Veridian Imposters - Documentation

Index

1. Introduction

1.1 Welcome to Veridian Imposters!

Thank you for downloading and installing Veridian Imposters! This tool is the first public release from **Veridian Systems**, developed with the goal of providing a robust, professional-grade optimization solution to the Unity community for free. Our philosophy is to build a brand and a community by offering high-value tools that solve real-world (game) development problems. This asset is the foundation of that effort, and we plan to build upon it with more advanced features and new assets in the future.

This document serves as a comprehensive guide to every feature and concept within the Veridian Imposters system. We have aimed to be as detailed as possible to empower you to get the best results and to pre-emptively answer any questions you might have.

1.2 The 'Why': Understanding Imposters and Performance

In modern game development, creating vast, lush, and believable environments is a primary goal. However, this ambition often collides with the technical limitations of real-time rendering. One of the most significant performance bottlenecks, especially in natural environments, is foliage. A single, high-quality tree model can consist of tens of thousands of triangles, and a scene may require hundreds or even thousands of such trees. Rendering this amount of complex geometry every frame is a monumental task for any GPU.

The Problem: The Cost of Detail

- **Vertex Count:** Each tree adds a significant number of vertices to the scene. A model with 15,000 triangles has at least that many vertices to be processed by the GPU. A forest of 100 such trees is already 1.5 million triangles, before accounting for any other objects, characters, or terrain in the scene.
- **Shader Complexity:** The shaders used to render realistic foliage are often complex. They may calculate translucency (light passing through leaves), subsurface scattering, and dynamic wind animation. These calculations are expensive and are performed for every pixel the object covers on screen.
- **Overdraw:** Foliage models have a large amount of empty, transparent space between leaves and branches. The GPU still has to run the fragment shader for these transparent pixels just to determine they should be discarded, a costly problem known as overdraw.

The Solution: Level of Detail (LOD) and Imposters

The standard solution to this problem is using a **Level of Detail (LOD)** system. An LOD system swaps out high-detail models for lower-detail versions as they get farther from the camera. An **imposter** is the ultimate expression of this concept; it is the final, most aggressively optimized LOD level.

Veridian Imposters creates a sophisticated type of billboard. Instead of a single flat plane that looks unconvincing from the side, it generates a mesh of several intersecting planes, or **quads**. It then captures the appearance of your original 3D model onto these planes as a texture. When viewed from a distance, these intersecting textured planes create a convincing pseudo-3D illusion that looks solid from almost any angle.

The performance gains are staggering. A 3-quad imposter consists of just 6 triangles. A 4-quad imposter is only 8 triangles.

- **Vertex Reduction:** A 15,000-triangle tree is reduced to a 12-triangle imposter (for a 3-quad setup). This represents a geometric complexity reduction of over **99.9%**.
  15,000 triangles→12 triangles
- **Shader Reduction:** The imposter material uses one of Unity's highly optimized, standard Lit shaders, which is far less computationally intensive than a specialized foliage shader.

By replacing distant, complex trees with these lightweight imposters, you can render vast, dense forests while maintaining high and stable frame rates.

1.3 Introducing Veridian Imposters: Features and Philosophy

The core philosophy behind Veridian Imposters is **Compatibility and Control**. We built this tool to integrate seamlessly into any project, regardless of the render pipeline or workflow, while giving you deep control over the final result.

Feature Breakdown:

- **True Multi-Pipeline Architecture:** Veridian Imposters is designed from the ground up to work flawlessly with the **Built-in Render Pipeline**, the **Universal Render Pipeline (URP)**, and the **High Definition Render Pipeline (HDRP)**. At runtime, the tool automatically detects which pipeline your project is using by checking `GraphicsSettings.defaultRenderPipeline`. It then intelligently selects the correct shaders (e.g., `Universal Render Pipeline/Lit`, `HDRP/Lit`, or the standard `Standard` shader) and configures the generated materials with the appropriate properties. There are no manual steps required to switch between pipelines; it simply works out of the box.
- **Advanced Mesh Generation:**
  - **Radial Planes:** You can choose to generate imposters with 4, 6, 8, 12, or 16 viewing angles, which correspond to 2, 3, 4, 6, or 8 intersecting quads, respectively. This gives you fine-grained control over the trade-off between the smoothness of the 3D illusion and the final triangle count.
  - **Horizontal Quads:** This unique feature allows you to add one or more horizontal planes to the imposter mesh. The system performs an additional top-down orthographic capture of your model and maps it to these planes. This is critical for making objects like trees look solid when viewed from above (e.g., from a cliff or in a flight game) and is also an excellent technique for creating fields of grass or low-lying bushes.
  - **Profile Shaping:** You can choose between two mesh profiles:
    - **`Quad`:** A standard, rectangular plane. Simple and efficient.

        - **`Octagon`:** A custom 8-vertex shape that can be tapered at the top and bottom. This allows the mesh to more tightly fit the silhouette of your model, reducing transparent pixel area and helping to mitigate overdraw.
- **Non-Destructive Workflow and LOD Integration:**
    - Veridian Imposters **never modifies your original assets**. It creates all new files (prefab, mesh, material, textures) in a newly generated sub-folder, keeping your project clean and your source assets safe.
    - The tool features intelligent `LOD Group` integration. If your source prefab already has an `LOD Group` component, the system will non-destructively insert the generated imposter as the final LOD level and automatically recalculate the transition percentages for all preceding LODs. If your source prefab does not have an `LOD Group`, one will be created for you, with the original model as LOD0 and the new imposter as LOD1.
- **High-Quality Texture Generation:**
    - **Controlled Lighting Environment:** To ensure clean and consistent results, the tool performs all texture captures in a temporary, isolated environment. It programmatically creates its own camera and lighting setup (`SnapshotTempKeyLight_Service`, etc.), ensuring your imposters look perfect regardless of the lighting in your active scene.
    - **Normal Maps:** Beyond capturing the object's color (Albedo), the asset can generate a high-fidelity **world-space normal map**. It does this by temporarily replacing the object's materials with a special internal shader (`Hidden/AdvancedBillboardCreator/NormalCapture_...`) that encodes directional data into an image. This allows the final imposter to react realistically to dynamic lighting in your scene, preserving depth and surface detail.
    - **Built-in Post-Processing:** The generation pipeline includes essential post-processing steps to guarantee professional quality. This includes **Supersampling (SSAA)** to reduce aliasing (jagged edges) and a multi-pass **Edge Padding** algorithm that bleeds pixel colors into transparent regions, effectively eliminating the dark halo artifacts that can appear around billboards due to texture filtering and mipmapping.

2. Getting Started

This section will guide you through the initial setup process, from installing the asset to opening the tool for the first time. We've designed the process to be as seamless as possible, but understanding the details below will ensure a smooth experience.

2.1 Installation

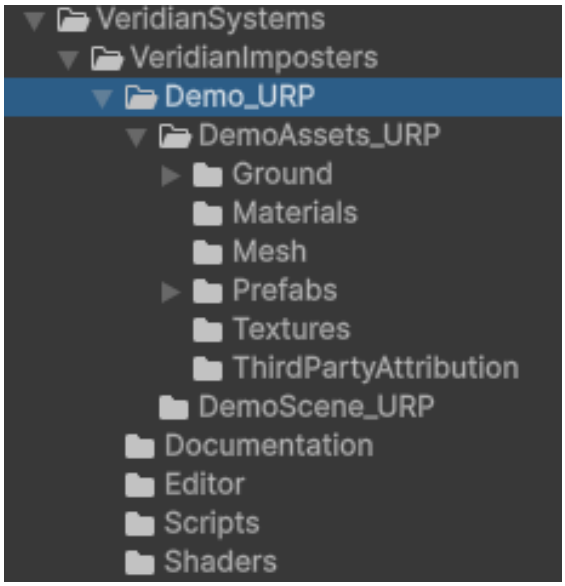
Veridian Imposters is installed through the standard Unity Package Manager.

1. Open your Unity project.
2. Navigate to the main menu and select **Window > Package Manager**.
3. In the Package Manager window, click the dropdown at the top-left and select **My Assets**. This will display all the assets you have acquired from the Unity Asset Store.
4. Find **Veridian Imposters** in the list. You can use the search bar to locate it quickly.
5. Click the **Download** button if the asset is not yet downloaded to your computer.
6. Once downloaded, the button will change to **Import**. Click it.

A new "Import Unity Package" window will appear, showing all the files and folders included in the asset.

The 'Demo' Folder: An Important Choice

Before you click the final "Import" button, take note of the folder structure.



The asset is split into two main parts:

- **Core System (`/VeridianImposters/Scripts`, `/Editor`, etc.):** These are the essential scripts that make the tool work. This part of the asset is extremely lightweight.
- **Demo Content (`/VeridianImposters/Demo`):** This folder contains an example scene with pre-configured 3D models, materials, and textures to showcase the tool's capabilities. This folder is significantly larger than the core system.

You have a choice:

- **For a minimal installation,** you can **un-check the `Demo` folder** before importing. The core tool will still function perfectly. This is the recommended approach if you do not use URP or if you want to keep your project size down.
- **To see the examples,** leave the `Demo` folder checked.

If you choose to import the demo content, you can **safely delete the entire `/VeridianImposters/Demo` folder** from your Project window at any time without affecting the tool's operation.

2.2 System Requirements & Compatibility

Veridian Imposters is built for flexibility, but it's important to be aware of its supported environments.

Unity Version

Veridian Imposters is officially developed and tested for **Unity 6 and later versions**.

While the core C# scripts and logic may function correctly on older LTS releases (such as Unity 2022.3 or 2021.3), we do not provide official support or guarantee full functionality for any version prior to Unity 6. We recommend using the latest official Unity release for the best experience.

Render Pipeline Compatibility

The core generation system of Veridian Imposters is designed to work with all of Unity's render pipelines, with some specific requirements for HDRP.

✅ **Built-in Render Pipeline:** Fully compatible. ✅ **Universal Render Pipeline (URP):** Fully compatible. ⚠️ **High Definition Render Pipeline (HDRP):** Partially compatible with a required manual workflow.

The tool features automatic pipeline detection. When you generate an imposter, the system checks your project's active render pipeline and creates a material using the correct default "Lit" shader.

- For Built-in, it uses the **Standard** shader.
- For URP, it uses the **Universal Render Pipeline/Lit** shader.
- For HDRP, it uses the **HDRP/Lit** shader.

While the tool correctly creates the assets, a manual workflow is required during the *capture* phase in HDRP to achieve correct lighting on the final textures. This involves temporarily using an Unlit material on the source object. For detailed instructions, please refer to the **"Important Instructions for HDRP Users"** section in the main README file.

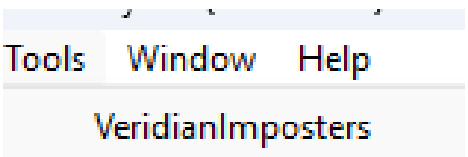
> ⚠️ **Demo Scene Incompatibility Warning**

Please be aware that the included Demo Scene and all of its assets are pre-configured for the **Universal Render Pipeline (URP) only**. If you are using the Built-in or HDRP render pipelines and you open the demo scene, the materials for the example trees will appear solid pink or magenta. This is the expected behavior in Unity when a material's shader is not compatible with the active render pipeline. The core tool itself will still function correctly in your project. (You can manually change the shaders to Standard or HDRP Lit and it should work for the other pipelines.

2.3 Accessing the Tool

Veridian Imposters is an **Editor Window**, not a component that you add to a GameObject. To open it:

1. In the Unity Editor's top menu bar, click on `Tools`.
2. Click on `Veridian Imposters` from the list.



This action will open the main **Veridian Imposters** window. This window is the central hub for all imposter creation and configuration tasks. For ease of use, we recommend docking this window somewhere convenient in your layout, such as next to the Inspector or Project windows.

2.4 Required HDRP Workflow

To get correct results in a High Definition Render Pipeline (HDRP) project, a few manual steps are required before you begin generating billboards.

Step 1: Add Scripting Define Symbol

First, you must add `UNITY_HDRP` to your project's Scripting Define Symbols. This allows the tool's scripts to enable the HDRP-specific code.

1. Go to **Edit > Project Settings > Player**.
2. Find the **"Scripting Define Symbols"** field.
3. Add `UNITY_HDRP` to the list, separating it from others with a semicolon (e.g., `EXISTING_SYMBOLS;UNITY_HDRP`).
4. Click **Apply**. Unity will recompile its scripts.

Step 2: Use an Unlit Shader for Capturing

The primary challenge with HDRP is its physically-based lighting system, which is controlled by Volumes. When capturing an object that uses the standard **HDRP/Lit** shader, the intense lighting from the controlled capture environment often results in the billboard texture being almost entirely white (overexposed).

The recommended solution is to temporarily use an **Unlit** shader on your source object during the capture process. This isolates the object from the lighting, ensuring the capture uses only the object's base texture colors.

Workflow:

1. Create a temporary material that uses a basic **Unlit** shader (e.g., `HDRP/Unlit`). Assign the base color texture from your original object to it.
2. Temporarily apply this Unlit material to the renderers of the object you want to capture.
3. Use the Billboard Creator tool to generate the billboard. You may need to adjust the **Snapshot Lighting Settings** in the tool to get the desired brightness.
4. After the billboard is created, you can revert the material on your original object back to its original HDRP/Lit shader.

This method provides a reliable way to get a clean capture in HDRP. If you discover a workflow that allows for direct capture from the HDRP/Lit shader without overexposure, please let me know so I can update the asset.

3. The Demo Scene (URP Only)

We've included a comprehensive demo scene to provide a practical, hands-on environment where you can see the results of Veridian Imposters and test its features. This section breaks down everything you need to know about the included content.

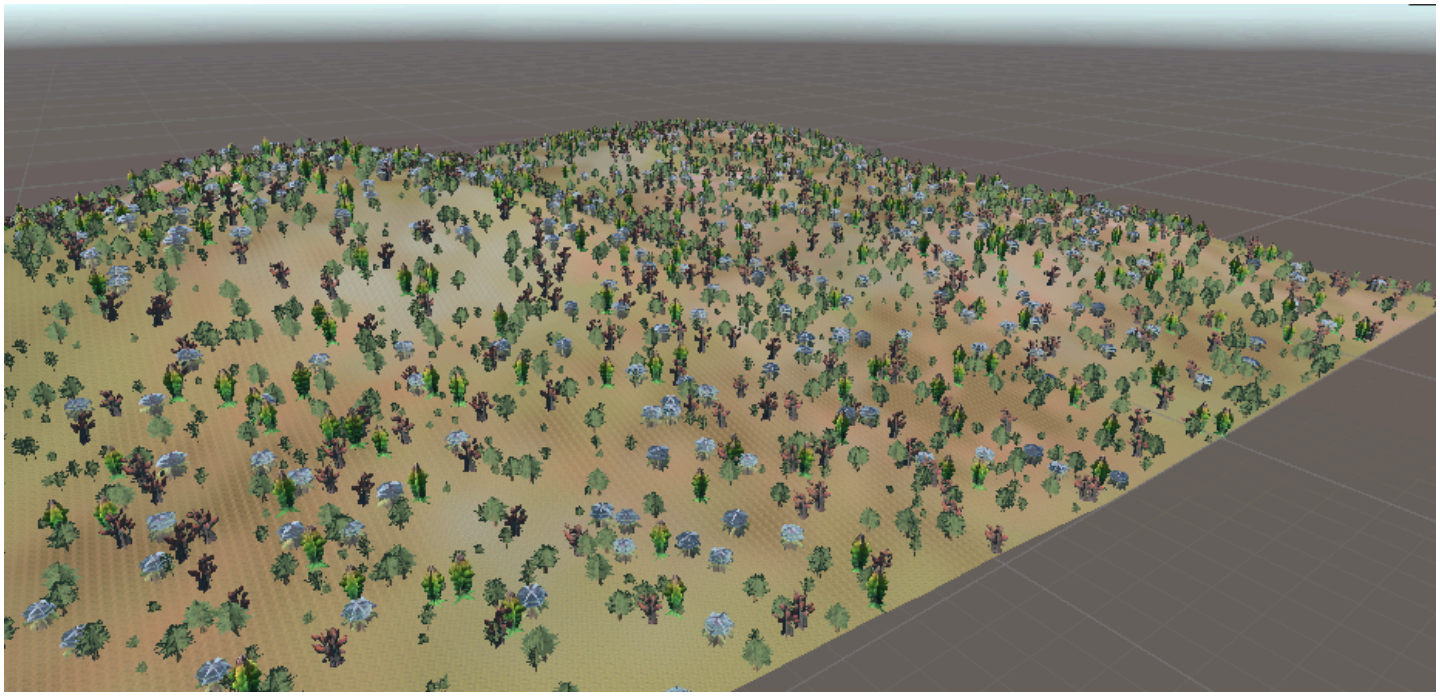
3.1 Purpose of the Demo Scene

The primary goal of the demo scene is to serve as a real-world example and a starting point for your own experiments. By exploring the scene, you can:

- **See the Final Quality:** Observe the visual fidelity of the generated imposters and how seamlessly they transition from the high-detail models.
- **Inspect the Generated Assets:** You can select the example prefabs and look at their `LOD Group` components, materials, and meshes to understand how the tool assembles the final product.
- **Find Example Source Models:** The included tree and grass assets are ready to be used as test subjects. You can drag them into the Veridian Imposters window and try generating your own imposters with different settings.

> ⚠️ **A Note on Render Pipeline Compatibility**
>
> To keep the asset's file size reasonable, the **Demo Scene and all of its associated assets are pre-configured exclusively for the Universal Render Pipeline (URP)**. We have not included duplicate assets for the Built-in and HDRP pipelines. If you open the demo scene in a non-URP project, the materials will appear bright pink/magenta. This is expected behavior. The core tool itself remains fully compatible with all pipelines.

Demo Scene - Near View The smaller objects are ‘imposter grass’.



Demo Scene - Far View (0.5km x 0.5km) All game objects shown are imposters from this distance.

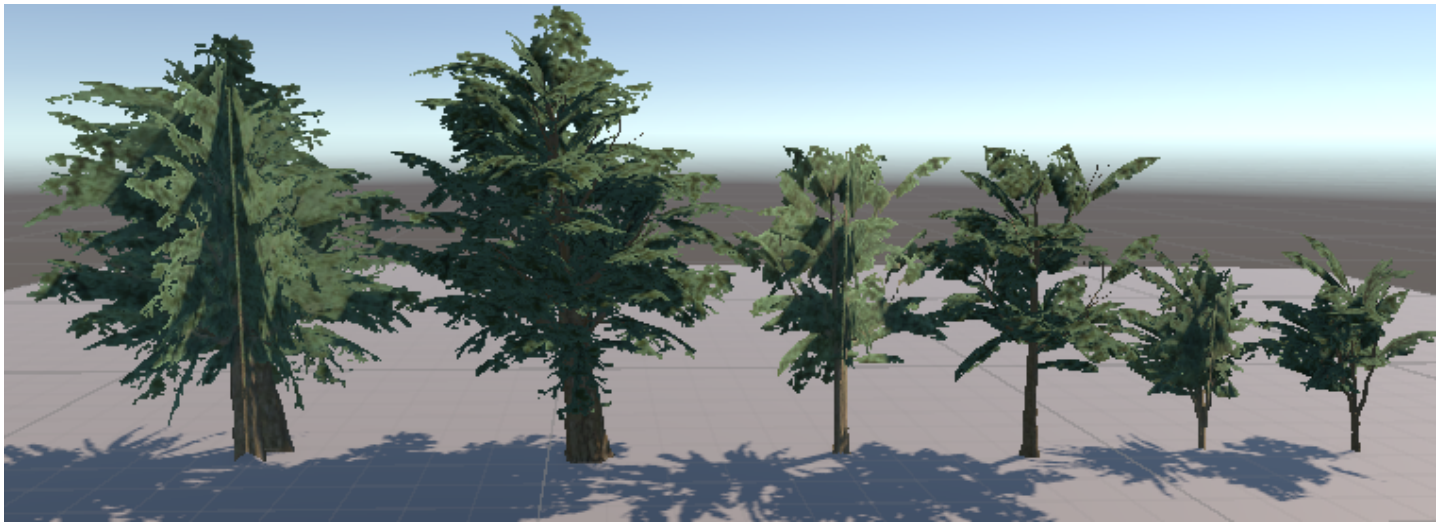
3.2 Included Assets: A Detailed Breakdown

The demo folder contains several assets you are free to use in your projects, each with its own licensing and origin. We believe in being fully transparent about our sources.

Oak Tree Set (x3)

These three realistic oak tree models are high-quality, third-party assets.

- **Source:** Sourced from a provider of CC0 assets.
- **License:** These assets are provided under a **CC0 (Creative Commons Zero) license**, which means they are in the public domain. You are free to use, modify, and distribute them in your personal and commercial projects without attribution. For full details, please refer to the `LICENSE_OAK_TREES.txt` file located in the `VeridianImposters/Demo/` folder.

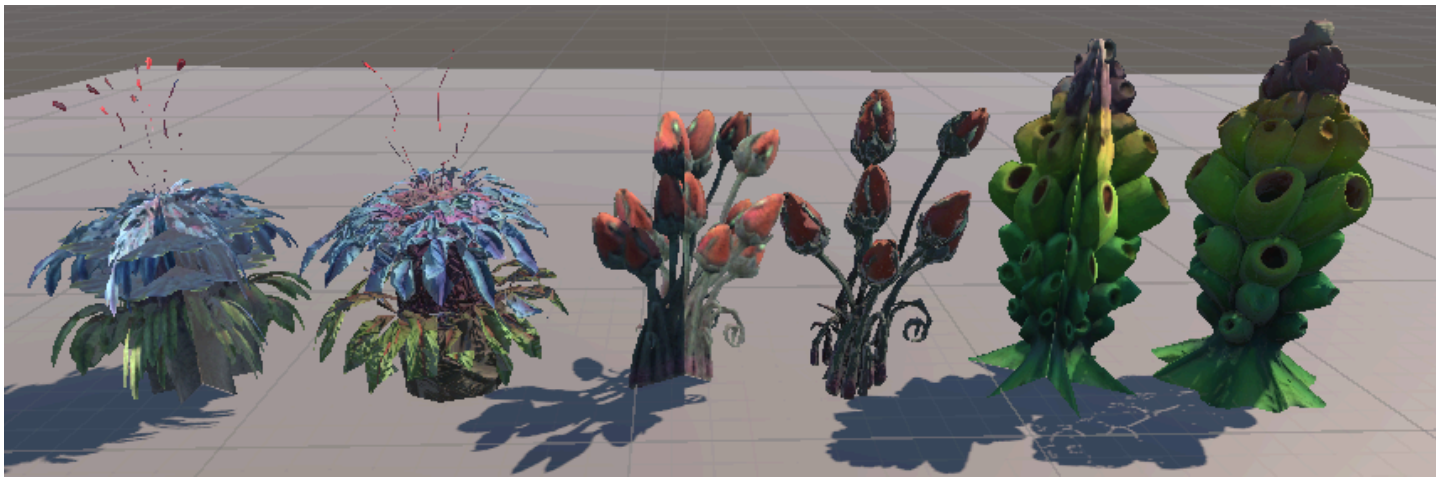


Oak Trees - Imposters (Left) vs LOD0 (Right)

AI-Generated "Alien" Tree Set (x3)

For visual variety, the demo also includes three stylized, otherworldly trees.

> **Disclaimer on AI-Generated Content**
>
> The three "alien" tree models included in this demo were created using **generative AI tools**. We understand that developers and studios have a wide range of policies regarding the use of AI-generated content. If your personal or professional workflow does not permit the use of such assets, please feel free to **delete them immediately**. They are included purely as a visual demonstration and are **not required** for Veridian Imposters to function in any way.

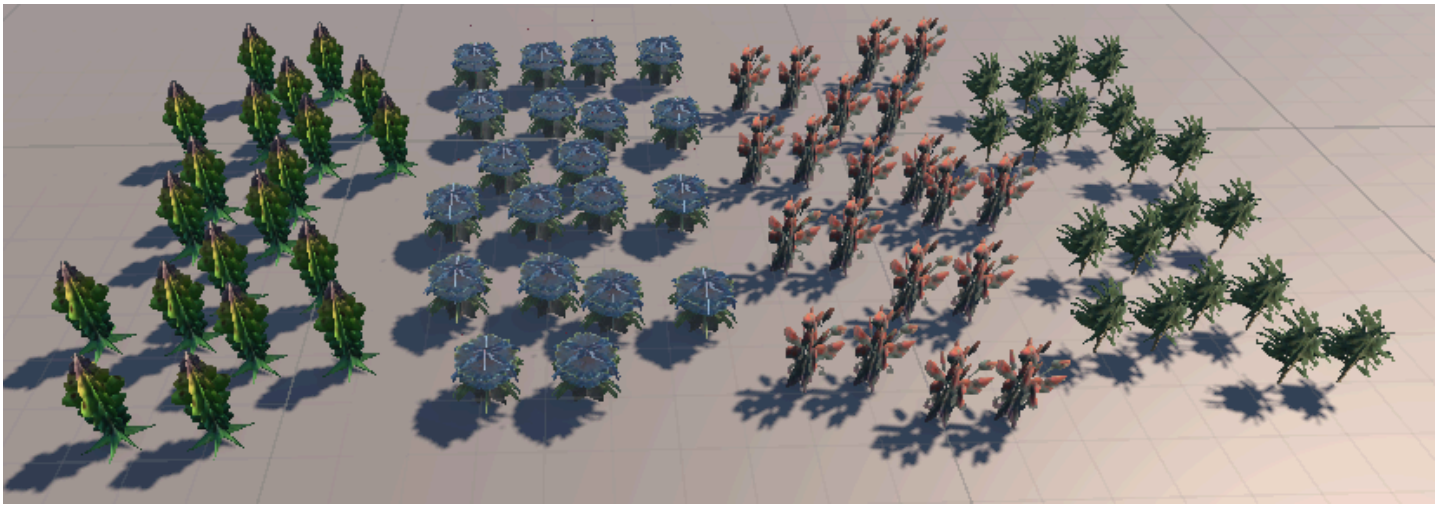


Alien Trees - Imposters (Left) vs LOD1 (Right)

Grass Demonstration

In the demo scene, you will find patches of grass that showcase the versatility of Veridian Imposters. This is not a separate grass asset, but rather an example of an alternative use case for the tool.

- **How It Was Made:** The grass was generated from a single, small 3D mesh of a grass clump. We used a **1-quad imposter setting** with an **additional horizontal top-down quad**. This is a highly effective and performant technique for creating dense-looking fields of grass, flowers, or other ground cover.



This shows the imposters used as ‘grass’

3.3 Adapting Demo Assets to Non-URP Projects

If you want to use the included 3D models in a project running the Built-in or HDRP render pipeline, you will need to manually update their materials. The process is simple:

1. **Create a New Material:** In your `Project` window, right-click and choose `Create > Material`. Give it a descriptive name (e.g., `M_Oak_Tree_HDRP`).
2. **Set the Shader:** Select your new material. In the Inspector window, click the **Shader** dropdown menu at the top. Choose the appropriate default "Lit" shader for your pipeline:
    - For **Built-in:** `Standard`
    - For **HDRP:** `HDRP/Lit`
3. **Assign Textures:** Locate the textures in the `VeridianImposters/Demo/Textures/` folder.
    - Drag the **Albedo** texture (e.g., `T_OakTree_D.png`) into the `Base Map` (for URP/HDRP) or `Albedo` (for Built-in) texture slot on your new material.
    - Drag the **Normal** map (e.g., `T_OakTree_N.png`) into the `Normal Map` slot. Unity may show a "Fix Now" button if the texture is not marked as a normal map; click it.
4. **Apply the Material:** Find the oak tree model in the demo assets folder and assign your newly created material to its mesh renderers.

3.4 Managing Project Size

The core Veridian Imposters tool is incredibly lightweight, consisting of only a few C# scripts. The bulk of the asset's file size comes from the 3D models and textures in the demo folder.

If you have finished exploring the demo scene or do not require the example content, you can and should **safely delete the entire `VeridianImposters/Demo` folder** from your Project window.

This action will have **zero impact** on the functionality of the Veridian Imposters tool and will help keep your project lean.

4. The General Workflow

This section provides practical, step-by-step instructions for the most common use cases. While the core process is always the same, the setup and expected results differ depending on your source asset.

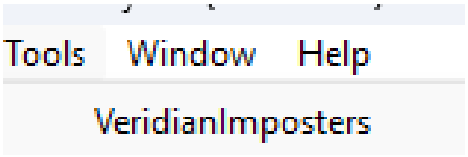
4.1 The Core Workflow: A Step-by-Step Guide

This section will walk you through the entire process of creating an imposter from start to finish. While the subsequent sections detail specific scenarios (like handling existing LODs), they all follow this fundamental process. We'll use placeholders for the screenshots you'll add to your final document.

Step 1: Open the Tool & Prepare Your Settings

First, open the Veridian Imposters window from Unity's main menu bar.

- Navigate to **`Tools > Veridian Imposters`**.



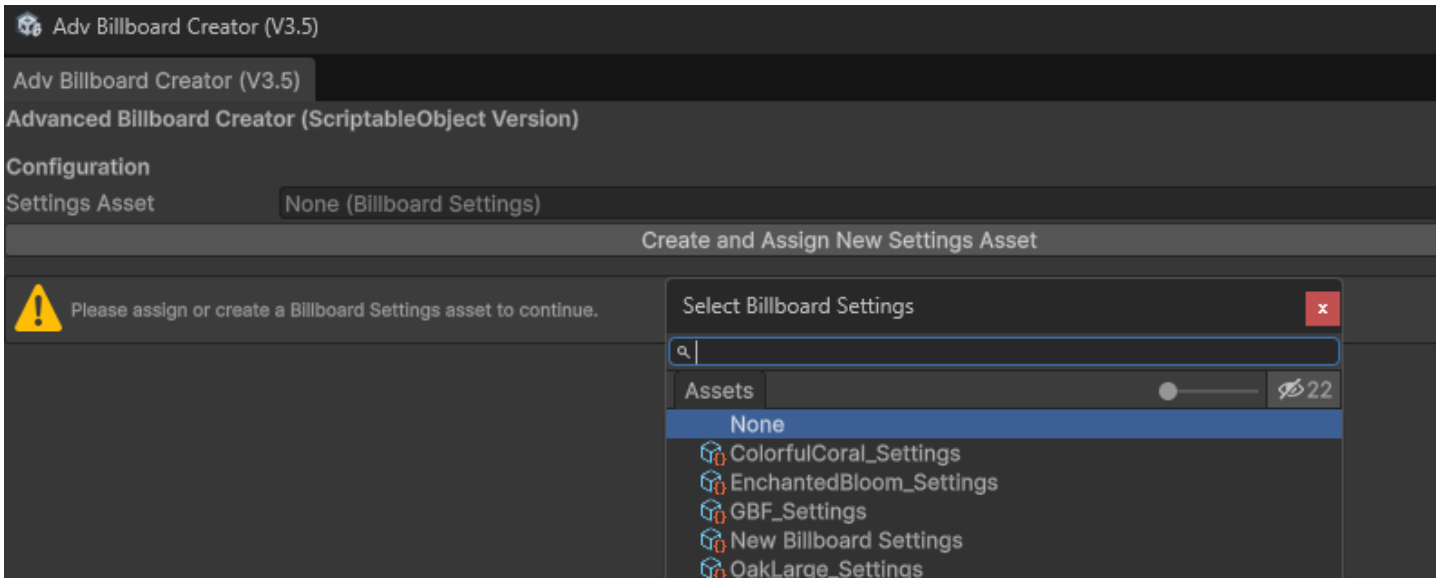
When you open the window for the first time, the main panel will be disabled, prompting you to either create a new settings asset or assign an existing one

You have two options:

1. **Create a New Preset:** Click the **`Create and Assign New Settings Asset`** button. A file save dialog will appear, allowing you to create and name a new settings file (e.g., `MyTreeSettings.asset`). This is the best option for your first time.
2. **Load an Existing Preset:** If you've already created a settings asset, you can simply drag it from your Project window into the **`Settings Asset`** field.

Remember, every change you make in the UI is **automatically saved** to this active settings asset, allowing you to build a library of presets

.

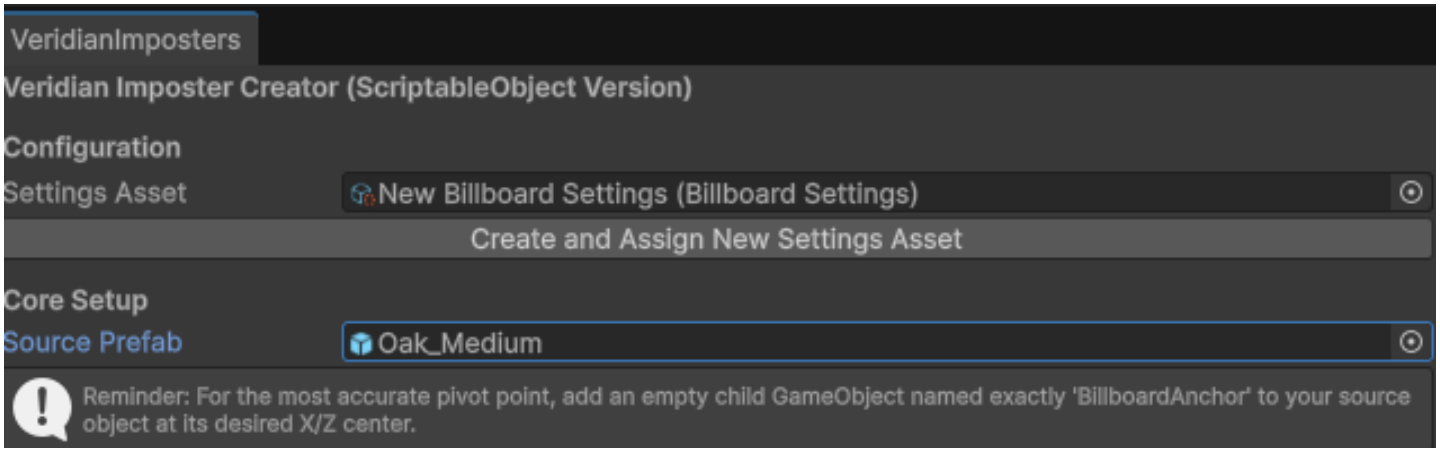


Step 2: Choose and Prepare Your Source Prefab

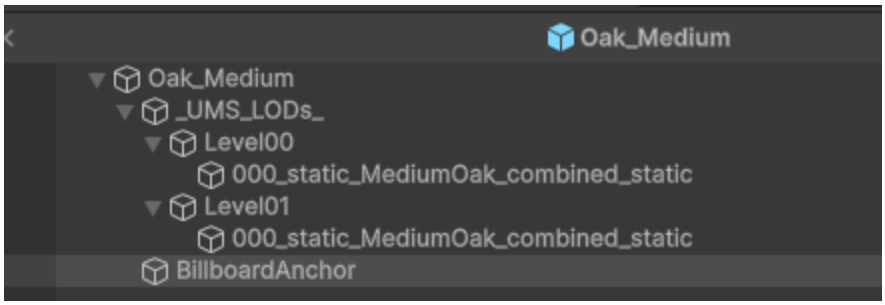
Next, you need to tell the tool which model to process.

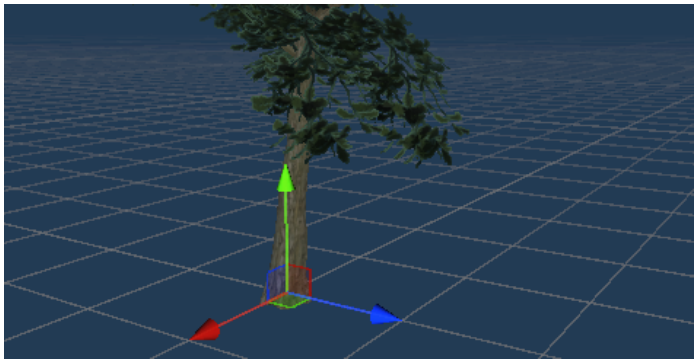
- Drag a prefab from your Project window into the **`Source Prefab`** field at the top of the UI.

It's recommended to use a source prefab that does not already have a final imposter LOD. The specific workflows later in this section provide more detail on handling different types of prefabs.



- **Crucial Step: The `BillboardAnchor`** To ensure your imposter pivots perfectly around its base, we strongly recommend adding a `BillboardAnchor`. Without it, the tool centers the imposter based on the model's overall bounding box, which can be inaccurate for asymmetrical trees.
  To add one, edit your prefab, create an empty GameObject, and rename it to exactly **`BillboardAnchor`**. Place this anchor at the center of the model's base, right where it should touch the ground.





Step 3: Customize the Imposter Settings

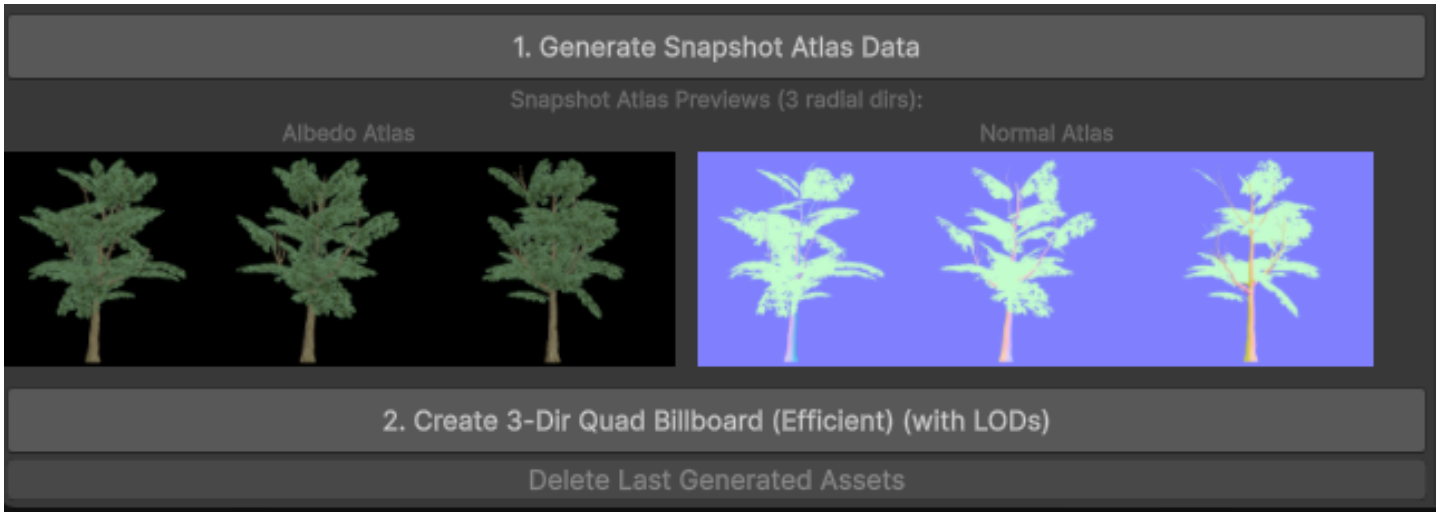
With your settings asset and source prefab assigned, you can now use the UI options to customize the imposter. You can control the number of planes, texture resolution, shape, lighting, and LOD transitions.

The specific workflows that follow offer recommendations for these settings. For a detailed explanation of every option, please see **Section 5** or simply hover over any label in the UI to see its helpful tooltip.\

Step 4: The Two-Click Generation Process

Creating the imposter is a simple two-step process using the main action buttons at the bottom of the window.

1. `1. Generate Snapshot Atlas Data` Click this button to begin the texture capture process. The tool will "photograph" your model and generate the necessary texture atlases. When complete, previews of the **Albedo** and **Normal** maps will appear in the window.
2. `2. Create Billboard Assets` Once the snapshots are generated and you're happy with the previews, this button will become active. Click it to build the final mesh, material, and the fully configured LOD prefab.
3. `Delete Last Generated Assets` This is a convenient utility button. If you're not satisfied with the result, clicking this button will find and safely delete the entire folder and all assets created during the immediately preceding run, allowing you to easily try again.



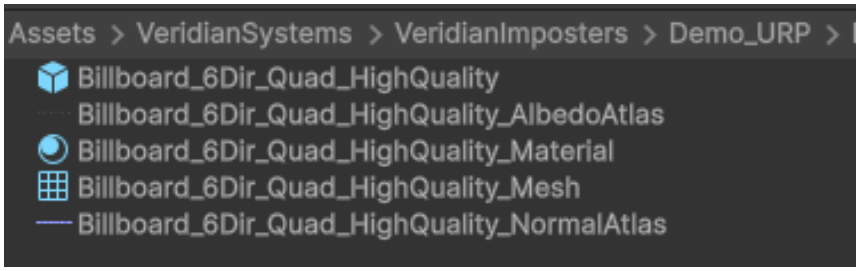
Step 5: Review Your Final Assets

After the process is complete, a new folder will be created in your Project window, located in the same directory as your source prefab.

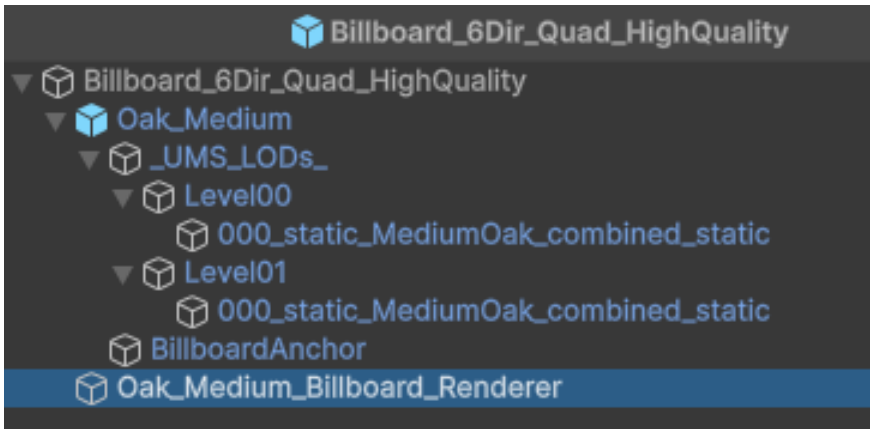
Inside this folder, you will find all the generated assets:

- The final **Prefab** with the `LOD Group`.
- The generated **Mesh** (`.asset` file).
- The configured **Material** (`.mat` file).
- The **Albedo** and **Normal** map textures (`.png` files).

> ⚠️ **Important Warning:** The generated prefab requires all of these assets to function correctly. If you manually delete the mesh, material, or texture files from this folder, the imposter will fail to render or will show errors. Use the "Delete Last Generated Assets" button for a safe cleanup.



This shows the files in the newly created folder.



This shows the Imposter as a direct child object of the newly created prefab.

4.2 Workflow 1: Asset without an Existing LOD Group

This is the most common use case: taking a single, high-detail model and creating a new, optimized prefab that will automatically switch to an imposter at a distance.

Goal: To create a new prefab with two LOD levels: **LOD0** (your original model) and **LOD1** (the new imposter).

Step-by-Step Guide:

1. **Assign Source:** Open the Veridian Imposters window (`Tools > Veridian Systems > Veridian Imposters`) and drag your source prefab (the one without an LOD Group) into the **`Source Prefab`** field.
2. **Configure LOD Settings:** In the UI, navigate to the **Output & LOD Settings** section.
    - Ensure the **`Create/Update LOD Group`** checkbox is **enabled**. This tells the tool to build the LOD system.
    - **`Billboard LOD Transition Percent`:** Adjust this slider to define when the imposter appears. This value represents the object's height relative to the screen. A value of `10%`, for example, means that when the object is far enough away that it only takes up 10% of the screen's height, Unity will switch from the full 3D model to the imposter.
    - **`Cull Cutoff Percent`:** Adjust this slider to define when the object disappears completely. For performance, you should always cull very distant objects. A value of `1.5%` is a good starting point.
3. **Configure Imposter:** Set the `Number of Snapshots`, `Resolution`, and other parameters as desired.
4. **Generate:** Click **`1. Generate Snapshot Atlas Data`**. Review the texture previews.
5. **Create:** Click **`2. Create Billboard Assets`**.

Result: A new folder will be created next to your source asset. Inside, you'll find a new prefab. If you select this prefab, you'll see in the Inspector that it has an `LOD Group` component. **LOD0** will contain your original high-detail model, and **LOD1** will be the newly generated imposter, with the transition and cull percentages you defined.

4.3 Workflow 2: Asset with an Existing LOD Group

This workflow is for optimizing assets that already have multiple levels of detail (e.g., LOD0, LOD1, LOD2). Veridian Imposters will add its generated billboard as the final LOD level.

Goal: To insert a high-performance imposter as the last LOD stage before the object is culled.

Step 1: Inspect the Source Prefab (Crucial Pre-flight Check)

Before you begin, select your source prefab in the Project window and examine its **`LOD Group`** component in the Inspector. Look at the last LOD level.

- **Does it already have an imposter?** Some assets come with their own billboards. Veridian Imposters **will not** overwrite or remove existing LODs; it only adds a new one. To avoid having two redundant imposters (which is wasteful), you should **remove the old imposter LOD** before proceeding. You can do this by right-clicking on the LOD bar in the `LOD Group` component and selecting "Delete".

Step 2: Generate the Imposter

1. **Assign Source:** Drag your prefab into the **`Source Prefab`** field.
2. **Configure LOD Settings:**
    - Ensure **`Create/Update LOD Group`** is **enabled**.
    - Set the **`Billboard LOD Transition Percent`** and **`Cull Cutoff Percent`**. This will define the transition *from* your last original mesh (e.g., LOD2) *to* the new imposter.
3. **Generate & Create:** Follow the standard two-step process to generate the textures and create the final assets.

Step 3: The Result & Required Adjustments

The tool will create a new prefab that preserves your original LODs (LOD0, LOD1, etc.) and inserts the new imposter as the final level. For an asset that had LOD0, LOD1, and LOD2, the new imposter will become **LOD3**.

- **Manual Adjustment May Be Needed:** The tool only sets the transition point *to* the new imposter. You may need to manually adjust the transition percentages for your original LODs to ensure a smooth progression. Select the new prefab, look at the `LOD Group` component, and drag the vertical separator bars to fine-tune when each LOD level becomes active.

4.4 Recommendations

Before moving on to the third workflow, here are some key recommendations to get the best results:

- **Profile Shape (`Quad` vs. `Octagon`):** For the vast majority of assets, the **`Quad`** profile is recommended. It provides excellent results with the lowest triangle count. The `Octagon` profile adds significantly more triangles to create a tighter silhouette, a feature that should be reserved for special cases, such as stylistically complex models where reducing transparent overdraw is a primary concern.
- **Horizontal Cross-Sections:** Use this feature with care. It's most effective for assets that have distinct horizontal features (like a flat-topped acacia tree) or for assets that will frequently be seen from a high angle or directly above. Adding horizontal quads increases the triangle count.
- **Quad Count:**

- **Deciduous Trees:** Most regular trees look great with **3 quads** (6 viewing directions). This provides a significant visual improvement over the traditional 2-quad cross-hatch without much extra cost.
- **Conifers:** The conical and often denser shape of conifers (like pines and firs) usually benefits from **4 or even 6 quads**. This helps maintain their volume and avoid visual gaps from more extreme viewing angles.

4.5 Workflow 3 (Advanced): Creating Standalone Grass & Detail Prefabs

This workflow shows how to use Veridian Imposters to create extremely lightweight, standalone prefabs that still have a sense of 3D volume. These are ideal for use with terrain detail systems or procedural placement tools where you need high performance and good visual quality from multiple angles.

Goal: To create a simple prefab containing *only* the imposter mesh, with no LOD system, but with enough geometry to look three-dimensional.Table of Contents

Step 1: The Source and Generation

1. **Choose a Source:** Start with a small 3D model, such as a single clump of grass, a flower, or a small bush.
2. **Configure Settings:** In the Veridian Imposters UI, set up the billboard. For this use case, you want to create volume, not a flat card.
    - `Number of Snapshots`: Set this to `6 Directions` **(3 quads) or** `8 Directions` **(4 quads)**. Using multiple intersecting planes is crucial for giving the grass a convincing sense of volume and preventing it from looking flat as the camera moves.
    - **Horizontal Quads**: Consider adding one horizontal quad. This can help fill out the object when viewed from slightly above.
3. **Generate Assets:** Run the full two-step generation process (`Generate Snapshot`, `Create Billboard`). This will create a standard LOD prefab.

Step 2: Isolating the Imposter

Now, we will extract just the volumetric imposter from the generated LOD prefab.

1. Drag the **newly generated LOD prefab** from your Project window into your scene Hierarchy.
2. In the Hierarchy, click the arrow next to the prefab instance to expand it. You will see its children: the original 3D model (LOD0) and the imposter renderer object (e.g., `GrassClump_Billboard_Renderer`).
3. Right-click on the **imposter renderer object** and select `Duplicate` (or press `Ctrl+D`/`Cmd+D`). This creates a copy that is no longer part of the original prefab instance.
4. Delete the original prefab instance from the Hierarchy. You are now left with a standalone GameObject in your scene that is just the imposter.

Step 3: Creating the Final Prefab

1. Select the standalone imposter object in your Hierarchy. You can rename it, scale it, or adjust it as needed.
2. Drag this object from the Hierarchy window directly into a folder in your Project window.

Result: You have now created a new, ultra-lightweight prefab. This asset, while using imposter textures, behaves like a small 3D object thanks to its multiple intersecting planes. It has a single material, a very low triangle count (12-16 triangles), and is perfectly optimized for use with Unity's Terrain system (as a Detail Mesh) or for scattering across a scene with any procedural placement tool.

5. Understanding the User Interface (UI)

The Veridian Imposters window is the control center for the entire generation process. While it contains many options to give you fine-grained control, it's designed to be intuitive and helpful.

5.1 Navigating the UI: Tooltips and Reminders

We've built documentation directly into the tool to help you learn as you go.

- **Tooltips:** Every single setting, slider, and checkbox in the Veridian Imposters window has a detailed tooltip. If you are ever unsure what an option does, simply **hover your mouse cursor over its text label**. A descriptive pop-up will appear explaining its purpose and how to use it.
- **Help Boxes:** You will also notice several permanent, light-blue help boxes integrated into the UI. These provide context-sensitive tips and reminders for the most important features, ensuring you don't miss a critical step.

5.2 The Most Important Setting: The `BillboardAnchor`

If you learn only one thing from this section, let it be this: using a `BillboardAnchor` is the single most effective way to ensure your imposters are perfectly centered and grounded.

The Problem it Solves: By default, Veridian Imposters calculates the center of your object by finding the center of its **bounding box**. For a perfectly symmetrical model, this works fine. However, many models are not symmetrical. A tree might have a long branch extending to one side, which would shift the calculated center of the bounding box away from the trunk. This results in an imposter that appears off-center and doesn't rotate around its base correctly.

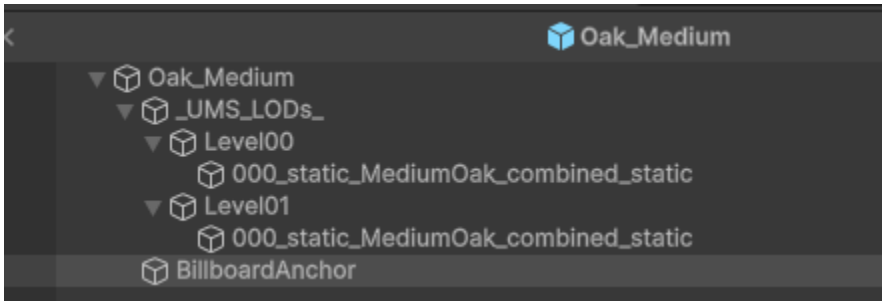
The Solution: You can override this automatic calculation by adding a special empty GameObject to your source prefab. The tool will detect this object by its name and use its exact position as the pivot point for the imposter.

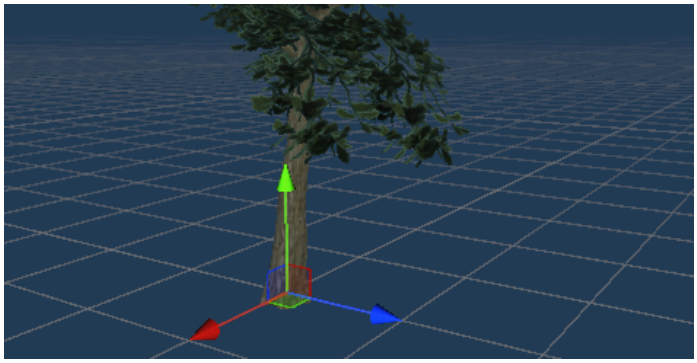
How to Add a `BillboardAnchor`:

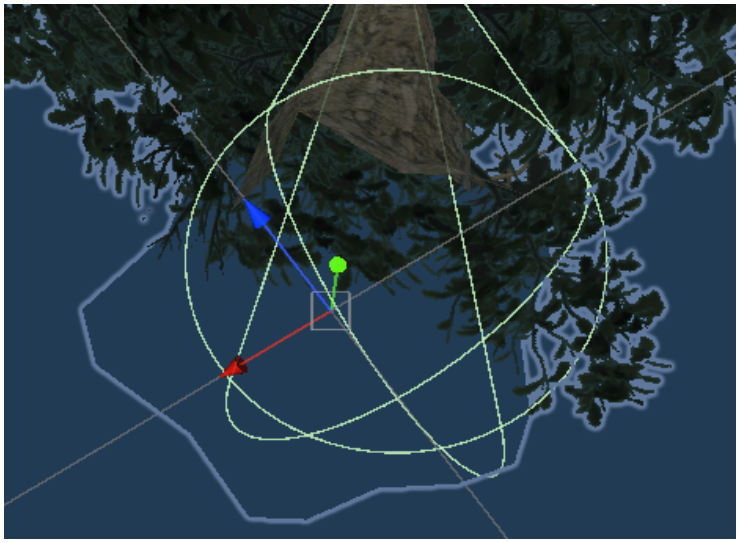
1. Drag your source prefab from the Project window into your scene to edit it.
2. In the Hierarchy window, right-click on the prefab's root object and select **`Create Empty`**.
3. Select this new, empty GameObject. In the Inspector, rename it to **`BillboardAnchor`**.
   Note: The name is **case-sensitive** and must be spelled exactly correctly.
4. With the `BillboardAnchor` object still selected, use the move tool to position it at the desired pivot point. For a tree, this is typically at the very **base of the trunk**, centered in the X and Z dimensions. For a bush, it would be the center of the plant's base where it meets the ground.
5. Once the anchor is positioned, apply the changes to your source prefab. Select the prefab's root object in the Hierarchy, and in the Inspector, click the **`Overrides`** dropdown and select **`Apply All`**.

6. You can now delete the object from the scene; the changes are saved to the prefab asset.

When you generate an imposter for this prefab, the tool will now use the `BillboardAnchor`'s position, guaranteeing a perfect pivot.







Notice how the anchor is in the very center.

5.3 Key UI Sections Explained

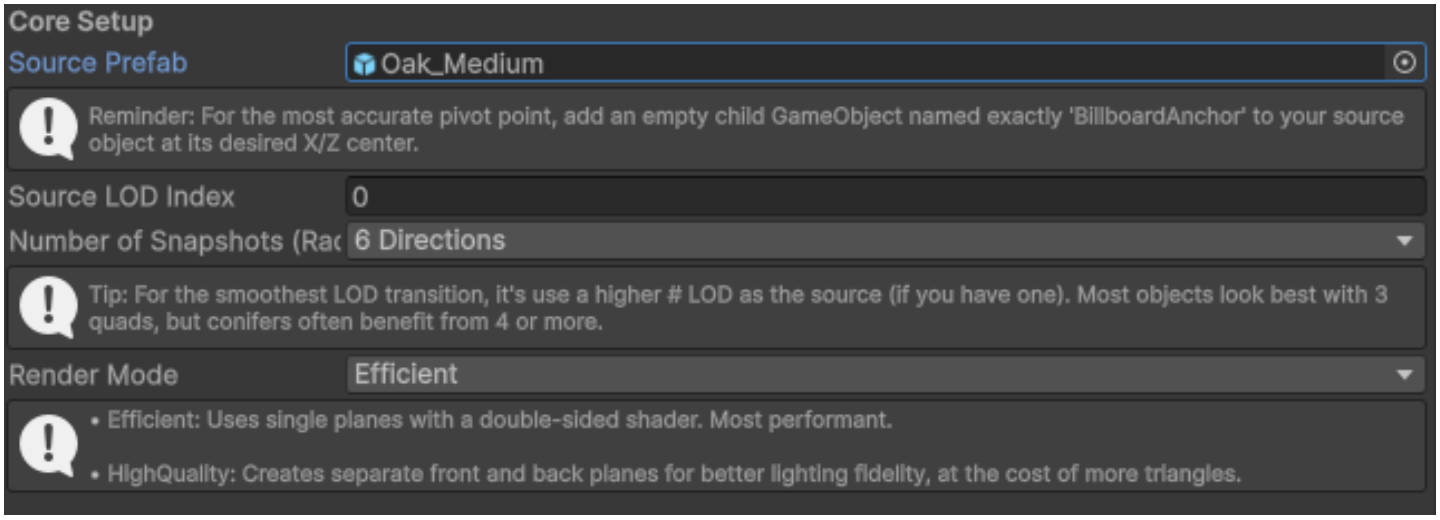
This section breaks down each part of the UI in detail. While the tooltips provide quick summaries, this guide offers a deeper explanation of the concepts behind the settings so you can make informed decisions.

5.3.1 Core Setup

This first group of settings defines the fundamental structure and quality of your imposter. Getting these right is the most important step.

- `Source LOD Index` This setting tells the tool which Level of Detail (LOD) mesh from your source prefab to use for the texture capture.
    - **Using LOD 0:** This is the default and most common choice. It uses your highest-quality model, resulting in the most detailed and accurate imposter texture.
    - **Using a Higher LOD (e.g., LOD 1 or 2):** Why would you use a lower-quality mesh? For performance and better transitions. Sometimes, the highest-detail model has tiny features (like individual twigs or single leaves) that don't capture well and create a noisy, aliased texture. Using a slightly simplified LOD mesh can result in a cleaner, smoother capture that actually looks better from a distance and transitions more seamlessly from the real 3D model.

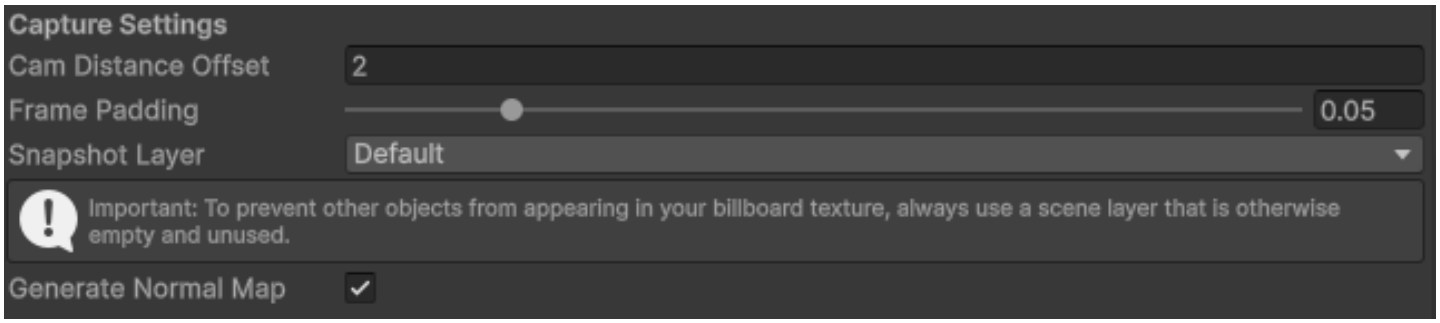
    - **Warning:** You must enter a valid LOD index that exists on your source prefab. If your model only has LOD0 and LOD1, entering `2` will result in an error. You cannot select the "Culled" state.
- **`Number of Snapshots (Radial)`** This determines the number of vertical planes your imposter will have, which dictates how smooth it looks as you rotate around it.
    - **`6 Directions` (3 Quads):** This is the sweet spot for most assets, especially broadleaf trees. It provides a significant visual improvement over a simple 2-quad cross, eliminating the "flat" look from diagonal angles, without adding excessive geometry.
    - **`8 Directions` (4 Quads) or more:** Fuller models like conifers often benefit from more planes to better represent their volume.
    - **More is not always better.** Each additional quad adds more triangles and increases texture generation time and file size. For most assets, going beyond 8 directions provides diminishing visual returns for the performance cost.
- **`Render Mode`** This is a critical choice that dictates how the mesh and materials are constructed, balancing performance against lighting fidelity.
    - **`Efficient` (Recommended Mode):**
        - **How it Works:** This mode creates a mesh with single, intersecting planes. It then captures only the "front-facing" textures (e.g., for 3 quads, it only captures 3 directions). The final material uses a **double-sided shader**, which tells the GPU to render both the front and back of each plane.
        - **Pros:** This is the most performant option. It uses the fewest triangles and requires the smallest texture atlas, saving both rendering time and memory.
        - **Cons:** Because a single texture is rendered on both sides of a plane, the lighting on the "back" side is technically incorrect (it's lit as if it were the front). However, for a fast-moving game, this is often completely unnoticeable.
        - **Verdict:** For 99% of use cases, **this is the mode you should use.**
    - **`HighQuality`:**
        - **How it Works:** This mode creates a mesh with separate front-facing and back-facing planes for each angle. It captures textures from all directions (e.g., for 3 quads, it captures all 6 directions). The final material is a standard, single-sided Lit shader.
        - **Pros:** This provides the most accurate lighting fidelity, as the back-facing planes use the correct texture captured from the back of the model.
        - **Cons:** This mode uses **twice as many triangles** and a significantly larger texture atlas compared to `Efficient` mode.
        - **Verdict:** This mode should be reserved for specific situations where absolute lighting accuracy on a distant object is critical, and you are willing to accept the performance trade-off.
- **`Resolution Index`** This dropdown controls the final resolution of your texture atlas.
    - **How it Works:** This setting defines the **height** of the texture in pixels. The width is calculated automatically based on the number of snapshots and the aspect ratio of your model. Even if the final atlas looks very wide, its vertical resolution is what primarily determines the sharpness of the imposter.
    - **Recommendation:** For most assets, **`512`** is a great balance of quality and memory usage. Use `1024` for important hero assets that might be seen closer to the camera.



5.3.2 Capture Settings

These settings control the virtual camera and rendering environment used to capture the imposter's textures. Think of this as adjusting the lens and setup for the "photo shoot" of your model.

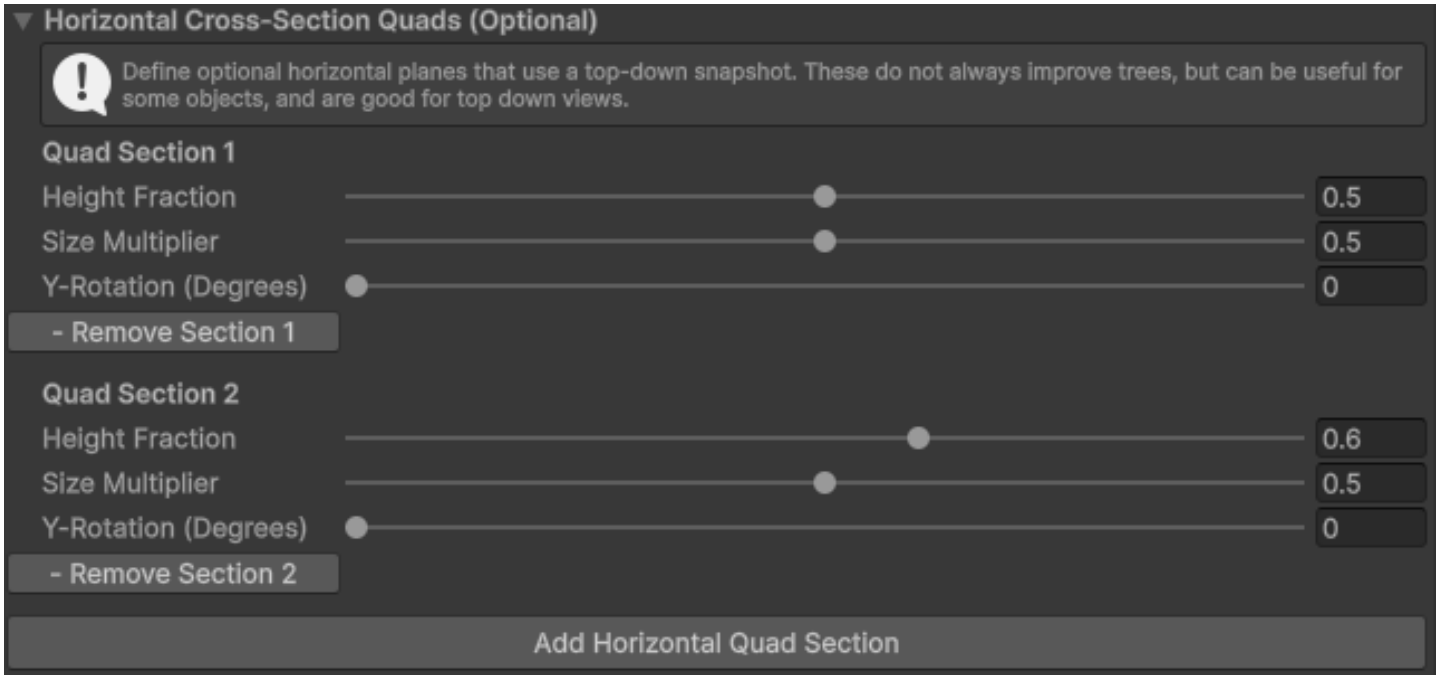
- **`Cam Distance Offset`** This value pushes the virtual camera further away from the object's calculated bounding box. The default value is usually sufficient, but if you have a model with unusual spikes or long, thin branches that get clipped off in the texture preview, **increasing this value** will move the camera back to ensure the entire model fits in the shot.
- **`Frame Padding`** This adds a small, transparent border around the captured image within its frame in the texture atlas. A small amount of padding is **highly recommended**. It helps prevent visual artifacts like "texture bleeding" or dark halos that can appear at the edges of the imposter planes due to texture filtering and mipmapping, especially at sharp viewing angles.
- **`Snapshot Layer`** This is a crucial setting for getting a clean capture. The tool works by temporarily moving your model to this specified layer and setting its internal camera to *only* see that layer. This isolates the model from the rest of your scene.
  The capture process itself takes place at the world origin `(0,0,0)`. If your scene is completely empty near the origin, you can often get away with using any layer. However, if you see **strange shapes, shadows, or other visual artifacts** in your generated texture previews, it's almost certainly because another object in your scene is on the same layer and is interfering with the capture.
  The solution is simple: Choose a dedicated, **unused layer** for this setting and regenerate the snapshot. Most Unity projects have several unused layers (e.g., Layers 8-31) available for this purpose.
- **`Generate Normal Map`** When enabled, this option performs a second capture pass to create a normal map, which allows your imposter to react realistically to dynamic lighting.
  Important: Normal maps are only used by **Lit shaders**. If your goal is maximum performance and you plan to use an `Unlit` or `Simple Lit` shader on your final imposter material, you should **un-check this box.** Generating a normal map in that case is an unnecessary use of texture memory and will have no visual effect.



5.3.3 Horizontal Cross-Section Quads (Optional)

This feature allows you to add one or more horizontal planes to your imposter. This is an excellent way to add top-down volume and substance, making assets like trees look much more convincing when viewed from above.

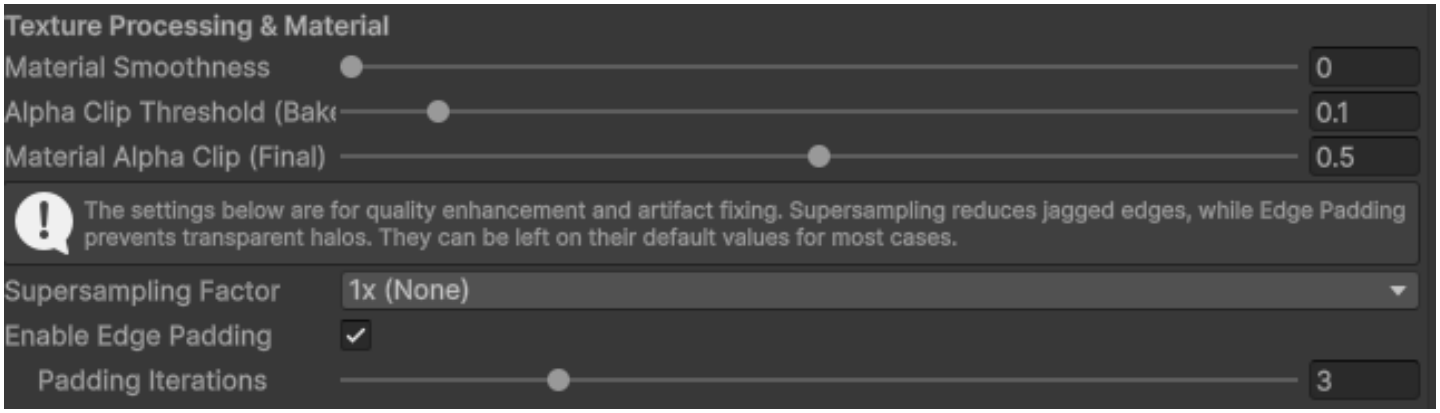
- **How it Works** When you add one or more horizontal sections, the tool performs **one single top-down capture** of your model. This one image is then mapped to all the horizontal quads you define.
- **Performance** Each horizontal plane you add is just a single, simple quad (2 triangles), so they have a very low impact on the final polygon count. The material for these quads is **always rendered two-sided**, so they are visible from both above and below, regardless of the main `Render Mode` setting.
- **UI Controls and Experimentation** You can add multiple sections, each with its own settings:
    - **`Height Fraction`**: Controls the vertical position of the quad on the imposter (0.0 is the bottom, 1.0 is the top).
    - **`Size Multiplier`**: Controls the width and depth of the quad relative to the average width of the vertical planes.
    - **`Y-Rotation (Degrees)`**: Rotates the quad around the vertical axis.
- **Getting these planes to look right often requires some experimentation.** We recommend starting with one quad, generating the full imposter, and viewing it in the scene. If it's not quite right, come back to the tool, adjust the sliders, use the **`Delete Last Generated Assets`** button, and regenerate. This iterative process is the best way to fine-tune the size and placement to perfectly match your model's canopy or horizontal features.



5.3.4 Texture Processing & Material

This section controls the properties of the final generated material and the post-processing effects applied to the captured textures to enhance their quality.

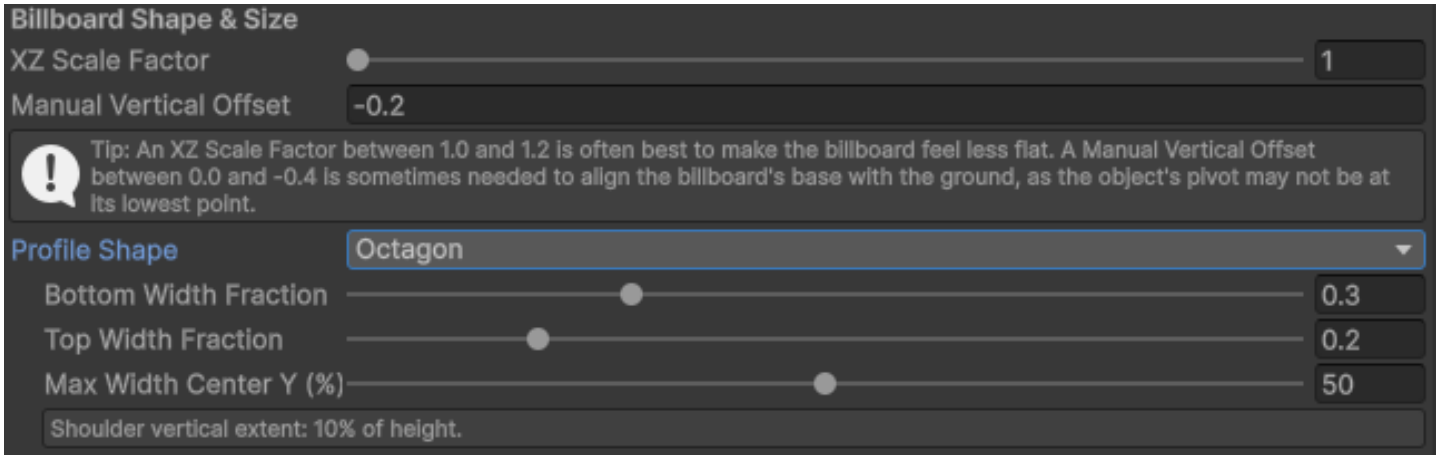
- **`Material Smoothness`** **&** **`Material Alpha Clip (Final)`** These two sliders are straightforward properties that are applied directly to the final material. **`Material Smoothness`** controls the glossiness of the imposter's surface, while **`Material Alpha Clip (Final)`** sets the transparency cutoff value for the final shader.
- **`Alpha Clip Threshold (Bake)`** This is a **pre-processing** step that cleans up the source texture *during* the capture. It discards any pixels from the original model that have an alpha value lower than this threshold. This is useful for eliminating noise or faint, semi-transparent areas from the source textures to create a cleaner, crisper imposter. The default value of **`0.1`** is a good starting point for most assets.
- **`Supersampling Factor`** Supersampling renders the snapshot at a higher resolution (2x or 4x) before downscaling it to the final size. The goal is to reduce aliasing (jagged edges) and produce a smoother result. While this can sometimes improve the quality on models with very fine, hard-surface details, for most organic assets like trees and bushes, the visual difference is often minimal. Leaving this at **`1x (None)`** is usually sufficient and results in the fastest generation time.
- **`Enable Edge Padding`** This is an essential quality feature that should almost always be left on. It extends the color of pixels at the edge of your imposter's silhouette into the surrounding transparent areas. This process is crucial for preventing ugly dark halos or seams from appearing around your imposter when viewed from a distance or at sharp angles, which is a common problem caused by texture mipmapping.



5.3.5 Billboard Shape & Size

These settings control the final dimensions and geometric profile of the imposter mesh itself, allowing you to fine-tune its presence in the world.

- `XZ Scale Factor` This slider uniformly scales the final generated prefab on its local X and Z axes, making the imposter appear wider and fuller.
    - **Why it's useful:** Due to the nature of capturing alpha-clipped foliage from a 2D perspective, imposters can sometimes feel slightly narrower or less voluminous than the original 3D model. Setting this to a value between **1.1 and 1.2** often helps the imposter's perceived volume better match the original.
    - **Note:** This only ever affects the horizontal (X/Z) dimensions. The Y dimension (height) is never changed by this setting.
- `Manual Vertical Offset` This is a critical adjustment for perfectly grounding your imposter. The tool automatically calculates the pivot based on the lowest point of the model's bounding box, but this isn't always perfect. Sometimes a model's origin isn't at its base, causing it to float, or a single stray root vertex might cause it to sink into the ground.
    - This slider lets you apply a final height correction (in meters). Based on our experience with a wide variety of assets, a small negative offset between **0.0 and -0.3** is often needed to get the placement just right.
- `Profile Shape` This choice determines the fundamental geometry of the vertical planes.
    - `Quad` **(Recommended):** This creates simple, rectangular planes. It's the most performant option because it uses the fewest triangles, and for most assets, it looks great.
    - `Octagon`**:** This option can yield a visually superior result. It creates an 8-sided mesh that you can customize to more tightly fit the silhouette of your model. When you select this, additional sliders appear, allowing you to control the taper. Typically, a tree should taper near its trunk (**Bottom Width Fraction**) and to a lesser extent at its top (**Top Width Fraction**), which you can configure here. This can reduce the amount of transparent area on your texture, which helps with overdraw performance, but it does use more triangles than the `Quad` profile.

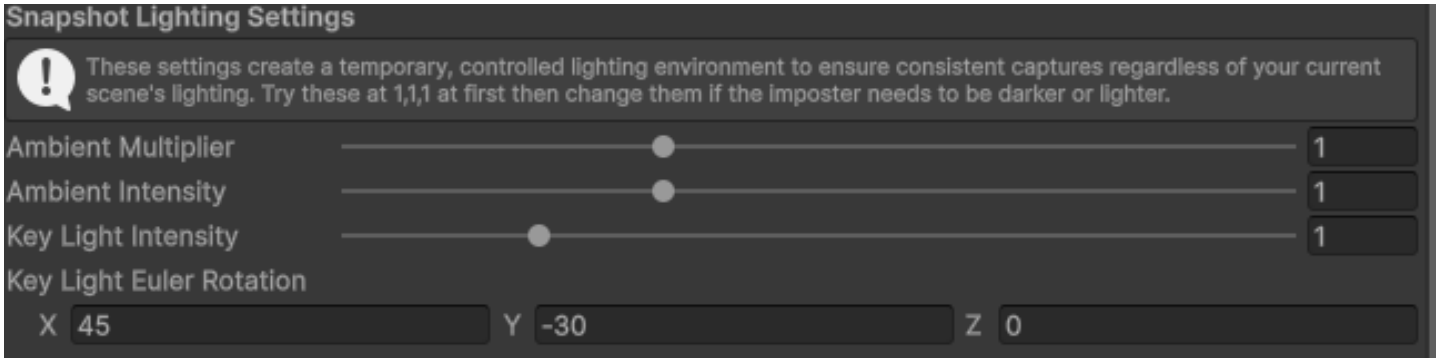


5.3.6 Snapshot Lighting Settings

This section gives you control over the temporary, controlled lighting environment used during the texture capture. This is crucial for ensuring your imposter's baked-in lighting matches the original model as closely as possible, regardless of your current scene's lighting.

Getting the lighting right can sometimes take a bit of iteration. Here's the recommended workflow:

1. **Start at Neutral:** For your first attempt, set all three intensity values (`Ambient Multiplier`, `Ambient Intensity`, `Key Light Intensity`) to **1**. This provides a good, neutral baseline.
2. **Adjust Brightness:** Generate your imposter and compare it to the original model.
    - If the imposter appears **darker** than the original, increase the **`Key Light Intensity`** and/or **`Ambient Intensity`** and regenerate.
    - If the imposter appears **lighter**, decrease these values.
3. **Adjust Angles:** If adjusting brightness alone isn't enough to match the look, the next step is to change the **`Key Light Euler Rotation`**. The Key Light acts as the "sun" in the capture scene. These rotation values control the angle from which it shines on your model. Changing the angle will change which parts of the model are in highlight and which are in shadow, which can dramatically alter the final captured texture.
4. **Last Resort - Tinting:** If the color or shade still doesn't quite match after adjusting the capture lighting, you can apply a tint to the final material. Select the generated material in your project folder and adjust its `Base Color` property. However, be aware that a tint can add color or make a texture darker, but it **cannot make it brighter**. It's often best to generate an imposter that is slightly too bright and then darken it with a tint if needed.



5.3.7 Output & LOD Settings

These final settings control how your assets are named and how they are integrated into Unity's Level of Detail (LOD) system.

- **`User Defined Base Name`** This field sets the text prefix for all the assets that are generated (the prefab, mesh, material, and textures).
- **`Create/Update LOD Group`** This is the master switch for the automatic LOD integration. When checked, it will either **create a new `LOD Group`** on a prefab that doesn't have one, or it will **insert the imposter** as the final level into an existing `LOD Group`.
- **`Cull Cutoff Percent`** This slider determines the screen height percentage below which the imposter will be culled (disappear completely). Because imposters are so incredibly efficient, you can afford to be generous and use a very small culling value. A value between **1-2%** is generally recommended, which allows your foliage to remain visible even at extreme distances with almost no performance cost.
- **`Billboard LOD Transition Percent`** This sets the screen height percentage at which the game will switch *from* the last 3D mesh *to* the generated imposter. Finding the perfect, seamless transition point often requires a bit of testing in-scene. For a detailed guide on the best way to compare your LODs and fine-tune this value, please refer to the workflow guides in **Section 4**.
- **`Destroy Source Scene Object`** This is a simple convenience toggle for when you are working directly with objects in your scene (rather than project prefabs). If you drag a scene object into the `Source Prefab` field, this option will automatically delete that original object from your scene after the imposter prefab is successfully created. **This will never delete a prefab from your project files.**

6. Use Cases & Best Practices

This section moves beyond the basic "how-to" and into the "how-to-do-it-well." Following these recommendations and workflows will help you produce high-quality, performant imposters consistently.

6.1 Common Use Cases

Veridian Imposters is a versatile tool that can fit into your production pipeline in several ways:

- **Adding Imposters:** Use it to create imposters for assets that were purchased or created without them.
- **Replacing Imposters:** If you have assets with old, low-quality, or inefficient billboards (e.g., simple 2-quad crosses), you can use this tool to generate superior replacements.

- **Rapid Iteration:** If you modify a source model (e.g., change its textures, colors, or shape), you can regenerate a new, perfectly matched imposter in seconds.
- **Creating a Base for Custom Shaders:** Since the tool generates materials with standard Unity shaders, they serve as a perfect, predictable starting point for you to extend with your own shader effects like wind, snow, or stylized lighting.

6.2 Quality and Performance: Key Recommendations

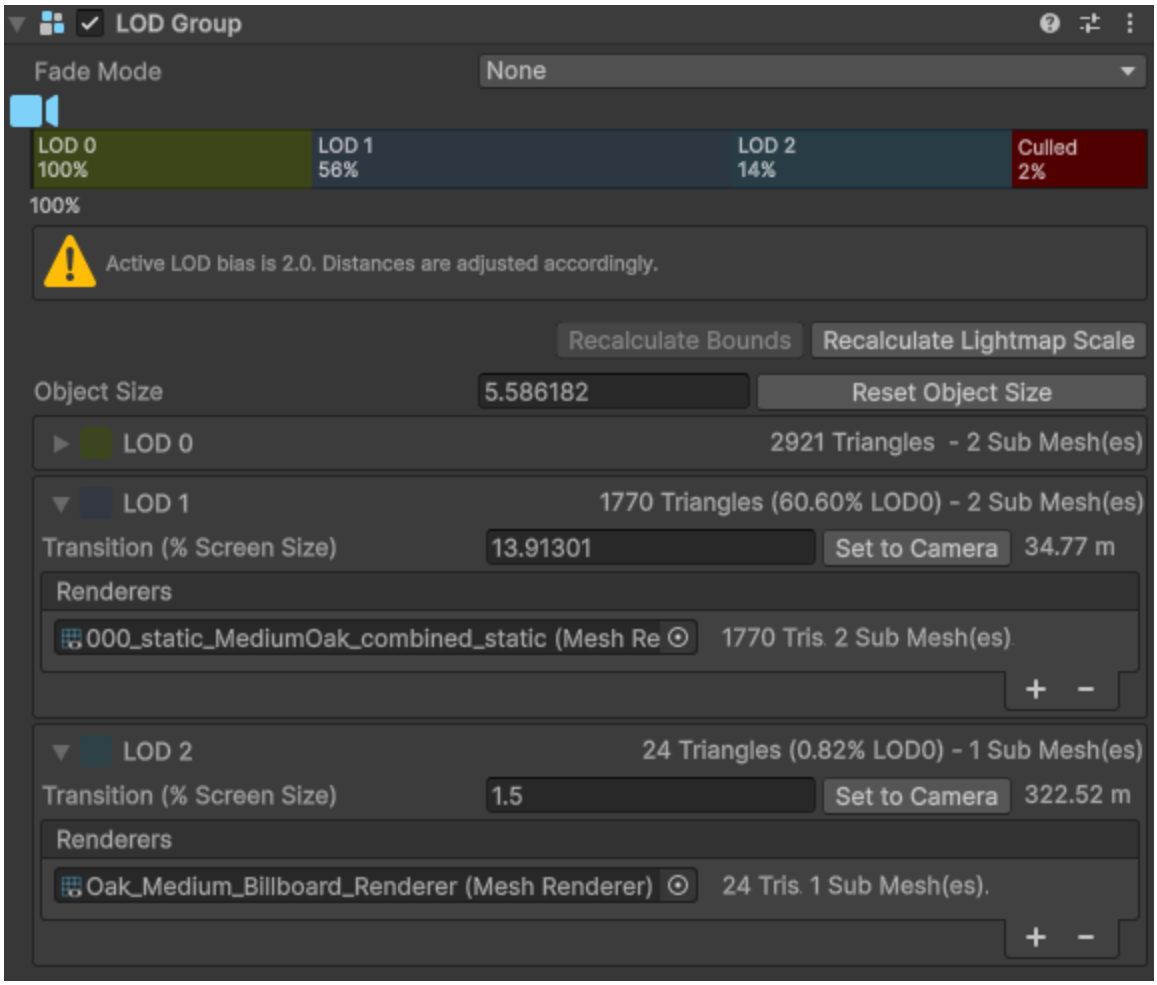
- **Texture Resolution:** In the UI, the `Resolution Index` setting controls the **height** of the final texture atlas. The width is calculated automatically. For most mid-ground assets, a resolution of **512** pixels is an excellent balance between sharp detail and memory efficiency. You may want to increase this to `1024` for hero assets.
- **Grass and Detail Objects:** When creating detail objects like grass or small bushes for terrain systems, the goal is to create a sense of volume. It is highly recommended to use **`6 Directions` (3 quads) or `8 Directions` (4 quads)**. This ensures the object looks good from all angles and doesn't appear as a flat, spinning card, which is a common issue with simpler grass billboards.

6.3 Best Practice: Comparing and Verifying Your Imposter

After generating a new imposter, it's crucial to verify that it's a convincing replacement for the original model, especially at the transition distance.

How to Compare Your LODs:

1. Find the newly generated prefab in your Project window and double-click it to open it in **Prefab Mode**. Alternatively, you can drag an instance into your scene.
2. Select the root object of the prefab in the Hierarchy. In the Inspector, you will see the **`LOD Group`** component, which displays a slider bar representing the different levels of detail.
3. You can **manually force a specific LOD to be visible** by dragging the white camera icon along this bar.
4. First, compare the imposter and the original up close. Drag the camera icon back and forth between the last mesh LOD (e.g., LOD0) and the new imposter LOD. Check for any major differences in color, lighting, or shape. This helps you confirm the capture was successful.
5. Next, position your Scene camera at a distance from the object that is similar to where the transition would naturally occur in-game (i.e., where the object takes up the percentage of screen height you defined in the **`Billboard LOD Transition Percent`** setting).
6. Once again, drag the LOD slider back and forth. At this distance, the transition should be barely noticeable—a slight "pop" is acceptable, but there should be no dramatic shift in color, brightness, or silhouette. If the transition is too jarring, consider adjusting your **Snapshot Lighting Settings** and regenerating the imposter.



6.4 Best Practice: Using Presets for a Consistent Workflow

Veridian Imposters uses **ScriptableObjects (`.asset` files)** to store all of your settings. This is an incredibly powerful feature for maintaining consistency and speeding up your workflow.

- **How it Works:** The first time you use the tool, you should click **`Create and Assign New Settings Asset`**. This creates a `BillboardSettings` file in your project. Every change you make in the UI—from resolution to lighting to LOD percentages—is **automatically saved** to this settings asset in real-time.
- **Creating and Reusing Presets:** You can create multiple settings assets for different types of models. For example, you could have:
    - `Conifer_Tree_Settings.asset` (using 4 quads and a tapered Octagon profile).
    - `Broadleaf_Tree_Settings.asset` (using 3 quads and a standard Quad profile).
    - `Grass_Settings.asset` (using 3 quads and a horizontal plane).
- When you want to make an imposter for a new conifer, you simply drag your `Conifer_Tree_Settings.asset` into the "Settings Asset" field in the UI, and all of your preferred settings will be loaded instantly.
- **Demo Presets:** The included Demo folder contains the exact `BillboardSettings` asset files that were used to create the example imposters. You can load these into the tool to see how each demo asset was configured. This is a great way to learn how different settings affect the final outcome.

7. Limitations: What Veridian Imposters Isn't

Veridian Imposters is a specialized tool designed to solve a specific set of performance problems. Understanding what it is *not* designed to do is just as important as knowing what it can do. Please review these limitations to ensure the tool is a good fit for your specific needs.

No Dynamic Wind Animation

The generated imposters are **static meshes**. They are effectively "photographs" of your original model, baked into a texture at a single moment in time.

- **What this means:** Imposters will not react to Unity's **Wind Zones** or any other form of vertex animation. If your original tree sways in the wind, its imposter will remain perfectly still.
- **The reason:** The texture capture process does not record animation data. To simulate wind on an imposter, a custom shader that procedurally distorts the mesh's vertices would be required. This is outside the scope of this tool, which focuses on providing maximum compatibility and performance by using Unity's standard, built-in shaders.

Object Shape and Symmetry are Key

Veridian Imposters produces the best results on objects that are roughly **radially symmetrical**, meaning they look reasonably similar from all horizontal viewing angles. Classic single-trunk trees and round bushes are perfect candidates.

The tool may produce poor or undesirable results for:

- **Highly Asymmetrical Models:** A tree that leans heavily or has all its major branches on one side will not be represented well from all angles.
- **Clusters or Multi-Trunk Models:** The tool calculates a single pivot point for the entire object. For a cluster of trees, this pivot will be in the empty space between them, leading to unnatural rotation. It is much better to generate imposters for each tree individually and then assemble them into a cluster.
- **Objects with Concave Gaps:** Models with large, hollow areas may not capture correctly, as the planes of the imposter might slice through these gaps in a visually jarring way.

Lighting is Baked onto the Texture

While the generated normal maps allow the imposter to react to **dynamic lights** in your scene (like a moving sun or a flashlight), the general ambient light and subtle shadows from the object onto itself are **baked** into the texture during the generation process.

- **What this means:** An imposter generated under bright, neutral lighting will look most natural in similarly lit environments. If you place that imposter in a scene with radically different ambient lighting (e.g., a dark blue cave), the baked-in lighting on the texture may not match perfectly.

- **Best Practice:** Use the **Snapshot Lighting Settings** in the UI to create imposters with neutral, diffuse lighting. This makes them more versatile and able to blend into a wider variety of environments.

Performance is an Optimization, Not Free

Imposters provide a massive performance boost, but they are not without their own costs.

- **Texture Memory:** Every unique imposter you generate adds new textures to your project. A `1024x1024` texture atlas can consume 4MB or more of memory. Instead of creating a unique imposter for every single tree in your scene, it is far more efficient to create a library of a few variations and reuse them.
- **Overdraw:** Imposters, by their nature, have significant transparent areas on their textures. When many imposters overlap on screen, the GPU still has to process those transparent pixels, a problem known as overdraw. While Veridian Imposters' `Octagon` profile can help mitigate this, it's a fundamental trade-off of the billboard technique.

Requires a Source Asset

Veridian Imposters is a utility for processing and optimizing existing models. It is **not a content creation tool**. You must always provide a pre-existing 3D model in a prefab format to serve as the source for the imposter generation.

---

8. Scripts & Architecture Overview

This final section provides a high-level overview of the asset's internal architecture. It is intended for advanced users who may be curious about how the tool works or are considering modifying its behavior.

For the most detailed, line-by-line understanding, we highly encourage you to open the scripts themselves. The code is heavily commented with XML documentation summaries for nearly every class and method, explaining the purpose of each function and variable. This section will serve as a guide to how all the pieces fit together.

The system is designed with a clear separation of concerns, broken into four primary components:

1. The User Interface (`BillboardSnapshotWindow.cs`)
2. The Settings Asset (`BillboardSettings.cs`)
3. The Service Logics (`TextureGeneratorService.cs`, `BillboardMeshService.cs`)
4. Supporting Data Structures

8.1 The Most Important Script: `BillboardSnapshotWindow.cs`

The `BillboardSnapshotWindow.cs` script defines an `EditorWindow`, which is Unity's class for creating custom tool windows like this one. It serves three primary roles:

1. **Drawing the User Interface (UI):** It creates every button, slider, and field you interact with.
2. **Managing State:** It holds the current settings and the data generated between steps.
3. **Orchestrating the Services:** It acts as a "manager," telling the specialized "worker" services (`TextureGeneratorService` and `BillboardMeshService`) what to do and when to do it.

Role 1: Drawing the User Interface

The entire UI is drawn and managed by one core Unity method and a helper method within the script.

- **`OnGUI()`**: This is a special method provided by Unity that is called multiple times per second whenever the editor window is visible. Its job is to draw the interface and respond to user input like clicks and value changes. In our asset, `OnGUI()` is kept very simple; it sets up a scroll view and then calls `DrawFullUI()`.
- **`DrawFullUI()`**: This large helper method contains the actual code for every UI element. It uses Unity's `EditorGUILayout` class to draw each component:
  - `EditorGUILayout.ObjectField(...)` is used for the "Source Prefab" and "Settings Asset" fields.
  - `EditorGUILayout.Popup(...)` creates the dropdowns for "Number of Snapshots" and "Resolution".
  - `EditorGUILayout.Slider(...)` and `EditorGUILayout.FloatField(...)` create the various sliders and number fields.
  - `GUILayout.Button(...)` creates the main action buttons.
- Crucially, every UI element is linked to the `activeSettings` (`BillboardSettings`) ScriptableObject. When the window is drawn, each slider gets its value *from* the settings asset. When you change a value, the script uses `EditorGUI.BeginChangeCheck()` and `EditorGUI.EndChangeCheck()` to detect this change and immediately saves the new value back *to* the settings asset, ensuring your configuration is always preserved.

Role 2: Orchestrating the Generation Workflow

The window script manages the entire step-by-step process. It doesn't generate textures or meshes itself; it delegates those complex tasks to the services.

Step 1: Texture Generation

- **Primary Method: `GenerateSnapshotData()`** This method is called when you click the **`1. Generate Snapshot Atlas Data`** button. It performs the following sequence:
  1. **Validation:** It first checks that a `sourcePrefab` has been assigned in the UI.
  2. **Bounds Calculation:** It calls an internal helper method, `CalculateObjectBounds()`, to measure the 3D model. This is a critical step to ensure the capture camera is positioned correctly. This is also where it checks for and uses the position of a `BillboardAnchor` child object for perfect pivot placement.
  3. **Package Settings:** It gathers all the relevant settings from the UI (resolution, lighting, normal map preference, etc.) and packages them into a `TextureGeneratorSettings` data object.

4. **Call the Service:** It calls the main method of the `TextureGeneratorService`, which is `GenerateAtlasAndSnapshots()`. It passes the packaged settings and the calculated bounds to the service.
5. **Cache the Result:** The service returns a `GeneratedTextureData` object, which contains the newly created `Texture2D` atlases and other metadata. The window script stores this object in a local variable, `lastGeneratedSnapshotData`. This cached data is then used to draw the texture previews in the UI and is held in memory for the next step.

Step 2: Final Asset Creation

- **Primary Method:** **`CreateBillboardAssetsFromGeneratedData()`** This method is called when you click the **`2. Create Billboard Assets`** button. It uses the cached data from the previous step to build the final assets.
    1. **Validation:** It first ensures that the `lastGeneratedSnapshotData` exists.
    2. **File Management:** It determines the correct output path and creates a new, descriptively named folder next to your source prefab asset.
    3. **Save Textures & Material:** It saves the cached textures to disk as `.png` files and creates a new `.mat` material file, assigning the new textures to it.
    4. **Call the Mesh Service:** It gathers the necessary data (including the cached texture data and shape settings) into a `MeshGenerationSettings` object. It then calls `GenerateBillboardMeshData()` on the `BillboardMeshService` to receive the final, constructed `Mesh`.
    5. **Save the Mesh:** It saves the returned `Mesh` object as a `.asset` file.
    6. **Assemble the Prefab:** In the final and most critical stage, this method does the assembly. It creates a new, empty `GameObject`, adds `MeshFilter` and `MeshRenderer` components, and assigns the newly created mesh and material. It then uses a helper method, `CreateLODPrefab()`, to handle the complex logic of creating or updating the `LOD Group`.
    7. **Save the Prefab:** Finally, it saves this fully assembled object as the final `.prefab` asset in your project using `PrefabUtility.SaveAsPrefabAsset()`.

By delegating the complex logic to services, the `BillboardSnapshotWindow` script remains focused on its core jobs: UI and coordination, making the entire system clean and maintainable.

Excellent. This is a crucial part of the architecture to understand, as it separates the *data* from the *logic*. Here is the expanded section covering both the ScriptableObject settings and the data transfer scripts.

8.2 The Virtual Photographer - `TextureGeneratorService.cs`

This script's job is to take high-quality "photographs" of your 3D model from every required angle. It's a specialized worker that's called by the main `BillboardSnapshotWindow` when you begin the generation process.

Its primary method is **`GenerateAtlasAndSnapshots()`**. When called, this method performs a sequence of automated tasks:

1. **Scene Setup:** It temporarily hides your scene's lighting and creates a controlled, temporary environment with its own lights and camera. This ensures every imposter is captured with consistent, clean lighting.
2. **Material Swapping:** To capture the normal map, it uses a helper method, **`ApplyNormalCaptureMaterials()`**, to temporarily replace your model's materials with a special internal shader that visualizes world-space normals. After the capture, it uses **`RestoreOriginalMaterials()`** to put everything back exactly as it was.
3. **Iterative Capture:** It loops through each required angle (e.g., 6 times for a 3-quad imposter), positioning its internal camera to frame the object perfectly for each shot. Each of these captures is handled by the **`CaptureAndProcessSingleView()`** method.
4. **Post-Processing:** After capturing each image, it processes the raw texture data. It performs alpha clipping and applies a multi-pass **edge padding** algorithm to prevent transparent halos on the final texture.
5. **Atlas Stitching:** Finally, it takes all the individual processed images and stitches them together side-by-side into a single large texture atlas for both the Albedo and Normal maps. This final `GeneratedTextureData` object, containing the atlases and metadata, is then returned to the main window.

8.3 The 3D Architect - `BillboardMeshService.cs`

Once the textures are created, this script's job is to build the actual 3D mesh that the textures will be applied to. It takes the dimensional data calculated by the `TextureGeneratorService` and the shape settings from the UI to construct the geometry.

Its primary method is **`GenerateBillboardMeshData()`**. This method builds the mesh from scratch:

1. **Data Initialization:** It creates empty lists to hold the mesh data: `vertices`, `triangles`, and `uvs`.
2. **Plane Generation:** It iterates through each of the required radial views (e.g., 3 times for a 3-quad imposter). For each view, it calculates the positions of the vertices needed to create a plane (a simple `Quad` or a tapered `Octagon`, depending on your settings).
3. **UV Mapping:** This is the most critical step. For each vertex it creates, it also calculates its corresponding UV coordinate. It maps the vertex's position to the correct rectangular region within the texture atlas that was generated for that specific viewing angle. This ensures the correct image is displayed on the correct plane.
4. **Triangle Definition:** It defines the triangles by creating lists of indices that connect the vertices in the correct order to form a visible surface. For `HighQuality` mode, it generates a second set of back-facing triangles.
5. **Final Assembly:** After all planes (including any optional horizontal quads) are generated, it assigns the completed lists of vertices, triangles, and UVs to a new `Mesh` object and returns it to the main window to be saved as the final `.asset` file.

8.4 Data Management: Settings & Transfer Objects

Unlike the scripts that control the editor window and services (which live in an `/Editor` folder), the scripts that handle data are located in the main `/Scripts` folder. This is because this data needs to be accessible by Unity at all times, not just when the editor tool is open. This section covers the two key types of data scripts: the persistent settings asset and the temporary data containers.

`BillboardSettings.cs` and What a ScriptableObject Is

This script is the key to the powerful preset system. To understand it, you first need to know what a ScriptableObject is.

- **What is a ScriptableObject (SO)?** Think of a ScriptableObject as a custom data container that you can save as an `.asset` file in your Unity project. While a normal script (`MonoBehaviour`) needs to be attached to a GameObject in a scene, a ScriptableObject asset exists on its own in your Project window, just like a material or a texture. Its purpose is simply to hold data.
- **The Role of `BillboardSettings.cs`** The `BillboardSettings.cs` script is the **blueprint** for our settings assets. The script itself just defines a list of all the possible variables you can change in the UI—`MaterialSmoothness`, `NumSnapshotsIndex`, `CullCutoffPercent`, etc.
  When you click the **`Create and Assign New Settings Asset`** button in the UI, you are creating a new `.asset` file based on this blueprint. This new file (e.g., `MyTreeSettings.asset`) is what actually stores the values from the sliders and checkboxes. The `BillboardSnapshotWindow` then reads from and writes to this active `.asset` file, making the entire preset system possible.

`BillboardData.cs` — The Data Intermediary ✉️

This script is a collection of simple classes and structs that act as "Data Transfer Objects" (DTOs). Their sole purpose is to bundle up information neatly to pass it between the other major scripts. This keeps the code clean and organized.

- **The Role of a Data Transfer Object** Think of the main window script as a manager and the service scripts as specialized workers. Instead of the manager shouting 20 separate instructions to a worker, it's more efficient to write all the instructions down on a single "Work Order" form and hand that form to the worker. That's exactly what these data objects do.

Here are the key "forms" used by the system:

- **`TextureGeneratorSettings`**: This is the work order passed from the `BillboardSnapshotWindow` **to** the `TextureGeneratorService`. It contains all the settings the texture service needs to do its job, such as the resolution, lighting settings, camera padding, etc.
- **`MeshGenerationSettings`**: This is the work order passed from the `BillboardSnapshotWindow` **to** the `BillboardMeshService`. It contains the parameters needed to build the mesh, like the chosen `BillboardProfile` and the object's dimensions.
- **`GeneratedTextureData`**: This is the "Completed Work Report" passed **from** the `TextureGeneratorService` back to the main window. After the textures are created, they are bundled up into this object, which the main window then caches and uses for the texture previews and for the final asset creation step.

By using these intermediary data containers, the system avoids having to pass dozens of individual parameters between methods, making the entire architecture much more robust and easier to read.

8.5 Internal Shaders: Capturing Normals

A key feature of Veridian Imposters is its ability to generate true normal maps, which allow your imposters to react realistically to dynamic lighting. This process is handled by a set of internal, hidden shaders.

The Shader Files (`.shader` and `.hlsl`)

In the asset's shader folder, you will find files for each render pipeline (e.g., `NormalCapture_URP.shader`).

- **Role:** These are not materials for your final imposters. They are specialized, internal shaders used exclusively by the `TextureGeneratorService` during the texture capture phase.
- **How They Work:** A `.shader` file is Unity's wrapper that defines properties and structure. It points to a `.hlsl` (High-Level Shading Language) file, which contains the actual GPU code. While this code could be created visually with Shader Graph, it was written by hand for this asset.
- **Pipeline-Specific Versions:** Each render pipeline (Built-in, URP, HDRP) has its own unique rendering architecture. Because of this, a separate version of the normal capture shader is required for each pipeline. The tool automatically detects which pipeline your project is using and calls the correct corresponding shader.
  Tip: If you know for certain that your project will only ever use one render pipeline (e.g., you are only developing for URP), you can **safely delete the shader files for the other pipelines** to save a small amount of space in your project.

The Superiority of True Normal Capture

The purpose of these shaders is to "bake" the 3D surface information of your original model into a 2D texture.

- **The Process:** When capturing the normal map, this shader temporarily overrides the rendering of your model. Instead of calculating lighting, for every pixel it renders, it outputs the direction of the model's **world-space normal** as an RGB color. This captures the true geometric detail—every bump, groove, and angle from the source mesh.
- **Why It's Better:** Many simpler tools or online converters try to generate a normal map from a standard color (Albedo) texture. This is fundamentally a guess; it infers depth from brightness or color changes. The result is often a fake-looking map that doesn't accurately represent the object. Veridian Imposters captures the **actual surface data**, resulting in a far superior normal map that reacts to light exactly as the original 3D model would.

Limitations and When to Skip Normals

- **Limitations on a Flat Surface:** A normal map creates the *illusion* of depth on a surface, but it cannot change the actual geometry. The silhouette or edge of the imposter's quad will always be a perfectly straight line. The normal map makes the flat surface *look* bumpy and detailed, but it won't make the edges look rough or organic.
- **When You Don't Need Normals:** Normal maps are specifically for use with **Lit shaders** to achieve realistic lighting, shadows, and reflections. If your goal is maximum performance and you plan to use a `Simple Lit` or `Unlit` shader on your final imposter, then generating a normal map is an unnecessary

use of texture memory and generation time. In this case, you should **un-check the `Generate Normal Map`** box in the UI before you begin the process.

9. Future Plans: The Veridian Systems Ecosystem

Thank you for being a part of the Veridian Systems community. **Veridian Imposters** is the foundational first step in our long-term vision to create a suite of powerful, intuitive, and seamlessly integrated tools for building vast and beautiful worlds in Unity.

We are already hard at work on the next components of this ecosystem. Here is a preview of what's coming soon.

Immediate Next Steps: Free Extensions for Veridian Imposters

To further enhance the performance gains from this asset, we will soon release two new, completely **free** tools designed to work directly with the imposters you create.

- **Object & Imposter Fuser:** While imposters drastically reduce vertex counts, scenes with thousands of them can still be limited by draw calls. This utility will allow you to fuse large groups of imposter prefabs into a single, combined mesh. This will dramatically reduce draw calls, allowing you to render even denser forests and fields with incredible efficiency.
- **Terrain Imposter Baker:** This tool will enable you to take the next leap in large-scale world optimization. It will give you the ability to "bake" entire forests of imposters directly onto the mesh of distant terrains. This is the ultimate technique for creating convincing, richly forested mountains and far-off landscapes that have a near-zero performance cost, extending the visual fidelity of your world to the horizon.

The World-Building Suite: Context-Aware Environment Tools

Beyond imposters, our next major focus is on a more intelligent, artist-driven approach to world creation. Instead of purely procedural generation that often feels random, our tools are designed to learn from and expand upon your existing work, creating natural, cohesive environments with minimal effort.

- **Contextual Terrain Generator:** This is not just another tool for generating terrains from scratch. Its unique strength lies in its ability to interact with existing landscapes.
    - It will feature robust **edge blending** to seamlessly match the height of adjacent terrain tiles.
    - It can **sample the height profile** of one or more neighboring terrains to generate a new tile that is stylistically similar, ensuring your world feels continuous. It can even blend the characteristics of two different biomes to create a smooth, natural transition.
- **Biome-Aware Object Placer:** This powerful tool moves beyond simple random scattering.
    - It begins by **sampling a source terrain** to learn the distribution rules of your objects. It analyzes how you've placed trees, rocks, and grass based on factors like **height, slope, density, and clustering**.
    - It then uses these learned rules to populate new terrains. The placement is driven by a **Metropolis algorithm**, which iteratively seeks a stable, natural-looking state based on the

weights you assign to each factor. This is not a simple time-based growth simulation, but a sophisticated sampling process to find the most believable distribution for your biome. You can even watch the fascinating "evolution" of the placement as the algorithm settles.

The Ultimate Vision: The Veridian World Suite

The individual terrain and placement tools may be released in a limited capacity as free assets or as standalone paid assets. Ultimately, all of these systems—the Imposter Generator, the Fuser, the Terrain Generator, and the Object Placer—will be fully enhanced, integrated, and bundled into a single, comprehensive, and professionally supported **paid asset:** the **Veridian World Suite**.

Our goal is to provide incredible value to the community with our free releases while building towards a complete, professional solution for serious world-builders. We are excited for you to join us on this journey.