in the Animation Preview Area. **Attribution**: These screenshots depict the “Cartoon Hartman” model by Willy Decarpentrie, skudgee@sketchfab, CC Attribution License.

Playing (Mecanim) Animations at Runtime

This section demonstrates how play animation clip assets from runtime scripts. By default, Editor glTF imports create Mecanim animation clips, and the instructions in this section are specific to Mecanim[4] [5]. For details about playing Legacy animation clips at runtime, see the Runtime Animation Tutorial section of this manual[6].

When a glTF file contains one or more animations, Piglet will attach two additional components to the root `GameObject` of the model: (1) an `Animator` component for controlling playback of the animation clips, and (2) an `AnimationList` component with an ordered list of the animation clips (Figure 4). The `AnimationList` component allows users to access the animation clips by their original index in the glTF file. More importantly, it provides access to the `.name` field of each animation clip, which is needed for play that clip with the `Animator` component.

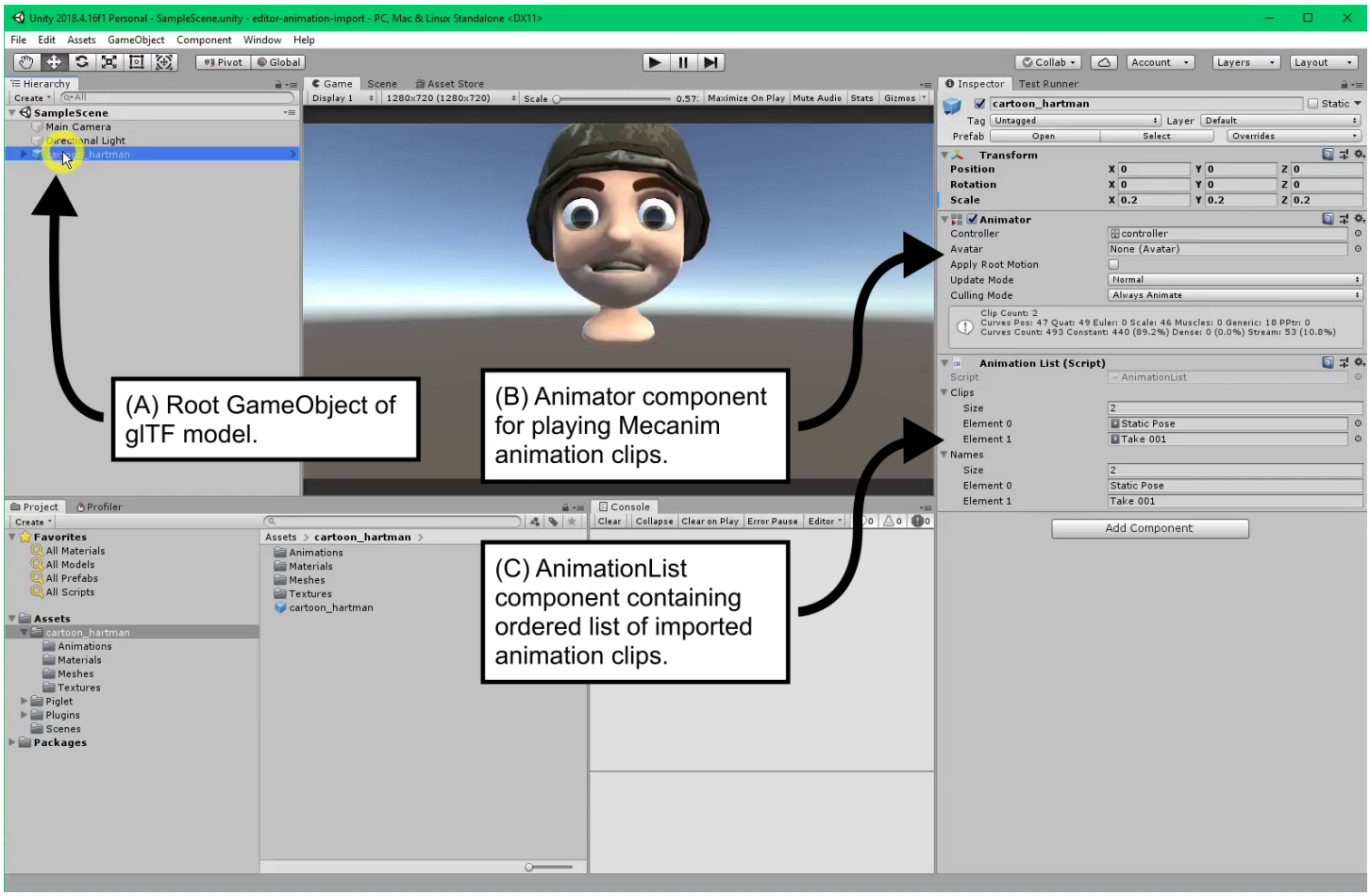


Figure 4: Animation-related components on the root `GameObject` (A) of an Editor-imported model. When a glTF model has one or more animations, Piglet will attach: an `Animator` component for playing the animation clips at runtime (B), and an `AnimationList` component for accessing the animation clips by their original index in the glTF file (C). **Attribution**: These screenshots depict the "Cartoon Hartman" model by Willy Decarpentrie, skudgee@sketchfab, CC Attribution License.

Every `Animator` component depends on a state machine called an `AnimatorController`, that determines which animation clip to play at any given time (Figure 5). In most cases, there is a one-to-one correspondence between `AnimatorController` states and animation clips. In order to start playing a particular clip at runtime, we just need to activate the correct state in the `AnimatorController` and start the `Animator` playing. Both of these tasks are accomplished by calling the `Animator.Play` method.

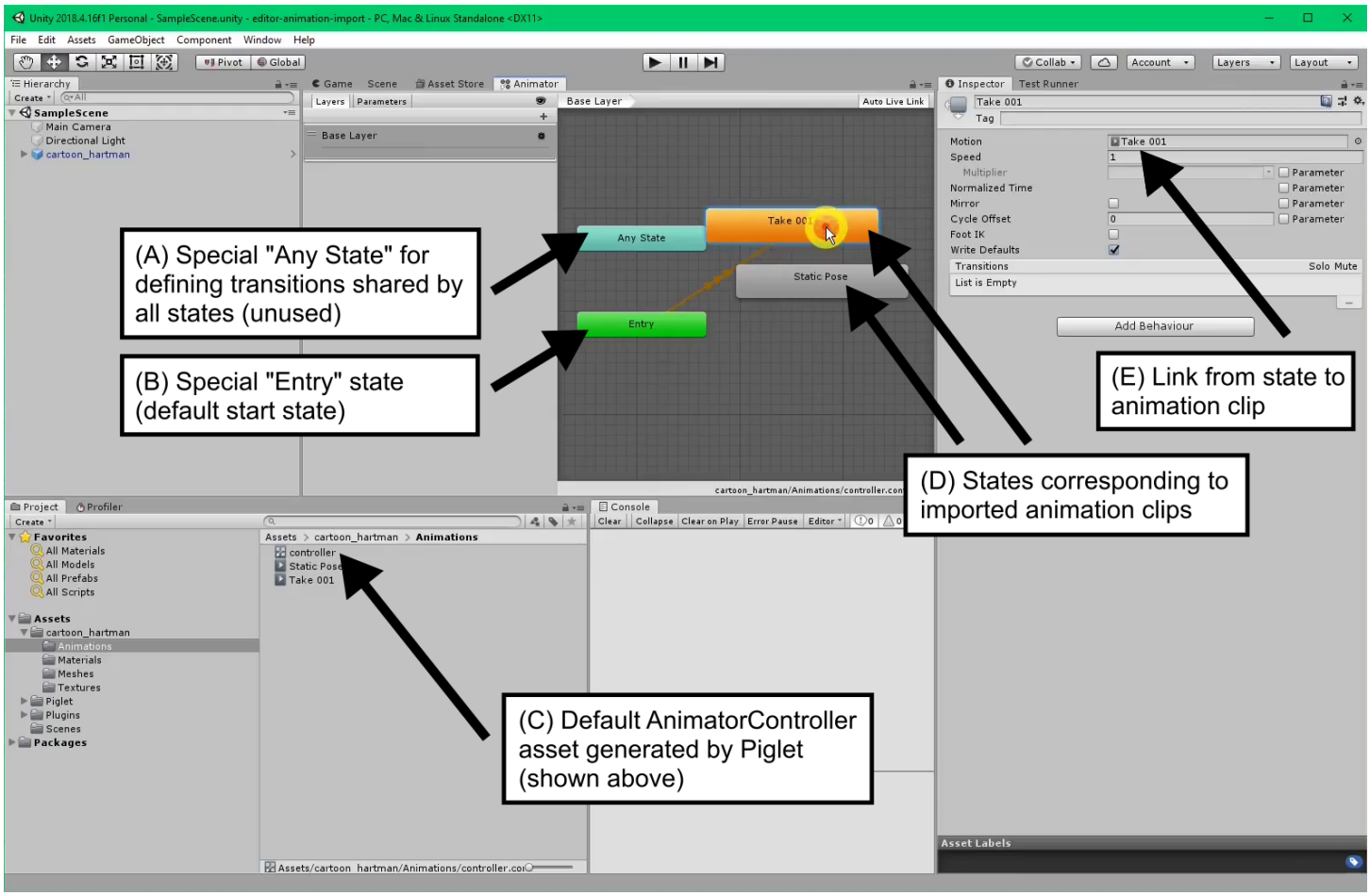


Figure 5: An example `AnimatorController` used for playing Editor-imported animation clips at runtime. An `AnimatorController` is a state machine used by the `Animator` component to determine which animation clip to play at any given time. Piglet creates a default `AnimatorController` asset called “controller” in the `Animations` subdirectory (C). This controller contains one state per animation clip (D) and two special states that are present in every `AnimatorController`: “Any State” (A) and “Entry” (B). For regular controller states, the link between the state and its corresponding animation clip is set by the `Motion` field (E).

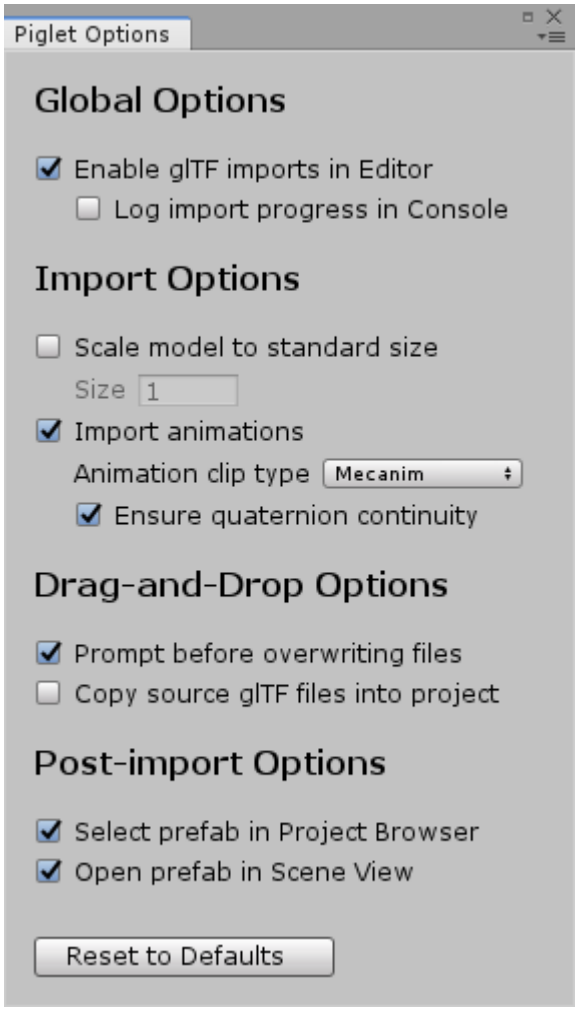
Listing 1 shows an example script that plays a Mecanim animation clip as soon as Unity enters Play Mode. We start the animation by calling the `Animator.Play` method, passing in the initial `AnimatorController` state and layer[7] as arguments. By convention, Piglet uses the `.name` field of an `AnimationClip` for the corresponding state name in the `AnimatorController`, and thus we can get the state name by accessing the target clip by index in the `AnimationList` component. Note that Listing 1 accesses the clip at index 1 in the `AnimationList`, rather than index 0, because index 0 is reserved for the special “Static Pose” clip that resets the model to its default pose[8]. As such, the animations imported from the glTF file always start at index 1. For the layer argument to `Animator.Play`, we simply pass in 0, because the controller generated by Piglet only uses the default layer.

```
using UnityEngine;
using Piglet;

public class AnimationBehaviour : MonoBehaviour
{
    void Start()
    {
        var anim = GetComponent<Animator>();
        var animList = GetComponent<AnimationList>();

        // Note: Imported animation clips always start
        // at index 1, because index "0" is reserved for
        // the "Static Pose" clip.

        var stateName = animList.Clips[1].name;

        // Note: We use 0 for the layer index argument
        // because the AnimatorController generated by
        // Piglet only uses the default layer.

        anim.Play(stateName, 0);
    }
}
```

Listing 1: A minimal script for playing an Editor-imported (Mecanim) animation clip at runtime. When this script is attached to the root `GameObject` of the model, it plays the first animation from the glTF file, immediately after Unity enters Play Mode. Note that the imported animations always begin at index 1 in the `AnimationList`, because index 0 is reserved for the "Static Pose" clip.

Editor Import Options (Piglet Options Window)

A number of options controlling Editor glTF imports can be set in the Piglet Options window, located under `Window => Piglet Options` in the Unity menu.



Global Options

Option	Description
`Enable glTF imports in Editor`	Enable/disable automatic glTF imports when adding new .gltf/.glb/.zip files to the project
`Log import progress in Console`	Log progress messages to Unity Console window during glTF imports (useful for debugging)

Import Options

Option	Description
`Scale model to standard size`	Enable/disable automatic resizing of imported model
`Size`	Target size of model, along its longest dimension

Option	Description
`Import animations`	Enable/disable import of glTF animations as Unity `AnimationClip` assets
`Animation clip type`	“Mecanim” or “Legacy”
`Ensure quaternion continuity`	Enable/disable AnimationClip.EnsureQuaternionContinuity() after importing each animation clip

Drag-and-Drop Options

Option	Description
`Prompt before overwriting files`	Show confirmation prompt if glTF import directory already exists
`Copy source glTF files into project`	Copy dragged-and-dropped glTF file/folder into project before performing glTF import. By default, only the Piglet-generated import directory is added to the project.

Post-import Options

Option	Description
`Select prefab in Project Browser`	After a glTF import has completed, select/highlight the generated prefab in the Project Browser window
`Open prefab in Scene View`	After a glTF import has completed, open the generated prefab in the Scene View tab. (This is equivalent to double-clicking the prefab in the Project Browser.)

Runtime Imports

Piglet can import a glTF model at runtime from a file path, an HTTP URL, or a `byte[]` array containing the raw byte content of a .gltf/.glb/.zip file. Runtime imports are performed incrementally, with minimum possible interruption to the main Unity thread.

Runtime Import Tutorial

This section walks through example code for importing a glTF model at runtime. For a video version of this section, see the Runtime Import Tutorial video. In addition, the example code in this section is included with Piglet under `Assets/Piglet/Examples/RuntimeImport`.

As our example glTF model, we will use the `.glb` file for Sir Piggleston (the Piglet mascot), which may be downloaded from https://awesomesaucelabs.github.io/piglet-webgl-demo/StreamingAssets/piggleston.glb. The minimal code to import the model at runtime is as follows:

```
using Piglet;
using UnityEngine;

/// <summary>
/// This MonoBehaviour provides a minimal example for using
/// Piglet to import glTF models at runtime.
/// </summary>
public class RuntimeImportBehaviour : MonoBehaviour
{
    /// <summary>
    /// The currently running glTF import task.
    /// </summary>
    private GltfImportTask _task;

    /// <summary>
    /// Unity callback that is invoked before the first frame.
    /// Create the glTF import task.
    /// </summary>
```

```
    void Start()
    {
        // Note: To import a local .gltf/.glb/.zip file, you may
        // instead pass an absolute file path to GetImportTask
        // (e.g. "C:/Users/Joe/Desktop/piggleston.glb"), or a byte[]
        // array containing the raw byte content of the file.

        _task = RuntimeGltfImporter.GetImportTask(
                "https://awesomesaucelabs.github.io/piglet-webgl-
         demo/StreamingAssets/piggleston.glb");
    }

    /// <summary>
    /// Unity callback that is invoked after every frame.
    /// Here we call MoveNext() to advance execution
    /// of the glTF import task.
    /// </summary>
    void Update()
    {
        // advance execution of glTF import task
        _task.MoveNext();
    }
}
```

Listing 2: Minimal code to import a glTF file at runtime.

As shown in Listing 2, a runtime glTF import happens in two parts. First, we create an import task by calling `RuntimeGltfImporter.GetImportTask`, passing in the URL of the glTF model as a parameter. To load a local .gltf/.glb/.zip file, we may instead pass `GetImportTask` an absolute file path (e.g. “C:/Users/Joe/Desktop/piggleston.glb”) or a `byte[]` array containing the raw byte content of the file. Second, we advance the execution of the import task by repeatedly calling `MoveNext()` on the task. A convenient place to call `MoveNext()` is in the `Update()` method, which is called by Unity once per frame. Continuing to call `MoveNext()` after the import has completed does no harm.

Attaching the script from Listing 2 to any game object in your Unity scene is sufficient to import a glTF model at runtime. However, in a real game/application, you will probably want tighter integration between your own code and the importer. For example, you may want to show progress messages while the model is loading, or to attach custom MonoBehaviours to the model once it has loaded. To achieve these types of behaviours, `GltfImportTask` provides

callback hooks for: progress messages (`OnProgress`), user cancelation (`OnAborted`), import errors (`OnException`), and successful completion (`OnCompleted`).

As a first example of callback usage, we'll extend the example script from Listing 2 to print progress messages during the glTF import. We can achieve this by assigning a custom method to the `OnProgress` callback for the import task, as shown in Listing 3.

```
using Piglet;
using UnityEngine;

/// <summary>
/// This MonoBehaviour provides a minimal example for using
/// Piglet to import glTF models at runtime.
/// </summary>
public class RuntimeImportBehaviour : MonoBehaviour
{
    /// <summary>
    /// The currently running glTF import task.
    /// </summary>
    private GltfImportTask _task;

    /// <summary>
    /// Unity callback that is invoked before the first frame.
    /// Create the glTF import task.
    /// </summary>
    void Start()
    {
        // Note: To import a local .gltf/.glb/.zip file, you may
        // instead pass an absolute file path to GetImportTask
        // (e.g. "C:/Users/Joe/Desktop/piggleston.glb"), or a byte[]
        // array containing the raw byte content of the file.

        _task = RuntimeGltfImporter.GetImportTask(
                "https://awesomesaucelabs.github.io/piglet-webgl-
         demo/StreamingAssets/piggleston.glb");
        _task.OnProgress = OnProgress;
    }

    /// <summary>
    /// Callback that is invoked by the glTF import task
    /// to report intermediate progress.
```

```
    /// </summary>
    /// <param name="step">
    /// The current step of the glTF import process.  Each step imports
    /// a different type of glTF entity (e.g. textures, materials).
    /// </param>
    /// <param name="completed">
    /// The number of glTF entities (e.g. textures, materials) that have been
    /// successfully imported for the current import step.
    /// </param>
    /// <param name="total">
    /// The total number of glTF entities (e.g. textures, materials) that will
    /// be imported for the current import step.
    /// </param>
    private void OnProgress(GltfImportStep step, int completed, int total)
    {
        Debug.LogFormat("{0}: {1}/{2}", step, completed, total);
    }

    /// <summary>
    /// Unity callback that is invoked after every frame.
    /// Here we call MoveNext() to advance execution
    /// of the glTF import task.
    /// </summary>
    void Update()
    {
        // advance execution of glTF import task
        _task.MoveNext();
    }
}
```

Listing 3: An extension of the runtime import script from Listing 2 that prints progress messages to the Unity console. In comparison to Listing 2, the new parts of the code are the `OnProgress` method and the assignment of `OnProgress` to `_task.OnProgress` in `Start`.

Another important use of callbacks is to run custom code after a glTF import has successfully completed. For example, you might want to automatically resize the model, parent the model to another game object, or attach a custom `MonoBehaviour` to the model. These types of tasks can be accomplished using the `OnCompleted` callback. To demonstrate, the example script in Listing 4 uses the `OnCompleted` callback to obtain a reference to the imported model, then

uses that reference to continually spin the model about the y-axis as if it were on a record turntable.

The example in Listing 4 marks the end of this tutorial. Good luck and happy coding!

```
using Piglet;
using UnityEngine;

/// <summary>
/// This MonoBehaviour provides a minimal example for using
/// Piglet to import glTF models at runtime.
/// </summary>
public class RuntimeImportBehaviour : MonoBehaviour
{
    /// <summary>
    /// The currently running glTF import task.
    /// </summary>
    private GltfImportTask _task;

    /// <summary>
    /// Root GameObject of the imported glTF model.
    /// </summary>
    private GameObject _model;

    /// <summary>
    /// Unity callback that is invoked before the first frame.
    /// Create the glTF import task.
    /// </summary>
    void Start()
    {
        // Note: To import a local .gltf/.glb/.zip file, you may
        // instead pass an absolute file path to GetImportTask
        // (e.g. "C:/Users/Joe/Desktop/piggleston.glb"), or a byte[]
        // array containing the raw byte content of the file.

        _task = RuntimeGltfImporter.GetImportTask(
            "https://awesomesaucelabs.github.io/piglet-webgl-
         demo/StreamingAssets/piggleston.glb");
        _task.OnCompleted = OnComplete;
    }
```

```
    /// <summary>
    /// Callback that is invoked by the glTF import task
    /// after it has successfully completed.
    /// </summary>
    /// <param name="importedModel">
    /// the root GameObject of the imported glTF model
    /// </param>
    private void OnComplete(GameObject importedModel)
    {
        _model = importedModel;
        Debug.Log("Success!");
    }

    /// <summary>
    /// Unity callback that is invoked after every frame.
    /// Here we call MoveNext() to advance execution
    /// of the glTF import task.
    /// </summary>
    void Update()
    {
        // advance execution of glTF import task
        _task.MoveNext();

        // spin model about y-axis
        if (_model != null)
            _model.transform.Rotate(0, 1, 0);
    }
}
```

Listing 4: An extension of the runtime import script from Listing 2 that spins the imported model about the y-axis. In comparison to Listing 2, the new parts of the code are the `OnComplete` method, the assignment of `OnComplete` to `_task.OnCompleted` in `Start`, and the call to `_model.transform.Rotate` in `Update`.

Runtime Materials Variants Tutorial (KHR_materials_variants)

Piglet supports the KHR_materials_variants extension, which allows a glTF file to provide multiple visual styles (“variants”) for the same model. The mesh data is shared among all

variants, but each variant applies a different set of materials (textures) to the mesh.

Piglet provides an example scene under `Assets/Piglet/Examples/RuntimeMaterialsVariants`, which demonstrates how to query and select materials variants at runtime. The example allows the user switch between variants of the MaterialsVariantsShoe model, by clicking the buttons along the top of the window (Figure 6).

Figure 6: The `RuntimeMaterialsVariants` example scene, which demonstrates how to query and select materials variants at runtime. The example allows the user to select variants of the `MaterialsVariantsShoe` model by clicking the buttons along the top of the window.

The code for the example scene is reproduced in Listing 5. (This code is highly similar to the examples from the Runtime Import Tutorial section, so the explanations provided in that section may also be helpful.) After importing the model, we get a reference to the `MaterialsVariantsSelector` component on the root GameObject, from inside the `OnCompleted` callback. The `MaterialsVariantsSelector` component is automatically attached to any glTF model that uses the `KHR_materials_variants` extension, and will be absent otherwise.

The `MaterialsVariantsSelector` component has two public fields, `VariantNames` and `VariantIndex`, which can be used for querying and setting the materials variants,

respectively. `VariantNames` is an ordered list of human-readable names for each variant, where the position of each variant in the list corresponds to its index in the glTF file. To select a particular variant, we simply assign its index to the `VariantIndex` field. The call to `GUI.Toolbar` in `OnGUI` handles drawing a button for each element of the `VariantNames` array, and updating value of `VariantIndex` whenever one of the buttons gets clicked.

The reader may wonder which materials variant is selected by default, when the model is first loaded. The answer is “none”! Every model has a default set of materials before any variant is selected, and this default set of materials does not necessarily correspond to any particular variant. (In the case of the example shoe model, the default materials exactly match the “midnight” variant.) In order to reset the model to its default state, the user can call the `ResetMaterials` method of the `MaterialsVariantsSelector` component, or equivalently, they may assign a value of `VariantNames.Length - 1` to `VariantIndex`. The last element of the `VariantNames` array is always the special “default” element, that resets the model to its default materials. In Listing 5, there is no need to explicitly call `ResetMaterials` because `GUI.Toolbar` creates a button for the “default” element.

```
using Piglet;
using UnityEngine;

/// <summary>
/// This MonoBehaviour provides a minimal example of switching
/// materials variants at runtime, for glTF models that use the
/// `KHR_materials_variants` extension.
/// </summary>
public class RuntimeMaterialsVariantsBehaviour : MonoBehaviour
{
    /// <summary>
    /// The currently running glTF import task.
    /// </summary>
    private GltfImportTask _task;

    /// <summary>
    /// Root GameObject of the imported glTF model.
    /// </summary>
    private GameObject _model;

    /// <summary>
    /// MonoBehaviour on the root GameObject of the imported
    /// model, which allows the user to select the active materials
```

```
    /// variant.
    /// </summary>
    private MaterialsVariantsSelector _variantsSelector;

    /// <summary>
    /// Unity callback that is invoked before the first frame.
    /// Create the glTF import task and register a callback
    /// method to be invoked when the glTF import completes.
    /// </summary>
    void Start()
    {
        // The default size of the shoe model is too small.
        //
        // Uniformly scale the model such that the longest
        // dimension of its world-space axis-aligned bounding
        // box becomes 4.0 units.

        var importOptions = new GltfImportOptions();
        importOptions.AutoScale = true;
        importOptions.AutoScaleSize = 4.0f;

        // Note: To import a local .gltf/.glb/.zip file, you may
        // instead pass an absolute file path to GetImportTask
        // (e.g. "C:/Users/Joe/Desktop/piggleston.glb"), or a byte[]
        // array containing the raw byte content of the file.

        _task = RuntimeGltfImporter.GetImportTask(
            "https://awesomesaucelabs.github.io/piglet-webgl-
         demo/StreamingAssets/shoe.glb",
            importOptions);

        // Method to be invoked when the glTF import successfully
        // completes.

        _task.OnCompleted = OnComplete;
    }

    /// <summary>
    /// Callback that is invoked by the glTF import task
    /// after it has successfully completed.
    /// </summary>
    /// <param name="importedModel">
```

```
    /// the root GameObject of the imported glTF model
    /// </param>
    private void OnComplete(GameObject importedModel)
    {
        _model = importedModel;
        _variantsSelector = _model.GetComponent<MaterialsVariantsSelector>();
    }

    /// <summary>
    /// Unity callback that is invoked after every frame.
    /// Here we call MoveNext() to advance execution
    /// of the glTF import task. Once the model has been successfully
    /// imported, we auto-spin the model about the y-axis.
    /// </summary>
    void Update()
    {
        // advance execution of glTF import task
        _task.MoveNext();

        // spin model about y-axis
        if (_model != null)
            _model.transform.Rotate(0.0f, 0.25f, 0.0f);
    }

    void OnGUI()
    {
        // Add some buttons along the top of the screen, which allow
        // the user to select the active materials variant.
        //
        // Note: `_variantsSelector` will be null until the model has
        // been successfully imported.

        if (_variantsSelector != null)
        {
            _variantsSelector.VariantIndex = GUI.Toolbar(
                new Rect(25, 25, 500, 30),
                _variantsSelector.VariantIndex,
                _variantsSelector.VariantNames);
        }
```

```
    }
}
```

Listing 5: Example code for importing a glTF model at runtime and selecting materials variants. This code is included with Piglet under `Assets/Piglet/Examples/RuntimeMaterialsVariants`.

Runtime Animation Tutorial

This section demonstrates how to import and play animations from a glTF file at runtime. For a video version of this section, see the Runtime Animation Tutorial video.

Runtime glTF imports always use the Legacy animation system, because Mecanim cannot create animation clips at runtime[9] (as of December 2020). In practice, I have not found this to be an issue – for simple playback of glTF animations, the Legacy system works very well.

When Piglet imports a glTF model with one or more animations at runtime, it attaches two additional components to the root `GameObject` of the model: (1) an `Animation` component for controlling playback of the animation clips, and (2) an `AnimationList` component containing an ordered list of the animation clips (Figure 7). The `AnimationList` component allows users to access the imported animation clips by their original index in the glTF file. More importantly, it provides access to the `.name` field of each animation clip, which is needed for playing the clip with the `Animation` component.

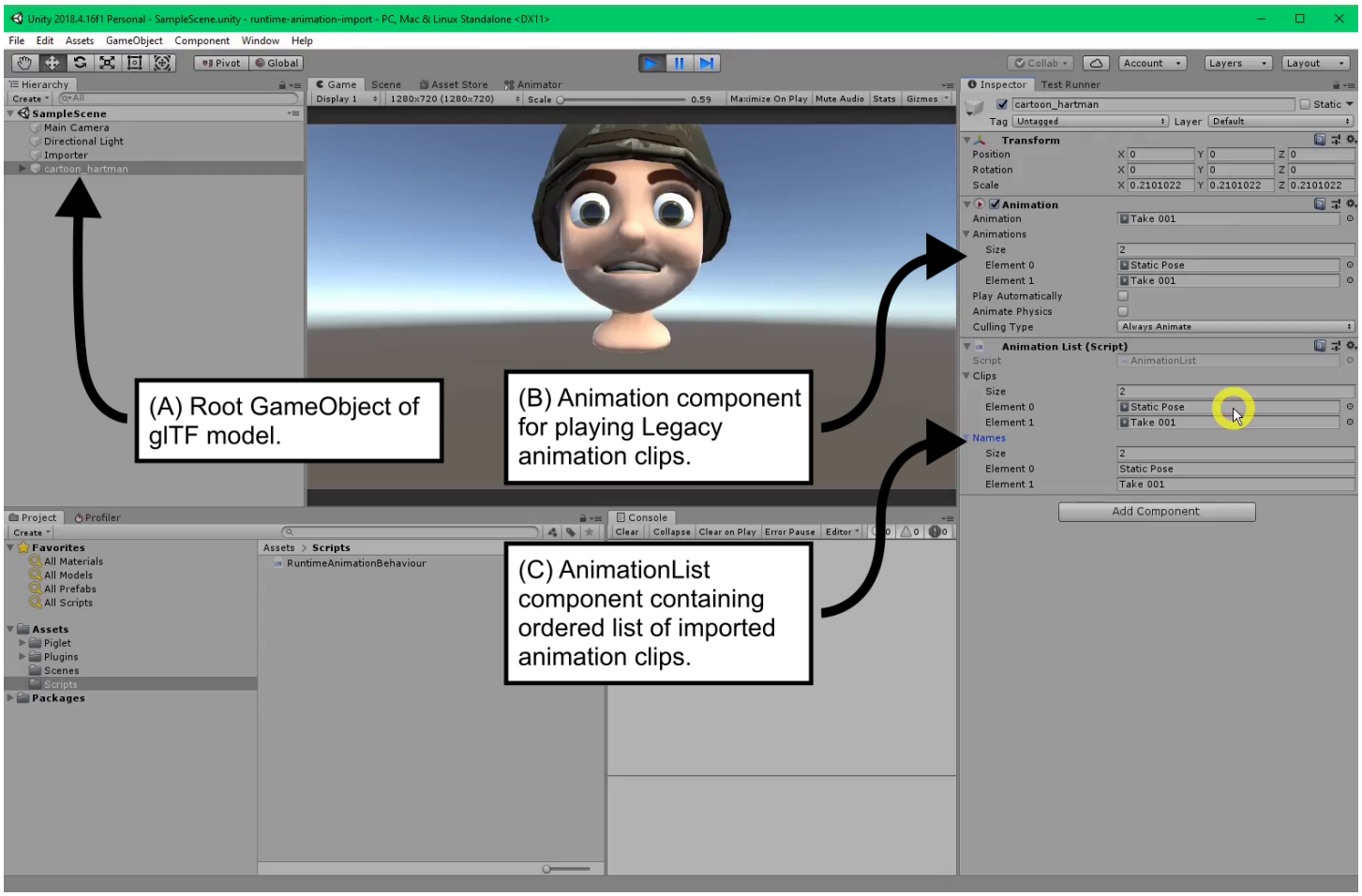


Figure 7: Animation-related components on the root `GameObject` (A) of a runtime-imported model. When a glTF model has one or more animations, Piglet will attach: an `Animation` component for playing the animation clips (B), and an `AnimationList` component for accessing the animation clips by their original index in the glTF file (C). **Attribution**: These screenshots depict the "Cartoon Hartman" model by Willy Decarpentrie, skudgee@sketchfab, CC Attribution License.

Listing 6 shows an example script that imports a glTF model with an animation at runtime, and then immediately plays the animation. The basic steps for importing an animated glTF model are the same as for static models: (1) create a `GltfImportTask` in the `Start` method, and (2) advance execution of the task by calling `GltfImportTask.MoveNext` in `Update`.

To play the animation after the model has finished loading, we assign the `OnComplete` method to `_task.OnCompleted` in `Start`. Piglet passes the root `GameObject` of the imported model as an argument to `OnComplete`, which we then use to obtain a reference to the `Animation` component for playing the animation. Since an `Animation` component can hold any number of animation clips, we need to provide a string-based key to `Animation.Play` that identifies the clip to play. By convention, Piglet uses the `.name` field of each animation clip as its key, and

so we can obtain the desired key by accessing the animation clip (by index) from the `AnimationList` component. Note that the animation clips imported from glTF file always begin at index 1 of the `AnimationList`, because index 0 is reserved for the “Static Pose” clip.

```
using Piglet;
using UnityEngine;

/// <summary>
/// This MonoBehaviour provides a minimal example for
/// importing and playing glTF animations at runtime.
/// </summary>
public class RuntimeAnimationBehaviour : MonoBehaviour
{
    /// <summary>
    /// The currently running glTF import task.
    /// </summary>
    private GltfImportTask _task;

    /// <summary>
    /// Unity callback that is invoked before the first frame.
    /// Create the glTF import task and set up callback for
    /// successful completion.
    /// </summary>
    void Start()
    {
        // Uniformly scale the model such that the longest
        // dimension of its world-space axis-aligned bounding
        // box becomes 4.0 units.
        var importOptions = new GltfImportOptions();
        importOptions.AutoScale = true;
        importOptions.AutoScaleSize = 4.0f;

        // Note: To import a local .gltf/.glb/.zip file, you may
        // instead pass an absolute file path to GetImportTask
        // (e.g. "C:/Users/Joe/Desktop/piggleston.glb"), or a byte[]
        // array containing the raw byte content of the file.
        _task = RuntimeGltfImporter.GetImportTask(
            "https://awesomesaucelabs.github.io/piglet-webgl-
         demo/StreamingAssets/cartoon_hartman.zip",
            importOptions);
```

```
        _task.OnCompleted = OnComplete;
    }

    /// <summary>
    /// Callback that is invoked by the glTF import task
    /// after it has successfully completed.
    /// </summary>
    /// <param name="importedModel">
    /// the root GameObject of the imported glTF model
    /// </param>
    private void OnComplete(GameObject importedModel)
    {
        var anim = importedModel.GetComponent<Animation>();
        var animList = importedModel.GetComponent<AnimationList>();

        // Note: Imported animation clips always start
        // at index 1, because index "0" is reserved for
        // the "Static Pose" clip.
        var clipKey = animList.Clips[1].name;
        anim.Play(clipKey);

        Debug.Log("Success!");
    }

    /// <summary>
    /// Unity callback that is invoked after every frame.
    /// Here we call MoveNext() to advance execution
    /// of the glTF import task.
    /// </summary>
    void Update()
    {
        // advance execution of glTF import task
        _task.MoveNext();
    }
}
```

Listing 6: An example script that performs runtime import of a glTF model with an animation, then immediately plays that animation. **Attribution**: This script uses the "Cartoon Hartman" model by Willy Decarpentrie, skudgee@sketchfab, CC Attribution License.

Runtime Import API

Overview

In Piglet, a runtime glTF import is accomplished by the following steps:

1. Create a `GltfImportTask` by calling `RuntimeGltfImporter.GetImportTask`, passing in the file path, URL, or raw byte content of the input `.gltf`/`.glb`/`.zip` file as a parameter. See Creating a GltfImportTask for details.
2. Configure callbacks on the `GltfImportTask` to execute custom code for success/failure/progress events (optional). See Configuring Callbacks on a GltfImportTask for details.
3. Execute the `GltfImportTask` by calling `MoveNext()` until the import has completed. See Executing a GltfImportTask for details.

For concrete code examples demonstrating the above steps, see the Runtime Import Tutorial.

Creating a GltfImportTask

`RuntimeGltfImporter` provides the following static methods for creating a `GltfImportTask`:

RuntimeGltfImporter Methods

Method	Return Type	Description
`GetImportTask(string uri, GltfImportOptions options=null)`	`GltfImportTask`	Create an import task that imports the glTF model from `uri`, where `uri` is an absolute file path, HTTP(S) URL, or Android content URI that points to a `.gltf`/`.glb`/`.zip` file.
`GetImportTask(Uri uri, GltfImportOptions options=null)`	`GltfImportTask`	Create an import task that imports the glTF model

Method	Return Type	Description
		from `uri`, where `uri` is an absolute file path, HTTP(S) URL, or Android content URI that points to a `.gltf` / `.glb` / `.zip` file.
`GetImportTask(byte[] data, GltfImportOptions options=null)`	`GltfImportTask`	Create an import task that imports from the raw byte content of a `.gltf` / `.glb` / `.zip` file.

All versions of `GetImportTask` accept an instance of `GltfImportOptions` as an optional second argument. See Runtime Import Options for a description of the available options.

Configuring Callbacks on a GltfImportTask

You can assign callback methods to delegate members of a `GltfImportTask`, in order to run custom code for success/failure/progress events. For example, callbacks can be used to position a successfully imported model within a scene or to attach a custom `MonoBehaviour`.

GltfImportTask Delegate Members

Callback	Description
`OnProgress`	Invoked at regular intervals to report progress of `GltfImportTask`
`OnAborted`	Invoked when `Abort()` is called on `GltfImportTask`
`OnException`	Invoked when `GltfImportTask` throws an exception (e.g. file not found)
`OnCompleted`	Invoked when `GltfImportTask` completes successfully

Executing a GltfImportTask

`GltfImportTask` provides the following methods for controlling its own execution:

GltfImportTask Methods

Method	Description
`MoveNext()`	Advance execution of the import task by a small increment. This method should be called repeatedly until the import task has completed.
`Abort()`	Abort the import task. This method should typically be called in response to a user action, such as pressing a “Cancel” button.

Runtime Import Options

Piglet supports a number of options for controlling runtime glTF imports. To configure these options, users may pass a `GltfImportOptions` object as an optional second argument to `RuntimeGltfImporter.GetImportTask`. (See Creating a GltfImportTask.)

For a concrete example of `GltfImportOptions` usage, see Listing 6 from the Runtime Animation Tutorial section.

Currently, `GltfImportOptions` provides the following options:

Option	Default	Description
`ShowModelAfterImport`	`true`	Automatically unhide the model after a successful glTF import, by calling `SetActive(true)` on the root GameObject. Users might want to keep the model hidder until they have completed their own post-processing on the model (e.g. adding colliders).
`AutoScale`	`false`	Automatically resize the model after a successful glTF import
`AutoScaleSize`	`1.0`	Target size of model along its longest dimension

Option	Default	Description
`ImportAnimations`	`true`	Import glTF animations as Legacy animatio clips
`EnsureQuaternionContinuity`	`true`	Call AnimationClip.EnsureQuaternionContinuity after importing each animation clip.
`CreateMipmaps`	`false`	Create mipmaps for PNG/JPG textures during runtime glTF imports. This option is false by default because it roughly doubles the texture import time and increases the probability of FPS drops. This option has nc effect on the creation of mipmaps for KTX2/BasisU textures, since mipmaps are always created in that case without any additional overhead. Likewise, mipmaps ar always created during Editor glTF imports for all texture formats (PNG, JPG, KTX2/BasisU).
`ZipPassword`	`null`	The password used to unpack encrypted `.zip` files.

Optimizing glTF Files

Supercompressed Textures (Unity 2019.3+)

Overview

Piglet can load glTF files that contain *supercompressed textures*[10] in KTX2/ETC1S or KTX2/UASTC format[11] (i.e. Basis Universal texture formats in a KTX 2.0 container). The main advantages of supercompressed textures are that: (1) textures load faster, and (2) glTF files are smaller. The main disadvantage is loss of image quality, which can give textures a "blocky" appearance. Depending on your application, the loss of image quality may not be noticeable, whereas the performance benefits are usually significant.

For a practical demonstration of the trade-offs, compare the ordinary and KTX2/ETC1S versions of the sample models at the Piglet Web Demo. Note the differences in appearance, loading times, and file sizes.

Before Piglet can load glTF files with supercompressed textures, you will need to install a third-party package called KtxUnity into your Unity project (see Installing KtxUnity). If you attempt to load a glTF file that contains supercompressed textures without installing KtxUnity, the textures will simply load as solid white, and Piglet will issue warnings on the Unity Console about failing to load the textures.

Finally, if you plan to use supercompressed textures with your models, you will probably need to preprocess the glTF files yourself. At the time of writing (Feb 2021), most glTF files on the web use PNG or JPEG textures, including the glTF files downloaded from Sketchfab. I recommend using the gltf-transform command line tool to compress the textures in your glTF files, as described in Supercompressing Your glTF Textures.

Installing KtxUnity

To load glTF files with supercompressed textures, you will need to install KtxUnity via the Unity Package Manager. Please use the following table to determine the KtxUnity versions that are compatible/recommended for your version of Unity, before proceeding with the installation instructions below. (Note: This table was last updated in July 2023.)

Unity Version	**Compatible KtxUnity Versions**	**Recommended KtxUnity Version**
2019.2 or older	*not supported*	*not supported*
2019.3 through 2021.1	0.9.1 through 1.1.2	1.1.2

Unity Version	Compatible KtxUnity Versions	Recommended KtxUnity Version
2021.2 or newer	2.0.0 or newer	2.2.3

Since KtxUnity is hosted by a third-party package registry (OpenUPM), you will need to tell Unity where to download the package by adding a scoped registry to the `Packages/manifest.json` file under your Unity project directory. You can do that by making the edits shown in Listing 7 and then restarting Unity. In addition, remember to change the KtxUnity version in Listing 7 to your required KtxUnity version. If you want to perform the same edits in an automated fashion, you can instead install the OpenUPM CLI tool and run `openupm add com.atteneder.ktx@1.1.2` (or similar).

Note!: I don't recommend using the "Installer Package" link from the KtxUnity README.md, since it is just a more convoluted and fragile method for performing the text edits shown in Listing 7. While the installer link is more automatic, it prevents the user from understanding what is really going on under the hood. Another advantage of manually editing `manifest.json` is that you can pin KtxUnity to a specific version (if desired).

In addition, please note the following "gotcha" when installing KtxUnity:

- When building for the PC/Standalone platform, remember to change `Architecture` from `x86` to `x86_64`. Otherwise, the native DLL for KtxUnity (`ktx_unity.dll`) will not be included in the build, and you may get a `DllNotFoundException` when you run your application.

```
{
  "dependencies" : {
    "com.unity.collab-proxy" : "1.2.16",
    "com.unity.ide.rider" : "1.1.4",
    "com.unity.ide.vscode" : "1.2.0",
    "com.unity.test-framework" : "1.1.13",
    "com.unity.textmeshpro" : "2.0.1",
    "com.unity.timeline" : "1.2.14",
    "com.unity.ugui" : "1.0.0",
    "com.unity.modules.ai" : "1.0.0",
    "com.unity.modules.androidjni" : "1.0.0",
    "com.unity.modules.animation" : "1.0.0",
    "com.unity.modules.assetbundle" : "1.0.0",
```

```
    "com.unity.modules.audio" : "1.0.0",
    "com.unity.modules.cloth" : "1.0.0",
    "com.unity.modules.director" : "1.0.0",
    "com.unity.modules.imageconversion" : "1.0.0",
    "com.unity.modules.imgui" : "1.0.0",
    "com.unity.modules.jsonserialize" : "1.0.0",
    "com.unity.modules.particlesystem" : "1.0.0",
    "com.unity.modules.physics" : "1.0.0",
    "com.unity.modules.physics2d" : "1.0.0",
    "com.unity.modules.screencapture" : "1.0.0",
    "com.unity.modules.terrain" : "1.0.0",
    "com.unity.modules.terrainphysics" : "1.0.0",
    "com.unity.modules.tilemap" : "1.0.0",
    "com.unity.modules.ui" : "1.0.0",
    "com.unity.modules.uielements" : "1.0.0",
    "com.unity.modules.umbra" : "1.0.0",
    "com.unity.modules.unityanalytics" : "1.0.0",
    "com.unity.modules.unitywebrequest" : "1.0.0",
    "com.unity.modules.unitywebrequestassetbundle" : "1.0.0",
    "com.unity.modules.unitywebrequestaudio" : "1.0.0",
    "com.unity.modules.unitywebrequesttexture" : "1.0.0",
    "com.unity.modules.unitywebrequestwww" : "1.0.0",
    "com.unity.modules.vehicles" : "1.0.0",
    "com.unity.modules.video" : "1.0.0",
    "com.unity.modules.vr" : "1.0.0",
    "com.unity.modules.wind" : "1.0.0",
    "com.unity.modules.xr" : "1.0.0",
    "com.atteneder.ktx" : "1.1.2"
  },
  "scopedRegistries" : [
    {
      "name" : "OpenUPM",
      "url" : "https://package.openupm.com",
      "scopes" : [
        "com.atteneder"
      ]
    }
  ]
}
```

Listing 7: Example edits to `Packages/manifest.json` in order to install KtxUnity. After adding the highlighted text, restart Unity to install the package.

Supercompressing Your glTF Textures

Most glTF files store their textures in PNG or JPEG format. To convert the textures in your glTF files to KTX2/ETC1S or KTX2/UASTC, I recommend using the gltf-transform command line tool.

In order to supercompress your textures with `gltf-transform`, you will first need to install:

- NodeJS and NPM
- KTX-Software. This project provides the `toktx` program that is invoked by `gltf-transform`. To install the software, go to the GitHub releases page, click/expand "Assets" at the bottom of the release notes, and choose the appropriate package/installer for your O/S. Note that if you are using Windows, you will also need to add `C:\Program Files\KTX-Software\bin` to your `Path`, so that `gltf-transform` can find the `toktx` binary.

Once you have installed the above prerequisites, you will be able to install `gltf-transform` by running:

```
npm install --global @gltf-transform/cli
```

You will then be able to convert the textures in a glTF file to KTX2/ETC1S by running:

```
gltf-transform etc1s input.glb output.glb
```

or to KTX2/UASTC by running:

```
gltf-transform uastc input.glb output.glb
```

The `gltf-transform` program provides options for restricting the conversion to specific textures ("slots"), adjusting quality settings, and more. See `gltf-transform etc1s --help` or `gltf-transform uastc --help` for further details.

Draco Mesh Compression (Unity 2019.3+)

Overview

Piglet can load glTF files that use *Draco mesh compression*[1][2]. The main benefit of using Draco compression is that it can substantially reduce the size of your glTF files (e.g. 20% of original size), especially if your model contains large/complex meshes. While using Draco compression does introduce some computational overhead, the impact on model loading times is usually neglible.

For a practical demonstration of the benefits, compare the ordinary and Draco-compressed versions of the models at the Piglet Web Demo. Note the large differences in file sizes and the small differences in loading times. For further examples of real-world Draco compression results, see Draco Compressed Meshes with glTF and 3D Tiles.

Before Piglet can load Draco-compressed glTF files, you will need to install a third-party package called DracoUnity into your Unity project (see Installing DracoUnity). If you attempt to load a Draco-compressed glTF file without installing DracoUnity, the glTF import will fail with an error in the Unity console.

Finally, if you plan to use Draco-compressed meshes for your models, you will probably need to preprocess the glTF files yourself. At the time of writing (May 2021), most glTF files available on the web use uncompressed meshes, including the glTF files downloaded from Sketchfab. I recommend using the gltf-transform command line tool to compress the meshes in your glTF files, as detailed in Draco-compressing Your glTF Meshes.

Installing DracoUnity

To load glTF files that use Draco mesh compression, you will need to install DracoUnity via the Unity Package Manager. Please use the following table to determine the DracoUnity versions that are compatible/recommended for your version of Unity, before proceeding with the installation instructions below. (Note: This table was last updated in July 2023.)

Unity Version	Compatible DracoUnity Versions	Recommended DracoUnity Version
2019.2 or older	*not supported*	*not supported*
2019.3 through 2021.1	1.1.0 through 3.3.2	3.3.2
2021.2 or newer	4.0.0 or newer	4.1.0

Since DracoUnity is hosted by a third-party package registry (OpenUPM), you will need to tell Unity where to download the package by adding a scoped registry to the `Packages/manifest.json` under your Unity project directory. You can do that by making the edits shown in Listing 8 and then restarting Unity. In addition, remember to change the DracoUnity version in Listing 8 to your required DracoUnity version. If you want to perform the same edits in an automated fashion, you can instead install the OpenUPM CLI tool and run `openupm add com.atteneder.draco@3.3.2` (or similar).

Note!: I don't recommend using the "Installer Package" link from the DracoUnity README.md, since that is just a more convoluted and fragile method for performing the text edits shown in Listing 8. While the installer link is more automatic, it prevents the user from understanding what is really going on under the hood. Another advantage of manually editing `manifest.json` is that you can pin DracoUnity to a specific version (if desired).

In addition, please note the following "gotcha" when installing DracoUnity:

- When building for the PC/Standalone platform, remember to change `Architecture` from `x86` to `x86_64`. Otherwise, the native DLL for DracoUnity (`draco_unity.dll`) will not be included in the build, and you may get a `DllNotFoundException` when you run your application.

```
{
  "dependencies" : {
    "com.unity.collab-proxy" : "1.2.16",
    "com.unity.ide.rider" : "1.1.4",
    "com.unity.ide.vscode" : "1.2.0",
    "com.unity.test-framework" : "1.1.13",
    "com.unity.textmeshpro" : "2.0.1",
    "com.unity.timeline" : "1.2.14",
```

```
    "com.unity.ugui" : "1.0.0",
    "com.unity.modules.ai" : "1.0.0",
    "com.unity.modules.androidjni" : "1.0.0",
    "com.unity.modules.animation" : "1.0.0",
    "com.unity.modules.assetbundle" : "1.0.0",
    "com.unity.modules.audio" : "1.0.0",
    "com.unity.modules.cloth" : "1.0.0",
    "com.unity.modules.director" : "1.0.0",
    "com.unity.modules.imageconversion" : "1.0.0",
    "com.unity.modules.imgui" : "1.0.0",
    "com.unity.modules.jsonserialize" : "1.0.0",
    "com.unity.modules.particlesystem" : "1.0.0",
    "com.unity.modules.physics" : "1.0.0",
    "com.unity.modules.physics2d" : "1.0.0",
    "com.unity.modules.screencapture" : "1.0.0",
    "com.unity.modules.terrain" : "1.0.0",
    "com.unity.modules.terrainphysics" : "1.0.0",
    "com.unity.modules.tilemap" : "1.0.0",
    "com.unity.modules.ui" : "1.0.0",
    "com.unity.modules.uielements" : "1.0.0",
    "com.unity.modules.umbra" : "1.0.0",
    "com.unity.modules.unityanalytics" : "1.0.0",
    "com.unity.modules.unitywebrequest" : "1.0.0",
    "com.unity.modules.unitywebrequestassetbundle" : "1.0.0",
    "com.unity.modules.unitywebrequestaudio" : "1.0.0",
    "com.unity.modules.unitywebrequesttexture" : "1.0.0",
    "com.unity.modules.unitywebrequestwww" : "1.0.0",
    "com.unity.modules.vehicles" : "1.0.0",
    "com.unity.modules.video" : "1.0.0",
    "com.unity.modules.vr" : "1.0.0",
    "com.unity.modules.wind" : "1.0.0",
    "com.unity.modules.xr" : "1.0.0",
    "com.atteneder.draco" : "3.3.2"
  },
  "scopedRegistries" : [
    {
      "name" : "OpenUPM",
      "url" : "https://package.openupm.com",
      "scopes" : [
        "com.atteneder"
      ]
```

```
    }
  ]
}
```

Listing 8: Example edits to `Packages/manifest.json` in order to install DracoUnity. After adding the highlighted text, restart Unity to install the package.

Draco-compressing Your glTF Meshes

Most glTF files store their meshes in standard uncompressed form. To Draco-compress the meshes in your glTF files, I recommend using the gltf-transform command line tool.

In order to install `gltf-transform`, you will first need to install NodeJS and NPM. Then you will be able to install `gltf-transform` by running:

```
npm install --global @gltf-transform/cli
```

Once `gltf-transform` is installed, you will be able to Draco-compress your glTF files by running:

```
gltf-transform draco input.glb output.glb
```

The `gltf-transform draco` command provides various options for controlling compression and quantization. See `gltf-transform draco --help` for further details.

URP Support (Unity 2019.3+)

Piglet supports the Universal Render Pipeline (URP) in Unity 2019.3 or newer. To use Piglet with a URP-based project, unpack the shaders from the appropriate `.unitypackage` file in `Assets/Piglet/Extras`. For Unity versions 2019.3.0f6 through 2020.1.x, use `URP-Shaders-2019.3.unitypackage`. For Unity 2020.2.0b14 or newer, use `URP-Shaders-2020.2.unitypackage`.

The shader files will be unpacked into `Assets/Piglet/Resources/Shaders/URP`. If you to forget unpack the shaders before performing an Editor or runtime glTF import, Piglet will fail with an error reminding you to install the shaders.

Sample Application: PigletViewer

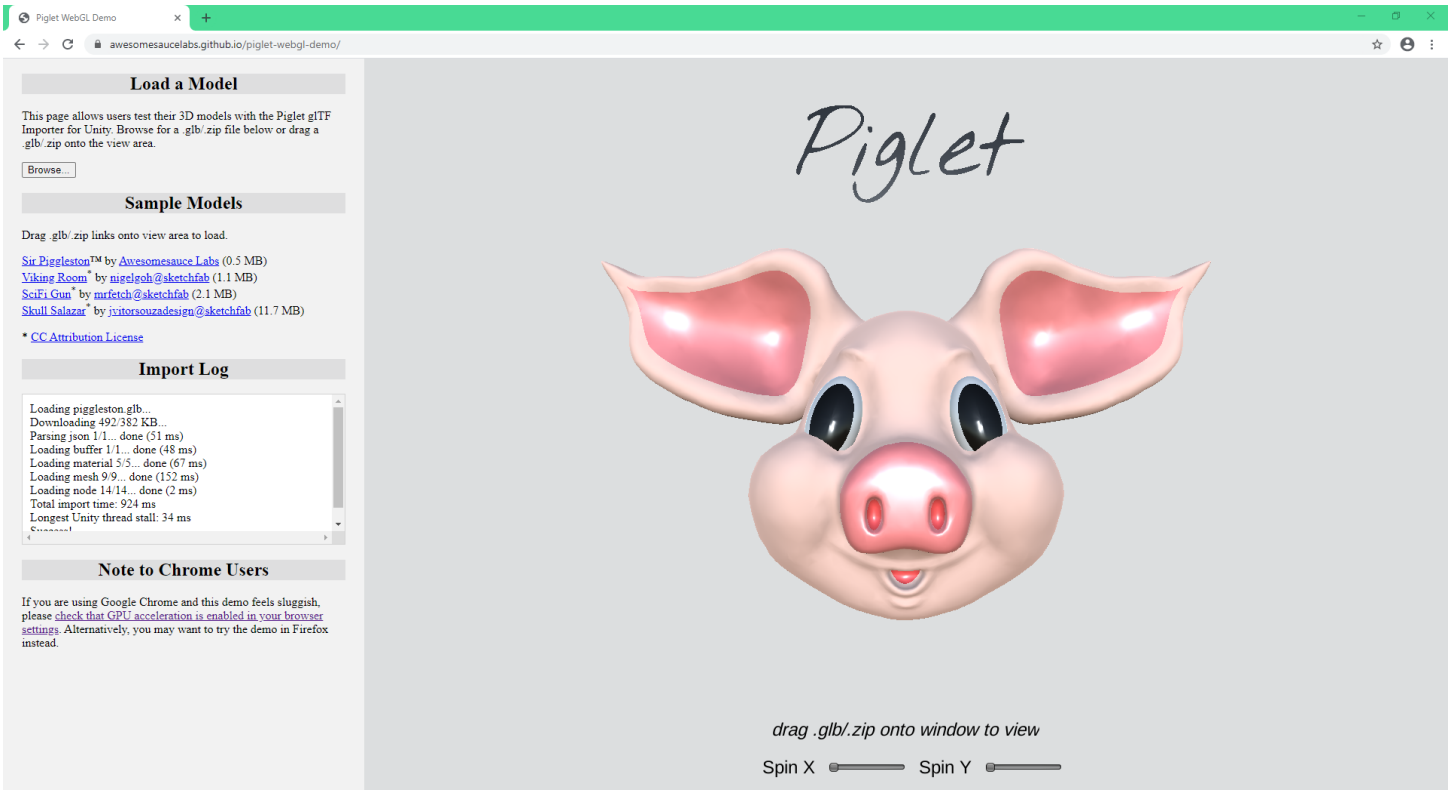


Figure 8: A screenshot of PigletViewer, a sample application which uses Piglet to view 3D models from glTF files.

PigletViewer is a sample application which uses Piglet to view 3D models from glTF files (`.gltf`, `.glb`, or `.zip`), and which is used for the Piglet Web Demo. For the benefit of Piglet customers, I have published the source code and documentation for PigletViewer online at https://github.com/AwesomesauceLabs/piglet-viewer, under an MIT license. PigletViewer currently supports builds for Windows, Mac, Android, iOS, and WebGL, and thus it may be a useful reference for Piglet users developing for those platforms. In general, I recommend looking at the Runtime Import Tutorial before exploring the PigletViewer code, as the tutorial provides a much more succinct introduction to the Piglet API.

Changelog

Release 1.3.8 (2023-07-17)

Added:

- Added support for `KHR_materials_variants`. See the “Selecting Materials Variants” and “Runtime Materials Variants Tutorial” in the manual for usage details.
- Improved naming of texture assets. For any unnamed textures in the glTF file, fall back to using the associated glTF image name (if any). If the associated image is also unnamed, fall back to using the generic texture name (“texture_0”, “texture_1”, etc.). Giving the textures meaningful/recognizable names can be very helpful in debugging scenarios.

Fixed:

- Fixed Editor import failures, when glTF file contains POINTS meshes or LINES meshes. (Piglet does not currently support any mesh types except TRIANGLES, and simply skips over other types of meshes.)
- Fixed import failure and misleading error message when glTF file requires `KHR_texture_basisu`: “This model requires support for the `KHR_texture_basisu` glTF extension, but Piglet does not yet support that extension.” Piglet *does* support `KHR_texture_basisu`, if the user installs the `KtxUnity` package! (See “Supercompressed Textures” in the Piglet manual for instructions.)

Release 1.3.7 (2022-06-21)

Added:

- UWP is now officially supported (e.g. HoloLens).
- Added support for `KHR_texture_transform`.

- Added support for `KHR_materials_unlit`.
- Added `CreateMipmaps` option to create mipmaps for PNG/JPG textures during runtime glTF imports. This option is disabled by default because the wallclock time for importing textures is roughly doubled. The `CreateMipmaps` option has no effect on Editor glTF imports because mipmaps are always created in that case.
- Added `EnsureQuaternionContinuity` option for both Editor and runtime glTF imports. This option is enabled by default, but disabling it sometimes solves minor animation glitches.
- Added `ZipPassword` for decrypting password-protected zip files (runtime glTF imports only).

Fixed:

- Fixed incorrect values for color factors (e.g. `baseColorFactor`, `emissiveFactor`). Unity expects `Color` shader properties to be provided in Gamma space, whereas I was directly passing through the linear color values from the glTF file. As a result, the colors for untextured glTF models were too dark/saturated. Most textured glTF models will be unaffected by this change, since they typically use the default white/black values for the color factors.
- Fixed incorrect rendering of `VoxEdit`-generated glTF files as solid black. This problem was caused by my use of negative UV coords, in order to fix the orientation of upside-down PNG/JPG textures. (For some reason, Unity's texture loading APIs load PNG/JPG images into textures upside-down.) I have now updated the code to use equivalent UV coords in the [0,1] range, so that textures render correctly with both `Repeat` and `Clamp` modes.
- Got rid of the annoying (but harmless) warning:
  `The Animator Controller (controller) you have used is not valid. Animations wil`
  when importing animations in the Editor.
- Piglet imports now fail with an error message if the glTF file requires a glTF extension (e.g. `KHR_mesh_quantization`) that is not implemented by Piglet. Previously, Piglet would continue importing the file anyway, and then crash or produce incorrect results. Failing with an error message is better because it lets the user know the exact cause of the problem.
- For projects that use the built-in render pipeline, materials will now render as single-sided or double-sided as per the `doubleSided` flag of the glTF material. Unfortunately,

the URP shaders are still hardcoded to use double-sided rendering, since Unity's shader graphs do not provide any way to control single-sided/double-sided rendering via script.

- Fixed rare animation import bug that would cause different channels within the same animation clip to become misaligned. This bug was caused by independently trimming "dead time" from the start of each animation channel. I now make sure to trim all channels in an animation clip by the same amount.
- Fixed rare crash while loading morph targets, when `mesh.weights` is not explicitly set by the glTF file. According to the glTF spec, `mesh.weights` should be treated as an array of zeroes when not explicitly set, and I have changed the Piglet code to behave accordingly.

Release 1.3.6 (2022-02-16)

Fixed:

- A permanent fix for conflicts between Piglet's `Newtonsoft.Json.dll` and Unity's "Newtonsoft Json" package (hurray!). To address this issue, I forked the `Newtonsoft.Json-for-Unity` project, moved all of the C# classes from namespace `Newtonsoft.Json -> Piglet.Newtonsoft.Json`, and renamed the output DLL file from `Newtonsoft.Json.dll -> Piglet.Newtonsoft.Json.dll`. You can see the exact changes I made at: https://github.com/AwesomesauceLabs/Newtonsoft.Json-for-Unity/commits/piglet
- IL2CPP builds work out-of-the-box now, for all supported platforms and Unity versions. Previously, IL2CPP builds in Unity versions older than Unity 2020.3.10f1 would throw exceptions on startup, due to Json.NET's use of C# reflection. Switching from the stock Json.NET DLL to a custom build of the Newtonsoft.Json-for-Unity "AOT" DLL (as described above) fixed this issue. For further info about using Json.NET with IL2CPP, see the `Newtonsoft.Json-for-Unity` wiki: https://github.com/jilleJr/Newtonsoft.Json-for-Unity/wiki/What-even-is-AOT
- Fixed animation of multi-material meshes. Previously, only one part of the mesh corresponding to the first material would be animated, while the other parts would remain stationary.

- Fixed morph targets on multi-material meshes. Previously, morph targets on multi-material meshes would cause an IndexOutOfRangeException.
- Fixed “missing URP shader” error when loading models that use the default material. (This bug was introduced in Piglet 1.3.5.)
- Fixed missing (pink) material when importing models in the Editor that use the default material. (This bug was probably introduced in Piglet 1.3.3.)

Release 1.3.5 (2022-01-06)

Fixed:

- Piglet now imports models correctly in linear rendering mode, in all possible scenarios: Editor imports or runtime imports, built-in rendering pipeline or URP. Hurray! (Previously, normal maps were too shiny and texture colors were too dark.)
- Fixed transparent materials in URP, which were previously showing as pink due to a missing “zwrite.mat” asset. (This bug was introduced in Piglet 1.3.3.)

Release 1.3.4 (2021-12-10)

Hotfix release for Piglet 1.3.3.

Fixed:

- Scrambled meshes in multi-mesh models during Editor glTF imports. I introduced a silly indexing error in Piglet 1.3.3 that was sometimes causing meshes to be assigned to the wrong GameObjects. Two models affected by this bug were the “2CylinderEngine” and “Buggy” models from https://github.com/KhronosGroup/glTF-Sample-Models/tree/master/2.0
- Incorrect resolution of relative paths by the –import option in the new PigletViewer command line interface. (See https://github.com/AwesomesauceLabs/piglet-viewer#command-line-options for general info about the new command line interface.)

Release 1.3.3 (2021-11-26)

Added:

- Automatic importing of glTF files (.gltf/.glb/.zip) that have been saved under `Assets` from an external program (e.g. Blender)! Please note that you need to switch application focus back to Unity (e.g. Alt-Tab) before the new/changed glTF file gets imported.

Changed:

- Major performance improvements for both runtime and Editor glTF imports!
  - Both runtime and Editor glTF imports now use `GltfImportTask.MillisecondsPerYield` to maximize the work done per frame.
  - Editor imports now do batch creation of Unity assets using `AssetDatabase.StartAssetEditing` / `AssetDatabase.StopAssetEditing`. For glTF files with a large number textures/materials/meshes/animations, the speedup is dramatic (e.g. 10X).
- Rearranged options in Piglet Options window (located under Window => Piglet Options in Unity menu). With the new ability to save glTF files directly under `Assets`, some of the old options no longer made sense.
- Moved code under `Assets/Piglet/Dependencies/UnityGLTF` from `UnityGLTF` namespace -> `Piglet.UnityGLTF` namespace, so that Piglet and Khronos UnityGLTF importers can be used in the same project.
- Refactored texture loading code, towards the goal of properly supporting linear color mode during runtime glTF imports (work in progress).

Fixed:

- `NullReferenceException` when a mesh does not specify a material in URP projects.
- Failure to recognize `.zip` files generated by some compression programs.
- Incorrect `sRGB` flag setting on texture assets after Editor glTF imports. Editor glTF imports now work correctly in linear color mode.

Release 1.3.2 (2021-06-21)

Bugfix release.

- Don't include `Assets/Piglet/Dependencies/Json.NET` when Piglet is installed with Unity 2020.3.10+ or Unity 2021.1.9+
- Unity 2020.3.10+ and Unity 2021.1.9+ now install the "Newtonsoft Json" package (i.e. `com.unity.nuget.newtonsoft-json`) by default when creating a new project. Including `Assets/Piglet/Dependencies/Json.NET` was causing a second copy of `Newtonsoft.Json.dll` to be added to the project, resulting in compile errors.
- See "Fixing Newtonsoft.Json.dll Errors (Json.NET)" in the Piglet manual for further details.

Release 1.3.1 (2021-06-15)

Bugfix release.

- Compatibility fixes for the recently released DracoUnity 2.x/3.x. Previously, these DracoUnity versions caused "unfreed collection" errors and produced GameObjects with incorrect orientations, but these problems are now fixed. Users are now recommended to use the latest DracoUnity version (currently DracoUnity 3.0.3).
- Fixed removal of non-Latin characters (e.g. Chinese characters) from GameObject names

Release 1.3.0 (2021-05-13)

This release finally adds support for Draco mesh compression!

- Reduces glTF files sizes (e.g. 20% of original size) without affecting performance
- Requires Unity 2019.3+ and installation of DracoUnity package

- Requires preprocessing your glTF files
- See Draco Mesh Compression section of the Piglet manual for further details

Bugfixes also included in this release:

- Print a warning and skip over meshes of type POINTS/LINES (previously Piglet would crash unless all meshes had type TRIANGLES)
- Fix escaping of illegal characters in filenames on MacOS
- Do not create upside-down PNG/JPEG texture assets during Editor glTF imports (to compensate for Unity's upside-down PNG/JPEG loading, set TextureScale = (1, -1) on the materials instead)

Release 1.2.2 (2021-04-16)

Bugfix release.

- Fix KTX2 texture loading in Unity 2021.1.1f1. Loading KTX2 images from a URI (local file or HTTP URL) was previously failing and generating a red-and-white "?" placeholder texture. (I attempted to fix this issue in Piglet 1.2.1, but my fix was not correct.)

Release 1.2.1 (2021-04-08)

Compatibility fix for Unity 2021.1.1f1:

- Fix NullReferenceException during texture loading, due to change in behaviour of UnityWebRequestTexture.

Release 1.2.0 (2021-02-09)

This release adds support for “supercompressed” textures in KTX2/ETC1S and KTX2/UASTC formats:

- Speeds up texture loading and reduces glTF file sizes
- Requires Unity 2019.3+ and installation of KtxUnity package
- Requires preprocessing your glTF files
- See Using Supercompressed Textures in manual for details

Bugfixes:

- Reduce memory usage during runtime glTF imports by marking textures as “non-readable”. (This was a silly mistake on my part.)
- When loading glTF files from HTTP URLs, proceed even if the initial HEAD request to determine file size fails. (Some cloud hosts don’t support HEAD requests.)
- Fix odd opaque appearance of models that are both transparent and metallic. (Change `alpha:auto -> alpha:fade` in `MetallicRoughnessBlend` and `SpecularGlossinessBlend` shaders.)
- Fix warnings about `.meta` files for non-existing folders, whenever an Editor glTF import overwrites an existing folder.

Release 1.1.1 (2021-01-05)

Bugfix release:

- Fix handling of duplicate names in glTF files (e.g. two meshes named “mesh”). Since Piglet 1.1.0, duplicate names would cause asset files to overwrite each other during Editor imports, resulting in missing model components (textures, materials, meshes, animations).
- Fix random number generation bug that was causing duplicate temp file names. This was causing runtime glTF import tasks to clobber each other’s temp files and/or generate Windows “sharing violation” errors, if the import tasks were run in parallel or in quick succession.

Release 1.1.0 (2020-12-11)

Major release!

This release adds animation support:

- Import and play glTF animations, in the Editor or at runtime
- Animation of skins and blendshapes is fully supported
- Editor imports create Mecanim (default) or Legacy clips, runtime imports create Legacy clips
- A “Static Pose” clip is automatically generated to reset the model to its default pose
- Full details provided in new manual sections and tutorial videos

Minor features:

- Editor glTF imports now use asset filenames that correspond to the original names from the glTF file. For example, if the glTF file contains textures named “grass” and “wood”, the output texture files are now “grass.png” and “wood.png”, instead of “texture_0.png” and “texture_1.png”.
- `RuntimeGltfImporter.GetGltfImportTask` methods now accepts an optional `GltfImportOptions` argument
- New option to automatically unhide model after import (enabled by default)
- New option to scale imported models to a standard size (disabled by default)
- New option to enable/disable import of animations (enabled by default)
- New options available in Piglet Options Window (see Window -> Piglet Options in Unity Menu)

Bugfixes:

- Fix URPZWrite shader compile error in Unity 2020.2.0b14+ (“invalid conditional expression”)
- URP projects now need to install either URP-Shaders-2019.3.unitypackage and URP-Shaders-2020.2.unitypackage, based on Unity version

Documentation:

- New manual sections for animation
- Manual is now included with Piglet in HTML format, rather than PDF (automated HTML -> PDF conversion was very error-prone)
- Moved Changelog into manual, so prospective Piglet customers can read it
- Nicer manual formatting with `pandoc` and `classless.css`

Release 1.0.4 (2020-09-28)

Added support for URP in Unity 2019.3+:

- If your project uses URP, just unpack Assets/Piglet/Extras/URPShaders.unitypackage and you are ready to go!

Bugfixes:

- Fix import failures when file path contains spaces
- Fix _emissiveFactor in shaders for built-in render pipeline
- Fix warning: “Script ‘Animation’ has the same name as built-in Unity component.”
- Fix warnings about obsolete UnityWebRequest API in Unity 2020.2

Release 1.0.3 (2020-09-02)

Bugfix release.

GltfImporter:

- Support models with > 64k vertices per mesh
- Minor code cleanup and refactoring (e.g. use more consistent method names, remove unused method parameters)

Release 1.0.2 (2020-08-10)

Bugfix release.

- Add .asmdef files to Piglet tree, so that advanced users have better control over their build config

ChangeLog.txt [new]:

- Move ChangeLog.txt to Assets/Piglet/Documentation, so that users can see it.

GltfImporter:

- Throw Piglet.JsonParseException instead of Newtonsoft.Json.JsonParseException on glTF parsing errors. This decouples Piglet applications from Json.NET, so that Piglet has the option to use other JSON parsing libraries in the future.

UnityGLTF:

- Replace GLTFSerialization.dll with C# source files, so that Piglet no longer depends on a specific version of Newtonsoft.Json.dll (10:0:0:0). This allows users to resolve the multiple-copies-of-Json.NET problem by deleting Piglet's copy of Newtonsoft.Json.dll. It also gives users more convenient access to the glTF parsing code, for their own understanding/modification.

Release 1.0.1 (2020-07-20)

Bugfixes:

- Fix hardcoded paths that would cause NullReferenceException after "Piglet" folder was moved/renamed
- Speed up reads from Android content URIs

Release 1.0.0 (2020-07-10)

First release!

Footnotes

1. I have tested the Piglet Web Demo with Firefox and Google Chrome on Windows 10 64-bit. If you are using Google Chrome, you can improve performance of the demo by turning on hardware acceleration (i.e. GPU acceleration) in the browser settings. Currently this option is disabled in Chrome by default.↩

2. To prevent DLL conflicts with Unity's "Newtonsoft Json" package, which is automatically installed in new Unity projects since Unity 2020.3.10f1, I have forked the Newtonsoft.Json-for-Unity project and compiled my own DLL (`Assets/Piglet/Dependencies/Json.NET/Piglet.Newtonsoft.Json.dll`). In addition to renaming the DLL from `Newtonsoft.Json.dll -> Piglet.Newtonsoft.Json.dll`, I changed the namespace of all C# classes from `Newtonsoft.Json -> Piglet.Newtonsoft.Json` and disabled some optional Json.NET features to reduce the file size of the DLL. You can see the changes I've made by looking at the commits on the `piglet` branch of my Newtonsoft.Json-for-Unity fork.↩

3. The `Assets/Piglet/Dependencies/UnityGLTF` folder does not include the full set of source files from the Sketchfab/UnityGLTF project. Piglet is actually a (heavily modified) fork of Sketchfab/UnityGLTF starting at commit c54fd45, and the `Assets/Piglet/Dependencies/UnityGLTF` folder only contains the subset of the source files that have remained (mostly) unchanged since the fork. I've changed the namespace of all C# classes from `GLTF -> Piglet.GLTF` to prevent code conflicts in Unity projects that want to use both Piglet and UnityGLTF at the same time.↩

4. As of December 2020, Unity has two animation systems: Mecanim and Legacy. While Unity recommends that new projects use Mecanim, each system has its own advantages and drawbacks. Briefly, Mecanim is a newer system that has more features than Legacy

(e.g. blending, retargeting), but has a steeper learning curve and does not (yet) provide an API for creating animations at runtime. On the other hand, the Legacy animation system is simpler, easier to learn, and supports creating animations at runtime. You should not let the name “Legacy” discourage you from using the Legacy animation system if it is a good fit for your project. Unity has continued to support and maintain the Legacy system since the introduction of Mecanim in Unity 4 (November 2012), and it is unlikely that the Legacy system will be removed until Mecanim supports runtime creation of animation clips.↩

5. If you are new to the Mecanim animation system, I highly recommend watching a series of videos called Controlling Animation on Unity Learn. These videos provide a very concise introduction to the capabilities and use of Mecanim.↩

6. Runtime glTF imports use the Legacy animation system because Mecanim is not capable of creating animation clips at runtime.↩

7. `AnimatorController` layers have not been mentioned up to this point, because the controllers generated by Piglet only use the default layer (i.e. layer 0, “Base Layer”). An `AnimatorController` can (optionally) be split into multiple layers, where each layer has its own state machine, for the purpose of blending multiple animations for the same mode (e.g. a running and shooting animation for a humanoid character). For further information, see the “Animator Controllers Layers” video from the Controlling Animation video series on Unity Learn.↩

8. The “Static Pose” clip (a.k.a “Bind Pose” or “T-Pose”) is useful because playing an animation clip permanently changes the transforms (translation/rotation/scale) of the game objects in the model. Users can play the “Static Pose” clip to return the transforms to their original state.↩

9. The main limitation is that AnimationClip.SetCurve only works at runtime for Legacy animation clips.↩

10. For the motivation behind supercompressed textures, see Basis Universal texture format introduction (Atteneder, 2019). For a more detailed technical explanation, see GST: GPU-decodable Supercompressed Textures (Krajcevski et al., 2016).↩

11. KTX2/ETC1S is more commonly used than KTX2/UASTC because it provides a higher data compression rate (at the cost of image quality).↩

12. For an introduction to the ideas behind Draco mesh compression, see Edgebreaker, the Heart of Google Draco. For a detailed description of the algorithm, see Edgebreaker: Connectivity compression for triangle meshes (Rossignac, 1999) and the follow-up paper 3D Compression Made Simple: Edgebreaker on a Corner-Table (Rossignac et al., 2001).↩