

Drop-In Terrain Renderer: GPU Trees & Grass

Documentation · v1.0.0 · Havoila Studio

Drop-In Terrain Renderer takes over drawing a Unity Terrain's trees and grass (GPU culling, LOD selection and indirect draws), so the main thread stops paying for every tree. The terrain stays the source of truth: paint with Unity's tools as usual, and remove the component to get Unity's own rendering back.

The full documentation is at havoila.com/assets/drop-in-terrain-renderer/docs (Unity 6.3 or newer, URP).

Contents

1. **Quick start**
2. **The demo**
3. **The Inspector**
4. **Compatibility**
What was tested · Render paths, platforms and URP features · Your own shaders · SpeedTree · Drop-In Leaves, Bark and Grass · Vendor packs · Grass streaming and garbage · Generated shader copies · Reflections · Known issues · How this was tested
5. **Quality levels and options menus**
6. **Why does Unity draw my trees?**
7. **Did it help? Measuring your scene**
Reading the report · A small or negative result · Measured results · Not measured
8. **Stats overlays**
9. **Grass interaction**
10. **Support**

Quick start

1. **Set up.** Select your terrains (or their parent) and choose `Tools > Havoila Studio > Drop-In Terrain Renderer > Set Up Selected Terrains`, or `Set Up Every Terrain In The Open Scenes` in the same menu. The Terrain component's menu has it too (`Set Up Drop-In Terrain Renderer`), and so does the Set Up button of the Drop-In Terrain Renderer Inspector.
2. **Read the summary.** When the setup is done, the Last setup box in the Drop-In Terrain Renderer Inspector of a terrain it set up (select one of them: clicking the Console line highlights the first in the Hierarchy, and the setup highlights it too when none of its terrains is selected) says what Drop-In Terrain Renderer draws, what stays Unity's and why, and the impostor memory written.
3. **Measure.** Press Measure in Play Mode in the box to see what it changed (see "Did it help? Measuring your scene").
4. **Impostors (optional).** Impostors are flat cards drawn for far trees: past a LOD Group's last LOD, where Unity draws nothing, or in place of a tree's mesh from a set screen height when it has no LOD Group. With your own shaders (Keep mine, the default), the first setup bakes no impostors: the summary says `Impostors: 0 baked` and why, with how many tree kinds can have one after Convert. "Convert those kinds and bake" (when the box offers it) converts them to Drop-In Leaves and Bark, which changes their look near and far (`Look > Keep mine` takes a kind back), then bakes them. Measure again afterwards: the verdict then notes that the two sides differ, for example `Drop-In Terrain Renderer draws more than Unity here, with other shaders` (see Compared as set).

What Set Up does. It runs in Edit mode, in short steps across editor updates, so the editor stays responsive:

- it removes the copies of shaders no longer in the project (see Copies left behind, under Generated shader copies);
- it adds the Drop-In Terrain Renderer component to each terrain that has none. This is its one undo step, "Set Up Drop-In Terrain Renderer": Undo removes the components it added;
- it patches each terrain's tree and detail materials, one terrain at a time;
- it waits until Drop-In Terrain Renderer has built the terrains' draw data once (its own build, not a player build);
- it bakes the impostors the tree kinds can have, at the Auto size (64 px frames for an impostor that starts below 1.5% of the screen's height, else 128 px: about 0.8 or 3.3 MB per tree kind);

- it writes the summary: one Console line and the Last setup box.

Inactive terrains and terrains without Terrain Data are named in the summary and not set up. An undo leaves the generated copies and impostors in Assets/Drop-In Terrain Renderer Generated (Patch Materials records no undo for them either). While it runs, the Inspector shows its progress (for example `Setting up: patched 2 of 12 terrains.`, then the build wait and the bakes) with Cancel: what is done stays, and the bakes it queued are cancelled. Entering Play mode stops it too. Set Up is unavailable in Play mode, while scripts compile, while impostors bake and while another setup runs. During a first setup and right after it, Unity-style grass (grass textures, and detail meshes with GPU instancing off) can read `is being adapted` for about 15 to 30 s: its copies are compile-checked in the background, one variant per editor update, and Unity draws those terrains' details (all their layers) until the check passes.

The summary is one paragraph: how many terrains, their trees and details drawn by Drop-In Terrain Renderer or by Unity (each with its reason), the impostors, what Undo takes back, and the next step. A first setup on one terrain whose four tree kinds use URP Lit reads, for example:

Drop-In Terrain Renderer set up on 1 terrain. Trees: ours. Details: ours. Impostors: 0 baked. Your trees keep their own shaders (Look: Keep mine), and impostors need Drop-In Leaves and Bark: 4 kinds can have them after Convert ("Convert those kinds and bake" below). Undo removes the component; generated copies stay in Assets/Drop-In Terrain Renderer Generated. Next: Measure in Play Mode to see what it changed.

When kinds qualify, the impostor part reads `Impostors: N kinds baked, X MB written (Auto size)` instead. It also counts the kinds that cannot have one, and why: kinds Convert cannot take (their Look rows say why; no impostors for them in version 1), kinds under Impostor Min Height and kinds that keep their SpeedTree billboard as their far LOD. Failed bakes and other kinds that cannot have one are named, with the reason. Kinds planted only where the Impostors switch is off are counted as your choice, with the way to include them (turn the switch on in the Impostors section). Copy adds each kind's own state, and the Look and Impostors rows name them.

The Last setup box stays for the editor session, in the Inspectors of the terrains it set up. Copy takes the summary with its details (the time each step took, each tree kind's state, the patcher's reasons); Dismiss hides it until the next setup. Its buttons:

- **Measure in Play Mode** enters Play mode, measures Unity against Drop-In Terrain Renderer from your Game camera, leaves Play mode and opens the report window (the Benchmark section's header shows the verdict). It is unavailable while impostors bake: Play mode would cancel them. After a setup with nothing to draw (no trees or details on the terrains, or Draw Trees and Draw Details off), the summary's next step is to paint them first, since Measure would find nothing to compare.
- **Convert those kinds and bake** (when kinds can have an impostor once converted) asks first, naming the kinds, then converts them to Drop-In Leaves and Bark and bakes their impostors. Their look changes, near and far: `Look > Keep mine` takes each kind back (one undo step per kind). It is a second click on purpose, because the look changes.
- **Bake the rest** (after a setup cut short) bakes the impostors these terrains' kinds can have and do not have yet. After a script reload, the box reads `Interrupted by a script reload: ...` with this button.

The demo

`Demo/DropInTerrainRenderer_Demo.unity` is one real valley (Big Meadows, Rocky Mountain National Park, from USGS lidar): a 2 km Unity terrain with about 290,000 trees, snags, logs and rocks (firs and pines with four levels of detail each) and 8.2 million grass clumps, ferns, flowers and shrubs painted, inside a 12 km terrain with about 232,000 more trees on its slopes. The valley terrain's Detail Density is 0.7 and all of its detail layers are Affected by Density Scale, so about 70 % of its grass and plants are placed, by Unity's terrain and by Drop-In Terrain Renderer alike (set it to 1 for all of them). One Drop-In Terrain Renderer component on each terrain is all it takes. Press Play:

- `Space` switches Drop-In Terrain Renderer off and on. Off, Unity's terrain draws the same trees and grass: same prefabs, same LOD Groups and cross-fade, same materials (GPU instancing on), the same Light Probe Groups.
- `D` steps the forest density (100, 50, 25 % on both terrains; trees only, the terrain assets are never written), to find how much forest Unity's terrain affords at the frame rate Drop-In Terrain Renderer holds at 100 %.

- **G** switches the grass between Extended (out to 500 m, thinned past 40 m) and Match Unity (Unity's 250 m: like for like). Measure in Match Unity for a like-for-like number.
- **M** runs Measure (the report window opens at the end in the editor; a development build shows the benchmark panel and writes its report). **Esc** cancels.
- **F8** the stats overlay (top right: what is on screen, the demo's planted counts and Unity's batches; **Show details** unfolds the rest), **H** the key list (hidden at start), **1-9**, **0** the viewpoints (the camera starts at **7**), **T** a tour, right mouse + **WASD/QE** to fly (**Shift** 3x faster; the mouse wheel, or the Demo Camera's Speed in the Inspector, sets the speed at once). The frame rate and the frame time show under the ON/OFF title at the top; the view's settings (forest density, grass mode, 60° LOD distances) and the last Measure in a bar at the bottom.
- The deer in the meadow about 150 m ahead of the start view press the grass where they walk and graze (a Foliage Interactor each, see Grass interaction), and pay the camera no attention: fly up close to see it. The grass bends while Drop-In Terrain Renderer is on (Unity's grass does not bend). Each deer's Behaviour (its Demo Deer component) sets what it does: Free (as shipped) lets a deer choose for itself; Graze keeps it grazing, Look keeps it looking round, and Walk walks it straight ahead until its herd's map edge, where it stands looking.
- The 35 mm viewpoints keep a 60° lens's LOD distances (the bar at the bottom says "60° LOD distances"): the demo sets the terrains' Tree LOD Bias Multiplier, in Play mode only. That is the multiplier **LodScale** writes too (see Quality levels and options menus): turn LOD As 60 off on the **Demo** object's viewpoints before trying **LodScale** in the demo.
- Each viewpoint has its own exposure. In free flight the exposure follows the light at the camera instead, lifting the shade under the trees by up to 1.5 EV (the **Demo** object's Adaptive exposure). It is off during Measure, benchmark runs and captures, which keep the viewpoint's own exposure.

Measured at the Lookout (press **1** before **M**) on 2026-10-21, with grass in Match Unity, the valley terrain's Detail Density at 1 and a URP asset with the template's settings (RTX 5080, Ryzen 9 7950X3D, Unity 6.3, Direct3D 12):

- a development player at 2560x1440: 0.07x Unity's frame time (255 -> 17.5 ms; 4 -> 57 fps), the render main thread 0.03x (95.6 -> 2.7 ms), the GPU 0.96x (17.8 -> 17.0 ms), with 522,116 trees on the two terrains (the demo has 522,156 since: 40 firs were added to the valley terrain), every grass and detail casting shadows and taking the terrain's baked light (Terrain Light). In the demo's densest grass views our GPU time is 0.95-0.97x Unity's. Measure on your own machine: the numbers depend on the CPU most of all.

Lighting. Shadowmask lighting baked with the CPU lightmapper: the terrains carry a lightmap, the trees take the scene's Light Probe Groups (the terrain's Bake Light Probes For Trees is off on both sides; Unity's default is on). The canopy's shadows in the bake come from simple stand-in trees used only for baking, which are not shipped: Generate Lighting in your project would bake without them (the forest floor loses its baked canopy shade). Keep the shipped bake. The ferns, shrubs and grass take that baked shade from the terrain's lightmap at their roots (Terrain Light, on their Drop-In Grass materials).

Wind. The **Demo** object's Foliage Wind tunes the valley's wind by group: Trees (the tall and mid-size firs and pines), Bushes (the young trees, ferns and shrubs) and Grass (see Wind groups, under Drop-In Leaves, Bark and Grass).

Shadows and SSAO come from your project's URP asset (in a new URP project: realtime shadows to 50 m). The camera uses TAA (a camera setting, the same on both sides). The demo ships a second asset, **Demo/Settings/DropInTerrainRenderer_Demo_URP.asset** (shadows to 350 m in 4 cascades, a 4096 px soft shadow map, SSAO tuned), assigned nowhere: to see the demo as in the store's images and video, set it in Project Settings > Quality (and back after). Drop-In Terrain Renderer never changes a project setting.

The store's images and video were made with that asset. The video's side-by-side comparisons ran with the same settings on both sides, written on screen: the terrains' Tree LOD Bias Multiplier at 0.5, 2 shadow cascades and a 2048 px shadow map (the demo's saved values: 1, 4 and 4096 px), timed in a 1600x900 Direct3D 12 development player on our test machine. The one titled Extended grass is not like for like: our grass to 500 m against Unity's 250 m. The video shows the sky of Drop-In Sky & Weather, sold separately and not part of this package; its frame rates were measured with the demo's own sky. The Lookout numbers above use the template's URP asset (shadows to 50 m) instead.

Size. The demo is about 460 MB of the package; untick **Demo/** when importing to leave it out. Its content is CC0 (Poly Haven, ambientCG) and public domain (USGS): see **Third-Party Notices.txt**.

The Inspector

The Drop-In Terrain Renderer Inspector shows, from the top:

- the product name and version, with a Documentation button (this file; also `Tools > Havoila Studio > Drop-In Terrain Renderer > Documentation`);
- the status boxes: what Drop-In Terrain Renderer draws on this terrain, or why Unity draws it (see "Why does Unity draw my trees?"). They stay outside the sections, so the reason never hides. A switch you turned off reads as information (`Draw Trees is off (the Trees switch): Unity draws this terrain's trees.`), and so does a passing state, with its reason: not built yet, `placing (N patches to go)`, a material not patched yet, a grass shader being adapted, a measurement's pause. A terrain Unity draws for a reason that stays until you act is a warning, with the reason;
- the setup's progress, or its Last setup box (see Quick start);
- the toolbar: Set Up; Patch Materials (makes the drawing copies of the terrains' materials now, after removing the copies of shaders no longer in the project); Read Again (reads the trees and details again, after a script changed them without Unity's change notice); Stats (the Scene view's stats overlay); Expand all and Collapse all;
- the sections. They start collapsed and stay as you leave them for the editor session. Each header says on its right what that part does now.

Four sections have a switch in their header. Off greys out what the section holds and skips that part's cost:

Section	Holds	Switch	Off
Trees	LOD Bias, Cross Fade Band, Max Distance, Cast Shadows, In Reflections, then the Tree kinds fold	Draw Trees	Unity draws this terrain's trees, as without the component: Drop-In Terrain Renderer uploads, culls and draws none of them. The header reads "Unity draws".
Impostors	Impostor Min Height, then each tree kind's impostor row	Impostors	This terrain's tree kinds ask for no impostor and end at their last LOD, as Unity draws them: no impostor draws, their textures and draw lists leave Drop-In Terrain Renderer's GPU memory, and builds leave their impostors out. A kind another terrain plants with Impostors on keeps its impostor (a kind is one for every terrain); the section then names those terrains, and the header reads "off here · N of M kinds kept". Greyed while Draw Trees is off.
Grass and Details	Detail Shadows, Detail Mode with the Extend Reach button, the Extended fold, Streaming Budget, Detail Memory, Watch Details At Runtime, then the Detail layers fold	Draw Details	Unity draws this terrain's details. The header reads "Unity draws".
Grass Interaction	Area, Resolution, Recovery, Max Angle, Preview in Edit Mode	Grass Interaction	No bend field, and its textures are released. One field serves every terrain, so grass still bends while another terrain has it on (the header reads "Active (another terrain)").

Look (Keep mine / Convert), Benchmark and Advanced have no switch: Look is a choice with no cost of its own, Benchmark is a tool (its header shows the last verdict, such as "2.2x faster"), and Advanced holds Motion Vectors, Punctual Shadow Radius and the component's references. With several terrains selected, the Inspector says `Select one terrain to see its tree kinds, detail layers, looks and impostors.`

Compatibility

Drop-In Terrain Renderer needs Unity 6.3 or newer and URP. By default it draws your trees and detail meshes with your own materials (Keep mine), through patched copies of their shaders, so they look as Unity draws them, apart from the differences listed below. This section says what was tested and on which Unity versions, where the image differs from Unity's, and what to keep in your project. Whatever Drop-In Terrain Renderer cannot draw stays drawn by Unity, with the reason in the Inspector (see "Why does Unity draw my trees?" below).

What was tested

Content	Unity 6.3	Unity 6.6	Difference from Unity's image
URP Lit and Simple Lit trees (prefabs without a LOD Group) and detail meshes	Tested, wind included	Tested, wind included	None beyond Unity's own noise
Shader Graph (patched automatically)	Tested	Tested	Unity's grass graph: none beyond Unity's own noise. SpeedTree 8 and NatureManufacture graphs: see their rows
SpeedTree 8, with Unity's wind	Tested, wind included	Tested, wind included	A fine speckle along leaf edges and in their ground shadows (fix planned for 1.1)
SpeedTree 9	Not tested	Not tested	Not rendered in any test (no test content). Its wind is not read, so its trees would stand still
NatureManufacture (with its URP pack)	Tested (its trees barely move in the wind test)	Tested (the same)	A few small trees dither differently partway through a LOD cross-fade
The Visual Engine (with its Universal pack and its manager)	Tested, wind included	Tested, wind included	Slightly above Unity's own noise, most from above (0.30 against Unity's 0.11, out of 255; eye level 0.89 against 0.82)
Unity-style grass: grass textures (cards and billboards) and detail meshes with GPU instancing off	Tested	Tested, wind included	A faint speckle in far grass cards
A hand-written URP shader built like URP's own (our test shader, see Hand-written shaders), as a LOD Group tree and an instanced detail mesh	Tested in the editor (Play mode), not in a player	Compiled only (its copy, on D3D11 and D3D12)	None beyond Unity's own noise (0.002 out of 255)
Drop-In Bark, Leaves and Grass (the Convert shaders; compared with Unity drawing the same Drop-In materials)	Tested, also in XR (Unity's Mock HMD)	Tested, also with DXC	Bark and Leaves: none beyond Unity's own noise. Grass: a little above it at eye level (1.22 against 0.81 on 6.6)

"Tested" means rendered in Windows players and compared with Unity's own drawing of the same scene. "None beyond Unity's own noise" means the difference is within a few hundredths of the difference between two of Unity's own renders of the same scene (see "How this was tested").

Unity versions.

- Unity 6.3 LTS: tested on 6000.3.25f1, in the editor and in Windows players. On 6000.3.8f1, an early patch, the package imported and compiled, the EditMode tests of the time passed (131 of 132; one skipped: it needs Drop-In Sky), and every pass of the four Drop-In shaders (Bark, Leaves, Grass and Impostor) compiled for D3D11, in its default variant and a common lit one; it was not run as a player, and the larger suite of version 1 has not been run on it. On patches before 6.3.17 (and 6.4 before 6.4.10), the Drop-In shaders adapt to the older URP so that Deferred still compiles (checked on 6.3.8). Patches before 6000.3.8f1 have not been tried.

- Unity 6.6: the package, its demo included, imported cold into fresh URP projects (the latest runs on 2026-10-30). On 6000.6.7f1, the version Asset Store reviewers use since 29 October 2026 (a new project from its 3D URP template): 0 compile errors and no warning from the package; 471 of the 479 EditMode tests passed in the D3D12 editor, none failed (8 skipped: those that cannot run in a new project or in batch mode, and one that needs Drop-In Sky); the demo opened and played twice with Drop-In Terrain Renderer on, in the D3D12 and in the D3D11 editor, with no error, warning or exception from the package (the D3D11 editor logs Unity's ring-buffer notice, see Known issues). On 6000.6.3f1, the reviewers' version before it: the same import and tests, with 470 of the 479 passed in the D3D11 editor (9 skipped: one more, a DXC test that needs Direct3D 12), and the demo played twice in the D3D11 and in the D3D12 editor (the package as exported on 27 October). Windows D3D12 players were tested on 6000.6.0f1.
- Unity 6.4 and 6.5: not tested. They were short tech streams and are no longer in Unity Hub's release list. Their known URP changes are handled by version checks written from Unity's release notes, never compiled on those versions.
- Later versions (6.7 and on): not tested.

The package's tests. Its 487 EditMode tests (`Tests/Editor`, in Unity's Test Runner) skip what cannot run in your editor rather than leave it inconclusive: a new project without a shader library yet, batch mode, no graphics device (`-nographics`, as on many CI machines: 56 of the 487 skipped in all on 6.3, 57 of the 479 on 6.6). On 6000.3.25f1 with a graphics device, 479 of the 487 passed and 8 were skipped. Unity 6.6 runs 479 of them: nine tests of the occlusion guard, which only Unity 6.3 to 6.5 need (see Tree casters per cascade, under Shadows), are left out there, and one checks that 6.6 needs none. Unity's command-line runner exits with code 2 when any test is inconclusive, as when one fails; a CI run of these tests exits with 0 when none fails (checked on 6000.3.25f1, 6000.6.3f1 and 6000.6.7f1, in batch mode with and without a graphics device, 2026-10-30).

Render paths, platforms and URP features

- **Render paths.** Everything in the table above was compared in Forward+. Forward and Deferred were checked in the editor with the Drop-In shaders, URP Lit detail meshes and Unity-style grass. SpeedTree, NatureManufacture and The Visual Engine content has not been compared in Forward or Deferred. Deferred+ is not tested.
- **The per-object light limit.** Outside Forward+, when more additional lights reach a camera's view than the URP asset's Per Object Limit, that camera gets Unity's own trees and details: the same image, at Unity's cost. The Inspector shows it as "Some cameras: ...". In Deferred this happens only when a drawn material is lit in the forward pass (no G-buffer pass, or transparent). Forward+ has no such limit.
- **Graphics APIs.** Everything in the table above was compared with Unity's image in D3D12 players. An earlier build also ran in D3D11 and Vulkan players with instanced detail meshes and Unity-style grass textures: the detail meshes matched within Unity's own noise on all three; on Vulkan, grass cards and billboards differed a little more than on D3D12 (cards 2.48 against 1.99/255). Trees have also been drawn on D3D11 and Vulkan, their image not compared with Unity's there: URP's SpeedTree 8 trees in D3D11 and Vulkan development players (timed, see SpeedTree and Motion vectors), URP Lit trees in D3D11 players, and, in the Unity 6.3 editor on both APIs, the demo and test scenes with URP Lit and Simple Lit, SpeedTree 8, The Visual Engine and impostors. NatureManufacture content has not been drawn on D3D11 or Vulkan, nor the Drop-In shaders in D3D11 or Vulkan players; the Drop-In shaders were compiled for D3D11 (every pass) and Vulkan (the forward pass). Metal (macOS, iOS), Linux, consoles and WebGPU have never been built.
- **Platforms.** Version 1 targets desktop, consoles and PC VR, in URP only; only Windows has been run. Quest and mobile, the Built-in pipeline and mesh terrains are not supported.
- **Hardware.** Drop-In Terrain Renderer needs compute shaders, GPU instancing, indirect draw arguments and vertex shaders that read five buffers (shader model 4.5). Where they are missing, Unity draws every terrain's trees and details, and the Inspector says why.
- **Built-in modules.** XR, Particle System and Wind (Package Manager > Built-in) can be turned off. Without XR there is no XR rendering; without Wind there is no WindZone, so no `FoliageWind.Source`, and the Drop-In shaders blow with each material's own direction and breeze (see Wind, under Drop-In Leaves, Bark and Grass); without Particle System there are no particle lights to count against the per-object light limit.
- **XR.** Tested only with Unity's Mock HMD on Windows, single-pass instanced and multi-pass. The Drop-In shaders (6.3 players) match Unity in each eye within its own noise, and Unity-style grass differs as much as without XR. Your own LOD Group trees (Keep mine) differ a little more: a band of mid-distance trees picks its LOD or cross-fade per eye, where Unity picks one for both eyes (up to 0.75/255 over the image without XR). Impostors face the point between the eyes in single-pass instanced and each eye in multi-pass. Real headsets and the multiview path have not been tested.

In XR, URP draws the details' shadow casters into every cascade, as before the details' own cascade pass (see Cascades, under Shadows), and the trees' shadows use the earlier test (no reach pass, see Shadows): neither pass has been checked in XR yet. Motion vectors in XR (TAA or motion blur in a headset) have not been checked either: their image and their cost were measured without XR only (see Motion vectors). The benchmark only labels an XR run in version 1, and its panel and the stats overlays draw on the desktop window (see Not measured, under "Did it help? Measuring your scene").

- **DXC** (Microsoft's newer shader compiler; Unity's default is FXC). A shader or a Shader Graph (its Graph Settings, on Unity 6.6) can ask for it (`#pragma use_dxc`), and on Unity 6.6 a project can choose it for Direct3D 12 (Graphics Settings > Shader Build Settings > Shader Compiler Backend Selection, or a build profile's). The copies of your shaders always compile with FXC: DXC does not let a shader write Unity's per-draw values, which the copies do, so each copy ends every program block with `#pragma never_use_dxc`. Your own shader still compiles as it asks, for everything Unity draws with it. A shader whose code needs DXC (wave intrinsics, for example) does not compile as an FXC copy: its terrain stays Unity's, with the reason. Tested on Unity 6000.6.3f1 with that choice set to DXC for Direct3D 12: the copies of URP Lit, URP's SpeedTree 8 shader and graph, Unity-style grass and a test shader asking for DXC compiled; the Drop-In shaders and the compute shaders compiled with DXC; a Direct3D 12 development player drew those trees (with Drop-In Bark) and the grass, with no error. Its image was not compared with Unity's. Without that choice, the Drop-In shaders were also tested with `#pragma use_dxc` added: on Unity 6.6 (D3D12), the same image as under FXC (a forward path); on Unity 6.3 (D3D12), Drop-In Bark's image in Deferred and Forward+ the same as under FXC but for a few edge pixels (an editor test). Deferred with the project's DXC choice has not been run in a player.
- **URP features.** MSAA 4x, post-processing, TAA, STP, render scale 0.5 and 1.5, depth priming (Forced) and SSAO were checked in the editor with patched Lit materials and Unity-style grass: the same image within the differences above (STP leaves scattered single pixels on blade edges).
- **Motion vectors** (TAA, STP, motion blur). The component's Motion Vectors setting. As Unity (the default): as Unity draws them. Details draw the camera's motion only, Drop-In shaders included, as Unity's instanced details do (waving grass can smear under TAA; grass bent by interaction draws its bend, see Grass interaction). Trees follow their terrain's Tree Motion Vectors setting, as Unity's do, when their material's motion-vector pass is on: Drop-In shaders with wind, and your materials whose pass is on and has an instancing variant (URP's SpeedTree 8 materials have it on, and a SpeedTree 8 import's renderers are on Per Object Motion). Where the terrain asks for per-object motion (Per Object Motion, or Inherit From Prototype, the default, with the prefab renderers' Per Object Motion) they draw their own motion (the wind's); where it asks for none (Force No Motion, or the renderers' Force No Motion under Inherit From Prototype) a tree with a LOD Group draws no motion, as Unity's terrain does (Drop-In shaders to 150 m, the camera's motion beyond, as for their wind's motion; your materials at every distance when their pass's fragment sets its instance up, see Hand-written shaders); else the camera's motion. Trees whose material's pass is off draw the camera's motion (Unity's terrain draws none for them under Force No Motion), and so do trees without a LOD Group under Force No Motion (see below). A tree prototype is drawn once for every terrain that plants it and follows only those terrains: when their settings differ, per-object motion wins, then the camera's. As Unity draws a tree's motion-vector pass where Unity's terrain draws it: Drop-In shaders to 150 m, your materials' own passes at every distance as Unity's terrain does (a SpeedTree pass computes the frame before's wind too). Their cost, measured once on our test machine (Unity 6.3 development player, Direct3D 12, 2560x1440, TAA): a forest of URP's SpeedTree 8 trees with their motion-vector pass on (two kinds, 16 tree draws) took 0.11 ms more GPU time (+1.6 %), 0.02 ms more on the render thread and 20 more draw calls under Per Object Motion than under Camera Motion Only. With Wind: the wind's motion everywhere (Drop-In shaders on details and trees, whatever the terrain's setting; your materials whose own motion-vector pass is on and has an instancing variant, such as a Shader Graph's time-based motion; Unity's grass wave), at some GPU cost. A material whose motion-vector pass has no instancing variant draws the camera's motion (see Hand-written shaders). Impostors draw the camera's. Measured against Unity's terrain (Unity 6.3, wind running, a 320x180 camera, each of the four settings, 256 still frames each): every tree drew the motion described here. Where both drew a tree's own motion it had Unity's size: a Drop-In LOD Group tree's wind 1.00 times Unity's, URP's SpeedTree 8 pine's 0.99 to 1.04 times (As Unity and With Wind, see SpeedTree), on the same pixels; under Force No Motion both drew none. Trees without a LOD Group whose material's motion-vector pass is on (Drop-In Bark and TVE's demo tree in our tests) got motion thousands of pixels long from Unity's terrain under every setting, on a terrain Drop-In Terrain Renderer does not draw as well; Drop-In Terrain Renderer draws them as described here. TAA ghosting with wind running has not been measured.
- **Lighting.** Trees are lit by light probes, as Unity lights them: each tree by the probes at its position when the scene has baked light probes (128 bytes of probe data per tree on the GPU, counted in the stats overlays' GPU memory), otherwise by the ambient probe. Detail meshes take the ambient probe, as in Unity, unless they are drawn with Drop-In

Grass or Leaves with Terrain Light on (below). Keep your trees inside the light probes' volume (a sparse grid is enough over far terrains): outside it Unity's own terrain looks each tree's probes up far more slowly (measured: 2.4 s a frame for 141,000 such trees, 0.1 s inside), and the interpolated values are extrapolated, at times far out of range; Drop-In Terrain Renderer gives such a tree its nearest probe's light instead. Unity-style grass on a lightmapped terrain uses its lightmap: plain, directional and Subtractive lighting matched Unity in Forward+ and Forward. A Shadowmask bake drew as the plain one in that test; the mask itself, which acts past the realtime shadow distance, has not been compared. Lightmapped billboard grass in Deferred, and a terrain lit by realtime GI (Enlighten) alone, are left to Unity, with the reason. A lightmapped draw with Drop-In Grass or Leaves takes its light through the blades or leaves from its own lightmap seen from behind, not from the ambient probe: converted grass (Drop-In Grass cards and billboards) on a lightmapped terrain, and a static lightmapped mesh such as a hand-placed bush. In a player, Unity-style and converted grass take a script's swap of `LightmapSettings.lightmaps` only through `DropInTerrainRenderer.RefreshLightmaps()`, as Terrain Light does (below).

- **Shadowmask Mode and the trees' probes.** In a Shadowmask bake (Mixed Lighting's Lighting Mode Shadowmask) with the quality level's Shadowmask Mode set to Shadowmask (as the URP template's Mobile level ships), URP takes the minimum of the realtime and the baked shadow within the shadow distance, so sunlit tree crowns there read their probe's baked occlusion, as URP Lit objects do: probes in a baked canopy's shade darken the crowns above them. Distance Shadowmask (the template's PC level) does not: it uses the realtime shadow alone within the shadow distance, fading to the baked one at its end.
- **Terrain Light on detail meshes.** A detail mesh drawn with Drop-In Grass (Terrain Light is on by default) or Drop-In Leaves (off by default: turn it on for ferns and bushes) takes the baked light of the lightmapped terrain under its root instead of the ambient probe: the terrain's lightmap (half-Lambert with the direction map in Directional mode), its shadowmask for the main light and the other lights past the realtime shadow distance, and the lightmap seen from behind for the light through the blades. Under a canopy whose shade is baked into the terrain, the plants are as dark as the ground they stand on, where the ambient probe made them glow. It applies in Drop-In Terrain Renderer's draws and in Unity's own instanced detail draws of the same material, so a measurement's pause shows the same light; in the test, ours matched Unity's within 0.013/255. Subtractive lighting is handled as on a lightmapped surface (the sun's direct light is in the lightmap; realtime shadows darken it). Trees keep their light probes. Limits:
 - up to 16 lightmapped terrains with a Drop-In Terrain Renderer component, sharing up to 4 lightmaps: past them, a terrain's details take the ambient probe, and its Inspector and the stats overlay's terrain list say so (bake fewer, larger lightmaps);
 - the terrain under a plant is found by position. A terrain with a component but no lightmap (not baked yet, lit by probes or realtime GI, past the 4 lightmaps) that lies over a larger lightmapped one takes one of the 16 places, so that its plants keep the ambient probe rather than take the lightmap of the ground under it; with all 16 in use they can take it;
 - terrains without a component, a terrain lit by realtime GI (Enlighten) alone, and scenes with Adaptive Probe Volumes (URP samples the volume per pixel) keep Unity's lighting of the details. One exception: the plants of a terrain without a component that stand over a lightmapped terrain with one take that terrain's lightmap in Unity's draws, when their material has Terrain Light on (add a component to that terrain, or turn Terrain Light off on them);
 - each plant reads the lightmap once, at its root, bilinearly (no bicubic sampling): a plant across a sharp shadow edge takes the side its root stands on, and the reflections under a canopy are still the reflection probe's;
 - a plant more than 2 m above the terrain's highest point or below its lowest (on a structure high over it) takes the ambient probe; where lightmapped terrains overlap (a detailed terrain over a large backdrop one), the smallest one holding the plant lights it;
 - a Drop-In Grass or Leaves material that a tree uses too takes terrain light in Unity's own tree draw: keep Terrain Light off on materials shared with trees. A GameObject drawn with such a material through GPU instancing over a lightmapped terrain takes it too;
 - bakes, scene loads and a change of the active scene are seen at once. A script that swaps `LightmapSettings.lightmaps` (lighting scenarios) calls `DropInTerrainRenderer.RefreshLightmaps()` after the swap, and the details change in the same frame as the ground. Unity sends no event for a swap, and reading the lightmaps makes garbage, so a player does not look for one: without the call, the details keep the lightmaps they had. The editor also finds a swap within a second.
- **Adaptive Probe Volumes.** Every variant compiles, and URP samples the volume per pixel. A render comparison with Unity under APV has not been done (planned for a later version).

- **Shadows.** Shadow casting follows each material and renderer, as in Unity: a material with Cast Shadows off, or a renderer set to Off or Shadows Only, is respected. The main light is picked per camera as URP picks it. A detail is drawn into the main light's shadows only when its shadow can fall on what the camera sees within the Shadow Distance, followed down to the ground (the terrain's lowest heights in cells of about 8 m, so that on ground falling away from the sun it is followed as far as it falls). A tree is drawn into them only when its shadow, followed down the light to the ground in the same way, can reach a place where URP reads the main light's shadow map at all: URP's cascade culling spheres for that camera and frame, grown to cover the widest filters that read the map (URP's soft shadows, Drop-In Water's soft sun shadows, Drop-In Sky's Soft Shadows and fog), the shadow bias, and the tile borders URP's soft filter reads across. The image is the same as Unity's.
 - **The ground.** Only a terrain that casts into the main light's shadow map ends a shadow:
 - it has an enabled Drop-In Terrain Renderer component (Draw Trees and Draw Details on or off);
 - it is drawn: Draw Heightmap on, its layer seen by the camera and the light, at its full heightmap detail (`heightmapMaximumLOD 0`, the default);
 - it casts: Cast Shadows not Off, a material with its shadow pass on;
 - it meets the light's shadow layers, when the URP asset uses rendering layers.

Holes, height or hole edits not yet synced (a sculpt stroke in progress) and a `SetHeightsDelayLOD` not yet applied count as no ground, and the terrain's LOD (its Pixel Error at that distance) and the shadow bias are allowed for. A shadow is ended only where all the ground it crossed on its way down is such ground: one that crosses a hole, a seam between terrains, a backdrop terrain with Cast Shadows Off or a terrain without the component is not ended there (more casters, the same image).

- **How many fewer tree casters** depends on the sun's height and the lens: most with suns of about 15-30° and wide views, little at very low suns (long shadows really do reach the view) or with a narrow lens. In the demo under a 17.4° sun (a D3D12 development player at 2560x1440 on our test machine), URP's main-light shadow pass took 0.35-0.4 ms less at two views.
- **The trees' reach pass.** The trees' test runs in a second culling pass that Drop-In Terrain Renderer adds to each camera from script, just before URP's shadow passes, with URP's own cascades for that frame. It runs on the GPU and adds no draw of its own (the trees' shadow draws are issued for a slightly larger region, a few of them empty). Nothing is added to your URP asset or renderer. Where that pass is not used, a tree is drawn into the main light's shadows when its shadow, followed without end, can fall on the camera's view within the Shadow Distance, as before. That is the case with:
 - Render Graph off (Compatibility Mode);
 - a renderer other than URP's Universal Renderer;
 - one shadow cascade;
 - a spot or point light with shadows lighting part of the view;
 - a camera rendered inside another camera's rendering (the camera it is rendered in keeps the pass);
 - XR.

In a frame where URP's cascades cannot be read in time for a camera, or do not fit what was prepared at its start, that camera keeps the earlier test for that frame.

- **Tree casters per cascade.** URP draws each of the trees' shadow draws into every cascade. So, in the same pass, a tree whose shadow only the later cascades are read for (a distant tree, with long shadow distances) can be moved to a draw that URP's own culling sends into those later cascades only, as it does for Unity's own trees. Each tree kind and LOD gets up to two such lists. Each list is drawn with one more shadow draw per material slot of that LOD's renderers, whose bounds are a small box placed where only those cascades take it. A tree moves there only when every cascade its shadow can be read from (its shadow reaching that cascade's culling sphere, grown as above, or its box on that tile's border) is checked to draw that box in that frame, with URP's own cascades. Every other tree is drawn into every cascade as before, so the image is the same; a check that fails costs time, never a shadow. Such a list is drawn only for a tree kind and LOD with enough trees in that camera's later cascades for the GPU time they save to pay about three times over for its draw calls' CPU time (counted by the culling itself, read back a few times a second without waiting for it; none for a kind planted fewer than four times), and in all they add at most half as many shadow draw calls again as the trees' own. A box farther from the camera than its far clip plane, or than its layer cull distance for the terrain's layer (`Camera.layerCullDistances`), would be culled, so its

draws are not made and those trees stay in every cascade (with a far clip plane at about the Shadow Distance, the boxes for all but the last cascades can be past it). A box also needs at least 3 m between two cascades' far ends. It is off, and the trees' casters are drawn as in the earlier text:

- wherever the trees' reach pass is not used (the list above). In XR (Unity's Mock HMD, single-pass instanced and multi-pass), no such draw was made;
- for an orthographic camera;
- in a camera's first frame, and in the first frame after its projection (field of view, aspect, clip planes), its URP asset or that asset's shadow settings change (the boxes are placed from its previous frame's cascades; a projection jittered from frame to frame for temporal anti-aliasing or an upscaler counts as unchanged);
- for every camera, while more than eight cameras with shadows render each frame (split screen, security cameras);
- when the trees of every taken-over terrain, each counted once per LOD of its kind, pass 7,456,539 (about 1.9 million trees with 4 LODs): the trees' lists then leave no room for these draws (the trees' own limit is 8,388,608: see `too many trees × LODs` under "Why does Unity draw my trees?");
- for a camera whose draws would not pay for themselves: all they save on the GPU must pay two and a half times over for all they cost on the CPU, choosing them included (once on, a camera keeps them until they would pay only twice over, so a view at the threshold does not switch them on and off). At short shadow distances (URP's default of 50 m among them) they switch themselves off as a rule: there, a tree's own shadow often reaches back into the nearer cascades, so the draws take far fewer trees out of them, and their extra draw calls would cost more CPU time than they save on the GPU. That camera checks again a second later, or as soon as its lens or the URP asset's shadow settings change. These thresholds come from measurements on our test machine (Ryzen 9 7950X3D, RTX 5080) at 50 m and 350 m shadow distances. With a faster CPU or a slower GPU they lean towards off; with a slower CPU or a faster GPU the draws can stay on where they pay less than here (the threefold margin covers a CPU about twice as slow). From its second frame, a new camera can draw them for a few frames, before the culling's first count comes back;
- while the main light has per-layer shadow distances (`Light.layerShadowCullDistances` above 0 on any layer): they cull those boxes. They are read once a second, never per frame, since reading them makes garbage. So a script that sets them calls `DropInTerrainRenderer.RefreshLayerShadowDistances()` right after setting them: without that call, the shadows of distant trees can be missing for up to one second: those of any tree drawn into the later cascades only, nearer than those distances too;
- on Unity 6.3 to 6.5, for a camera with Occlusion Culling on, while a loaded scene has baked occlusion data, or while that cannot be known. With baked occlusion data loaded, Unity 6.3 can leave shadow casters farther than about 125 m from such a camera out of the shadow map, in view or not, Unity's own too, and so these boxes (in our tests with a far clip plane of 1,000 m; nearer with a nearer one). Unity 6.6 does not. There is nothing to set up. In the editor, the open scenes are read as they are opened, baked, cleared and loaded. In a player or an AssetBundle, each scene built without baked occlusion data carries one hidden, empty object, "Drop-In Terrain Renderer Occlusion Mark", that says so (in the built copy only: the scene asset is not changed), whether the scene has terrains or not (see Removing Drop-In Terrain Renderer). A scene counts as having data when it cannot be read: a scene from Addressables content in a player, or in editor Play mode with Use Existing Build (our build step does not mark those scenes; with Use Asset Database the editor reads the scene itself, so the saving seen there is not in the player), a scene in an AssetBundle built with an earlier version (build it again), a scene saved as binary, a scene still loading or created by a script, more than 32 scenes loaded at once, and in the editor a new scene never saved that is not the active one. A camera's Occlusion Culling does nothing without baked occlusion data: with such scenes, turn it off on the camera to have tree casters per cascade there.

Measured in the demo with its 350 m asset (`DropInTerrainRenderer_Demo_URP`: four cascades, a 4096 map), in a D3D12 development player at 2560x1440 on our test machine (RTX 5080, Ryzen 9 7950X3D), each viewpoint run with them on and off: - They were on at 8 of the 10 viewpoints and took 1.3-4.2 ms of GPU time off each (at the Lookout, 1: 24.5 to 20.4 ms). That cost 0.2-0.4 ms more render main thread and 48-142 more tree shadow draw calls (470 to 594 at the Lookout). The camera rule kept them off at the other two, the Meadow (2) and the Forest edge (7), after their first three frames. - A tree caster still goes into more cascades than Unity's own trees. Weighted by its triangles, it went into 2.3 cascades on average at the Lookout, against 4 with them off; our estimate of Unity's rule gives 1.4 (the Aerial, 5: 2.1 against 1.0). A tree keeps every cascade its shadow reaches: a margin in the design for shaders that read coarser cascades, such as Drop-In Sky's Soft Shadows. Unity's culling drops those cascades after the first culling sphere that holds the whole shadow. Each kind and LOD also has two lists at most, and the trees they do not take stay in every

cascade. - With 50 m shadows (URP's default), the camera rule kept them off at every view measured, on the same asset set to 50 m and on the PC asset of Unity's URP project template. No such draw was made. Within one player process, their checks cost up to about 0.02 ms of main thread a frame where trees cast within 50 m, and GPU time stayed within ± 0.07 ms. - The image was pixel-identical with them on and off in every check: in the same player (350 m and 50 m, two cascades, a 12° sun), and in Unity 6.3's editor. On Direct3D 12: Forward, Forward+ and Deferred, one to four cascades, 50-350 m, soft shadows off and High, normal bias 1 and 3, cascade border 0 and 0.15, and rendering layers. On Direct3D 11 and Vulkan: a subset (Deferred and two cascades among them). On all three: SpeedTree 8 trees, and Drop-In Sky & Weather 1.0.0's Soft Shadows at Max Blur 2 and 10.

- A custom shader that filters the main light's shadow map around a point wider than both about 5 m and 24 shadow-map texels may miss the edge of a distant tree's shadow. Drop-In Sky and Drop-In Water are covered.
- **Cascades.** URP draws an indirect draw into every shadow cascade whole, where Unity culls its own casters per cascade. So Drop-In Terrain Renderer draws the details' casters into the main light's cascades itself: a render pass it adds to each camera from script while that camera renders, right after URP's shadow passes. Nothing is added to your URP asset or renderer, and there is nothing to set up. With URP's own matrices and bias for that frame, it draws each detail into URP's shadow map only for the cascades its shadow can be read from: those whose culling planes it meets (Unity's own rule), up to the first cascade whose culling sphere holds all of its shadow (followed down to the ground, as above). As in URP's own shadow pass, a detail casts only when its rendering layers meet the main light's shadow layers, when the URP asset uses rendering layers. The image is the same, and a detail is drawn into about two cascades on average instead of every one: at two of the demo's views with four cascades, 2.0 and 2.5 cascades a caster with 50 m shadows (39-51% fewer detail shadow triangles), 1.2 and 1.5 with 350 m (63-69% fewer). The trees' casters stay URP's: drawing their hundreds of draws per cascade from script would cost more CPU than it saves on the GPU (see Tree casters per cascade, above, for how many of them are kept out of cascades they do not need, where that pays).
- URP draws the details' casters into every cascade, as before, when Render Graph is off (Compatibility Mode), with a renderer other than URP's Universal Renderer, while a spot or point light with shadows lights part of the camera's view (its range meets the view), for a camera rendered inside another camera's rendering (the camera it is rendered in keeps the pass), and in XR.
- **Spot and point lights.** While a spot or point light with shadows lights part of the camera's view, trees and details within the component's Punctual Shadow Radius (40 m by default) cast into spot and point light shadows (a light made by a script counts within a second). When the main light also casts shadows, its casters are drawn into each shadowed spot and point light as well (Unity culls casters per light), so several shadowed spot or point lights in a forest under a shadowed sun cost more than with Unity: turn off the shadows of those that do not need them (see Known issues).
- **Rendering layers** (light layers, decals) behave as Unity's: trees and instanced detail meshes use their prefab renderer's rendering layer mask, Unity-style grass the terrain's. A terrain with Preserve Tree Prototype Layers on has its trees left to Unity (its details stay Drop-In Terrain Renderer's).
- **GPU Resident Drawer.** Terrain trees and grass go through Drop-In Terrain Renderer's own draws, so the Resident Drawer is not involved. Drop-In shaders have no DOTs instancing variant in version 1: a GameObject using a Drop-In material is drawn the normal way under it.
- **Quality and LOD settings.** Trees switch LODs as their LOD Groups would: Quality's LOD Bias and Maximum LOD Level and the terrain's Tree LOD Bias Multiplier apply, and the component's LOD Bias multiplies them (1 by default: as Unity). Quality levels, their Terrain Setting Overrides and an options menu's sliders: see Quality levels and options menus.
- **Your terrain settings.** While a frame renders, Drop-In Terrain Renderer sets the terrain's Tree Distance, Detail Distance or Draw Trees And Foliage to keep Unity from drawing what it draws, and puts them back at the end of the frame. Nothing is saved with the scene. A script that reads these settings during rendering sees Drop-In Terrain Renderer's values, and a script that changes them for one camera is not followed. In Play mode and players, the quality scales (see Quality levels and options menus) also write the terrain's Detail Distance, Tree Distance, Billboard Start, Detail Density and Tree LOD Bias Multiplier, and put them back. Tree colliders are not affected: the Terrain Collider is separate.
- **Moving terrains and floating origin.** A terrain moved at runtime, or a shifted world origin, keeps its trees and details: only the terrain's small record is uploaded again (checked with terrains moved 2.5 km and back). With baked light probes, its trees are lit again half a second after it stops moving, which in version 1 reads every terrain's trees again (see Trees changed at runtime). Motion vectors are wrong for the frame in which it moves.
- **Fast Enter Play Mode** works, with domain reload off and with scene reload off.

- **Game time.** Unity-style grass waves on unscaled time, as Unity's own grass does: it keeps waving with `Time.timeScale` at 0. The Drop-In shaders' wind and grass interaction follow game time, so a paused game stills them.
- **Edit mode and the Scene view.** Trees and grass are drawn in Edit mode and in the Scene view, but they are not GameObjects: they cannot be picked, and Scene view debug draw modes that replace shaders draw them wrong. While a shader variant compiles, Unity's cyan placeholder shows. Terrain tool strokes hide trees and grass, as in Unity.
- **Cameras.** Each camera culls and draws for itself; a camera that does not see a terrain's layer gets nothing from it. Cameras rendered two deep or more inside other cameras' rendering get Unity's own trees and details. Preview cameras (asset and material previews) render preview scenes, which Drop-In Terrain Renderer does not take over. A camera that appears or jumps far from the others shows Unity's grass for itself until its own is placed (see Grass streaming and garbage, below: `PrewarmDetails` places it ahead of a scripted jump). Camera stacking (Overlay cameras) and split screen have not been tested, and grass interaction bends round one view only. For reflection probes and planar reflections, see Reflections below.
- **UniversalRenderPipeline.RenderSingleCamera** (obsolete) renders without telling other assets, so its camera shows the terrain without trees or grass. The Inspector names the scripts that call it. Fix: use `RenderPipeline.SubmitRenderRequest`, or wrap only the render call in `using (DropInTerrainRenderer.UnityDraws()) { ... }`, which gives that render Unity's own trees and details.

Your own shaders

Keep mine or Convert. Each prototype (a tree prefab, a detail prefab or a grass texture) is drawn either with your own materials (Keep mine, the default) or converted (Convert, in the component's Look section). The choice holds for the whole project, so a prefab looks the same on every terrain. Keep mine draws through a patched copy of your shader: your shader, material and prefab are never changed. Convert draws with Drop-In Terrain Renderer's Leaves, Bark or Grass shaders, using materials made from yours. It understands URP Lit and Simple Lit, SpeedTree 8 and 9, and Drop-In Sky's Park Foliage; it keeps the rest as yours, with the reason (transparent Lit surfaces, Complex Lit, NatureManufacture and The Visual Engine shaders, custom Shader Graphs, SpeedTree billboards). Impostors, grass bending of detail meshes, and the Drop-In Sky and Drop-In Water hooks need Convert.

Edit your original materials. The draws read your material's current values and keywords (except a Drop-In material's lightmap keywords: our draws set those from the terrain or leave them off, see Builds): Inspector edits, and changes a script makes in the editor or in a player, show within a second, with no new patch. A script that animates a tree or detail material (a dissolve, a season) calls `DropInTerrainRenderer.RefreshMaterials()` after each change, and every drawn copy takes its source's values at the next frame. Shader globals (`Shader.SetGlobalFloat`, for example) apply at once. A material feature you turn on or off after the copy was made (a keyword, such as Normal Map or Alpha Clipping) shows at once in the editor, but a player may lack its shader variant: the build takes its variants from the generated material, which keeps the keywords it was made with. Fix: delete that material's copy in Assets/Drop-In Terrain Renderer Generated/Materials, then press Patch Materials, which makes it again with the current keywords.

Trees without a LOD Group are drawn as Unity's URP terrain draws them: every mesh renderer of the prefab, out to the terrain's Billboard Start plus its Fade Length, or its Tree Distance if that is nearer. URP draws none of Unity's legacy billboards, so past Billboard Start Unity's terrain draws nothing in their place, and neither do we. Both distances are measured horizontally from the camera, the edge is a hard one, and past Billboard Start (over the Fade Length) the trees cast no shadow: all as Unity's terrain does it (compared image for image on our plain-tree bench). With the terrain's default Billboard Start of 50 m, that is a short distance: raise Billboard Start (Terrain settings, Tree & Detail Objects) to draw them farther, and Unity's terrain follows it too, so Drop-In Terrain Renderer ON and OFF stay like for like. Unity instances such trees but still culls them one by one on the main thread: on a bench of 20,000 plain trees (Billboard Start at their 2 km Tree Distance), Drop-In Terrain Renderer took that work from 14.3 to 0.7 ms (see Measured results). Impostors (which need Convert) then cut the GPU time of the far trees.

URP Lit and Simple Lit are patched from URP's own shader code. They were tested on trees (opaque, and alpha-tested two-sided) and on detail meshes. Transparent materials are drawn, but their sorting has not been tested.

Shader Graph. Any graph is patched automatically: there is no sub graph, node or setting to add. Tested with Unity's terrain demo graphs (grass and SpeedTree 8, with their sub graphs and custom functions) and NatureManufacture's graphs. Drop-In Terrain Renderer reads the graph's generated code through an internal Shader Graph API; a shipped test reads URP's own graph with it, so each Unity version's test run checks it. If a future Shader Graph changes that API, the reason says so

and the terrain stays Unity's until Drop-In Terrain Renderer is updated. A graph's copy is made again when the graph is saved or imported again, and when an .hlsl file outside Drop-In Terrain Renderer is imported (a custom function's file, for example).

Hand-written shaders. A hand-written URP .shader is drawn as it is when its vertex functions set up URP's instancing, as URP's own shaders do. That is the one thing it needs: in every pass Drop-In Terrain Renderer draws (untagged, UniversalForward, UniversalForwardOnly, UniversalGBuffer, ShadowCaster, DepthOnly, DepthNormals and DepthNormalsOnly), declare `UNITY_VERTEX_INPUT_INSTANCE_ID` in the vertex input struct and call `UNITY_SETUP_INSTANCE_ID(input)` at the top of the vertex function. That call places each tree. Without it, the shader's terrain stays Unity's, and the reason names the pass, the struct and the function. Shader editors that write .shader files take the same route; none were tested.

Its copy gets the rest, and your shader is never changed:

- Drop-In Terrain Renderer's instancing line and include, in every HLSLPROGRAM block (the include writes URP's per-object values: object matrices, LOD fade, light probes, previous matrices for motion vectors). URP's `Core.hlsl` can be included in the block or in an HLSLINCLUDE block. `HLSLPROGRAM` and `ENDHLSL` may have a comment after them on their line; `ENDHLSL` must start its line.
- `#pragma multi_compile_instancing`, in a pass Drop-In Terrain Renderer draws whose vertex function is seen to set its instance up but which has no such pragma where Unity reads pragmas (the block, an HLSLINCLUDE block, or a file included with `#include_with_pragmas`): a shadow caster written for one pass, for example. A MotionVectors pass gets it on the same terms.
- FXC as its compiler (see DXC, under Render paths, platforms and URP features).

These stay drawn by Unity, with the reason:

- CG shaders (CGPROGRAM blocks, Built-in pipeline code);
- shaders or graphs with their own procedural instancing (`#pragma instancing_options procedural:`, made for another renderer) where Unity reads that pragma and outside any `#if`;
- a drawn pass taken from another shader with `UsePass`: write the pass in the shader;
- code that needs DXC.

Motion vectors: a material's own MotionVectors pass is drawn where its motion is asked for (With Wind; As Unity where the tree's terrain asks for per-object motion, or for none on a tree with a LOD Group) when it has an instancing variant (its own pragma, or the one its copy gets) and compiles in the check below; otherwise Drop-In Terrain Renderer draws the camera's motion for that material. For no motion the copy's include sets URP's `unity_MotionVectorsParams.y` to 0 in the fragment stage, as URP does for a Force No Motion renderer: a pass that sets its instance up in its fragment too and ends with URP's `CalcNdcMotionVectorFromCsPositions`, as URP's and Shader Graph's passes do, writes zero (measured with URP's SpeedTree 8 graph). Its vertex stage still reads 1 and computes both positions, so a pass whose fragment does not set its instance up draws its own motion there instead, never invalid values.

As in Unity's draws:

- Put values on the material, not on the renderer. Drop-In Terrain Renderer reads only each renderer's mesh, materials, place in the prefab, shadow casting mode and rendering layer mask. Light probes are read at LOD0's first renderer's bounds centre, as Unity's terrain reads them: a probe anchor on a tree prefab is ignored on both sides. A MaterialPropertyBlock set on a renderer is not read, and per-instance properties (`UNITY_DEFINE_INSTANCED_PROP`) read as zero in these draws.
- For Deferred, give the shader a UniversalGBuffer pass, or tag its lit pass UniversalForwardOnly. URP draws a pass tagged UniversalForward alone in Forward and Forward+ only, for any renderer.
- For dithered LOD cross-fades, support the `LOD_FADE_CROSSFADE` keyword as URP Lit does; without it, LODs switch with no dither.
- Read light probes through URP's own functions: each tree's probe values are written where URP reads them. A fragment reads its tree's probes and its cross-fade only when it calls `UNITY_SETUP_INSTANCE_ID(input)` too (with `UNITY_VERTEX_INPUT_INSTANCE_ID` in the interpolators and `UNITY_TRANSFER_INSTANCE_ID` in the vertex function), as URP's own fragments do. Otherwise it reads the scene's ambient probe and draws no dither.

- Read the main light's shadows through URP's function that takes the position (`GetMainLight(shadowCoord, positionWS, shadowMask)`), as URP Lit does: it fades shadows out at the Shadow Distance. Beyond it, the shadow map holds casters where Unity's does not, so a shader that samples without the fade shows shadows there that Unity's draw does not (measured with our test shader; with the fade, the images matched).

Tested: our test shader written from URP's pass structure (Core.hlsl in an HLSLINCLUDE block; UniversalForward, ShadowCaster, DepthOnly and DepthNormals; the instance set up in the vertex and forward fragment functions), as a tree prototype with a LOD Group and an instanced detail mesh, in the Unity 6.3 editor (Play mode, Forward+): the same image as Unity's drawing of it (0.002/255 on average, no pixel over 3/255), shadows included. Written with its shadow caster lacking `#pragma multi_compile_instancing`, its copy drew exactly the same image.

The SRP Batcher is not needed: the added code acts only in Drop-In Terrain Renderer's draws, and the same copy drawn by a normal renderer behaves as your original shader.

How copies are checked. Before a copy draws, the editor checks it for errors, for an instancing variant in every drawn pass, and, for hand-written shaders outside Unity's own packages, for the instance ID setup in each drawn pass's vertex function. Then it compiles the instanced variant in the background, one compile per editor update, with the material's keywords: for a hand-written shader, the vertex and fragment stages of every pass Drop-In Terrain Renderer draws, and of MotionVectors when its copy has an instancing variant there (a failure there only gives that material the camera's motion); for a Shader Graph or one of Unity's shaders, the main pass's vertex and fragment stages and the shadow caster's vertex stage, the G-buffer, depth and depth-normals passes being checked only for their instancing variant. A failure that your shader's own pass shares, with the same keywords, is your shader's own and does not hand the terrain back; a copy whose source asks for DXC is never excused this way. A copy can therefore draw for a moment, then hand its terrain to Unity with the reason. The compile covers the editor's graphics API (on Unity 6.6, Direct3D 12's own compiler when the editor runs Direct3D 12) and the active build target only. No check runs in batch mode or without a graphics device.

SpeedTree

SpeedTree 8, Keep mine. There is nothing to set up: adding the component patches copies of your SpeedTree materials (Shader Graph or shader) and assigns the wind capture shader. Tested with Unity's URP terrain demo (SpeedTree 8 trees with LOD Groups and billboards). Trunks match Unity's; the leaves have a known difference (see Known issues).

Wind. Unity computes SpeedTree wind per MeshRenderer, and Drop-In Terrain Renderer's instanced draws have none. So for each SpeedTree tree kind, Drop-In Terrain Renderer keeps one hidden copy of the prefab far below the world (never saved), lets Unity keep its wind current, reads the wind values back every frame and gives them to that kind's draws.

- Wind runs in Play mode and in players only, as for Unity's own SpeedTree trees: in Edit mode they stand still.
- It comes from the scene's Wind Zones through Unity's SpeedTree system: keep a Wind Zone, as with Unity. Drop-In Terrain Renderer's own wind settings (FoliageWind's Source, Hold) do not change it.
- Keep the Tree component that Unity's importer puts on the prefab: a prefab without it gets no wind.
- All trees of a kind share one set of wind values, which arrive a few frames late (too little to see in the wind; how often a set arrives, in the editor and in players: below the list).
- It needs asynchronous GPU readback (`SystemInfo.supportsAsyncGPUReadback`) and the component's Wind Capture shader (under Advanced; assigned when the component is added). Without either, SpeedTree trees stand still.
- Scripts on a SpeedTree prefab also run on its hidden copy in Play mode.

Measured with wind running on Unity 6.3, Drop-In Terrain Renderer's SpeedTree trees move as Unity's do (see "How this was tested"). Motion vectors: SpeedTree 8 materials whose motion-vector pass is on (URP's SpeedTree 8 materials) draw their wind's motion where the terrain asks for per-object motion and none under Force No Motion on trees with a LOD Group, as Unity's terrain does (see Motion vectors above). Each frame's motion is the step the trees make on screen that frame: the wind values drawn the frame before are the history of the values drawn now. In our tests a set of values arrived every frame, and the motion was 0.99 to 1.04 times Unity's, on the same pixels (Unity 6.3 editor, Direct3D 12, Unity's own SpeedTree 8 pine, 256 still frames per case, As Unity and With Wind under Per Object Motion, As Unity under Inherit From Prototype; the prefab's renderers on Per Object Motion, Camera Motion Only and Force No Motion alike). A frame with no new values, if the read back falls behind, holds the trees still and draws no motion for them; the next one draws the two

frames' step. In Unity 6.3 development players on our test machine a set arrived every frame on Direct3D 12; on Direct3D 11 (about 190 frames a second) as many arrived, but about one frame in four had none and as many had two; on Vulkan (150 to 180 frames a second) one frame in five had none. SpeedTree 9 wind is not read (see below): its trees stand still.

Billboards are kept. SpeedTree billboard materials always stay yours, drawn through their patched copy, even when the prototype is converted (Drop-In wind cannot turn them). Drop-In Terrain Renderer's impostors do not replace them. The terrain's Billboard Start, Fade Length and Max Mesh Trees settings change nothing for LOD Group trees (SpeedTree's among them), Unity's or ours. Trees without a LOD Group end at Billboard Start, see Trees without a LOD Group above.

LODs and cross-fade. LOD Groups in Cross Fade mode dither between LODs within the component's Cross Fade Band (0.1 of the switch distance by default; 0: LODs pop). The LOD Group's Animate Cross-fading is not read: where Unity fades a tree only while it switches, trees standing inside the band stay dithered. LOD Groups in SpeedTree fade mode switch LODs at the same screen heights with no dither and no morph. With LOD Cross Fade off in the URP asset (its Quality settings), a build strips every cross-fade variant: then LODs and impostors switch with no dither, as Unity's own LOD Groups do, Cross Fade Band is not used, and the Trees section says so.

Converting SpeedTree is optional. Convert maps SpeedTree 8 and 9 materials to Drop-In Leaves and Bark (base, normal and extra maps, subsurface to transmission, hue variation; the last two approximately). A converted tree moves with Drop-In Terrain Renderer's own wind, shaped from the tree's height and crown: it does not imitate SpeedTree's, and it runs in Edit mode too. Its billboard LOD stays yours, with Unity's SpeedTree wind. Impostors need Convert, and only a tree whose last LOD is a mesh, not a billboard, can have one.

SpeedTree 9 is not tested (no test content). Its trees should be patched like any Shader Graph, but none has been rendered. Its wind values are laid out differently from SpeedTree 8's and are not read (its trees take none of SpeedTree 8's), so its trees would stand still. Convert gives them Drop-In Terrain Renderer's wind; that mapping is tested on test materials only.

Not drawn. A tree prefab with more than 8 LODs, or with any renderer that is not a MeshRenderer (a BillboardRenderer, for example), hands that terrain's trees to Unity, with the reason. SpeedTree 7 content has not been tested.

Drop-In Leaves, Bark and Grass

The shaders. Convert draws a prototype with Drop-In Terrain Renderer's own shaders, made from your materials: Drop-In Bark (trunks and branches), Drop-In Leaves (leaves and needles, with light through them) and Drop-In Grass (detail meshes and converted grass textures, bent by grass interaction). The converted materials are saved in Assets/Drop-In Terrain Renderer Generated/Converted, where you can edit them. Your own materials can use these shaders too (`Havoila Studio/Drop-In Terrain Renderer/Bark`, `Leaves` and `Grass`); impostors, grass bending, and the Drop-In Sky and Drop-In Water hooks need them. A converted grass texture draws with Drop-In Grass (or your renamed copy of it) only: set its converted material to Drop-In Leaves, Bark or the Impostor shader and the layer stays Unity-style grass, with the reason `its converted material is not Drop-In Grass`.

Wind. The Drop-In shaders follow one wind for the whole scene:

- Its source is `FoliageWind.Source` when a script sets it (`FoliageWind.SetSource(zone)`); null: back to the lookup), otherwise the active directional Wind Zone blowing hardest (looked for once a second). Drop-In Sky drives that Wind Zone, so the plants follow its weather with no setup. A turned or strengthened zone is followed over 2 s, so the plants never snap. The zone's Main, Turbulence and Pulse settings give the strength, the turbulence and the gusts.
- With no Wind Zone (or with the Wind module turned off), each material blows in its own Wind Direction Without a WindZone, with a default breeze.
- Each material shapes its plant's motion in its Wind settings (below): the bend, the branch sway and the flutter, each with its frequency, and the Wind Weights that say which parts move (Generated from the plant's height and crown, the mesh's vertex colours, or Park Map: the vertex colours of Drop-In Sky's Park Foliage plants).
- `FoliageWind.Hold()` keeps the plants still where they are (a shot, a cut scene); `Hold(seconds)` holds them at the same pose every time; `Release()` lets them move again. Holding the wind also holds grass interaction.
- In Play mode the wind follows game time (a paused game stills it); in Edit mode it runs on the real time.
- Other assets' shaders can read the same wind from four global vectors: `_HavoilaFoliageWind`, `_HavoilaFoliageWindPrevious`, `_HavoilaFoliageWindClock` and `_HavoilaFoliageWindParams`.

This wind does not move your own shaders (Keep mine), SpeedTree's (see Wind, under SpeedTree) or vendor plants.

Wind groups. To tune many materials at once, add a Foliage Wind to any object in the scene (Add Component > Havoila Studio > Drop-In Terrain Renderer > Foliage Wind). It has three groups, Trees, Bushes and Grass, each with four sliders that scale what its materials set:

- Strength scales the bend (each material's Bend), Branches the branch sway, Flutter the flutter of leaves and blades (bark has none): from 0 (that motion off) to 4.
- Speed scales the three frequencies, from 0.1 to 4. The plants sway on a running clock, so a change of Speed makes them jump once: tune it, do not animate it.

Each material picks its group with Wind Group, at the top of its Wind settings: Trees by default on Drop-In Leaves and Bark, Grass on Drop-In Grass (so converted details start in Grass); set Bushes on undergrowth, ferns, shrubs and young trees. Give all of a plant's materials (bark, leaves, billboard, every LOD) the same Wind Group: split between groups, its leaves sway off its trunk as soon as one group's sliders move. Without a Foliage Wind, or with every slider at 1, each material moves exactly as its own settings say. Only the Drop-In shaders follow it: impostors stand still, and your own shaders, SpeedTree's and vendor plants keep their own wind. One Foliage Wind is used at a time, the most recently enabled (the others' Inspectors say so). Sliders above 1 sway plants further out of their bounds, so the culling widens its margins by the largest Strength, Branches or Flutter in use (a little more is drawn near the screen's edges).

Scripts can set its groups' fields (`trees.strength`, `grass.speed`...): they are read every frame in Play mode; in Edit mode call `Apply()` after a change. Change Strength, Branches and Flutter gradually (a jump leaves one frame of mismatched motion vectors under TAA); for gusts use the Wind Zone's Pulse settings, which Speed does not scale. The shaders read three global vectors, `_HavoilaFoliageWindTrees`, `_HavoilaFoliageWindBushes` and `_HavoilaFoliageWindGrass` (each slider minus 1, so 0 when no Foliage Wind sets them).

Material settings, in the Inspector's groups (each shader shows the groups it has):

- Surface: base map and colour, normal map, metallic and smoothness (Mask Layout: Lit, SpeedTree Extra, or smoothness in the base map's alpha), occlusion.
- Blades (Grass): the tip colour and its gradient, the blade's height in mesh units, base occlusion and Ground Normal (how far the blades' lighting leans to the ground's).
- Alpha and sides: Alpha Cutoff, Alpha Mode (Cutout or Dithered), Render Face, Flip Back Normals.
- Detail (Bark): a detail mask, albedo and normal map.
- Light through (Leaves, Grass): transmission, thickness, tint, scatter power, shadowed and ambient transmission, and Additional Lights Transmit (point and spot lights, in Forward and Forward+ only).
- Terrain light (Grass, Leaves): Terrain Light, on by default on Grass and off on Leaves: on a lightmapped terrain, a detail plant takes the baked light of the terrain under it instead of the scene's ambient probe (see Terrain Light on detail meshes, under Render paths, platforms and URP features). Turn it on for ferns and bushes drawn with Leaves.
- Plant: colour variation between plants, crown normals and height (Leaves), SSAO strength.
- Wind: Wind Group (the Foliage Wind group that scales it), then as above.
- Grass interaction (Foliage Interactors): Bend (Grass).
- Weather and water (Drop-In Sky, Drop-In Water): porosity and wet gloss for rain, snow, the shore band, Hide Under Water (Grass).
- Distance fade (plain renderers): Fade End and Fade Range, for a GameObject drawn with these materials outside the terrain.
- Shadows: Shadow Normal Bias Scale.

Vendor packs

NatureManufacture and The Visual Engine plants go through the same automatic patching as any URP shader: nothing to set up in Drop-In Terrain Renderer, no project setting changed and nothing written to the vendor's files (the vendor's own install steps below still apply). They are drawn as yours (Keep mine); Convert keeps them as they are, with the reason a NatureManufacture shader (its own wind and look): kept as it is (or The Visual Engine).

NatureManufacture. Install NatureManufacture's own URP pack from its folder, the one for your URP version (NatureManufacture ships one per URP version; the tests used "URP 17.3 Unity 6.3", on Unity 6.3 and 6.6 alike). Tested: the Forest Environment Dynamic Nature beeches and a beech plant, as tree prototypes with LOD Groups. NatureManufacture

detail layers have not been tested. Known difference: a few small trees dither differently partway through a LOD cross-fade.

The Visual Engine. Install its Universal pack (the tests used "Universal 6000.0+", on Unity 6.3 and 6.6 alike) and keep its TVE Manager in the scene: without it, TVE plants draw nothing, in Unity's own rendering as well. Drop-In Terrain Renderer does not check for the manager. Keep TVE's folder whole: its shaders need its scripts to compile. Tested: TVE's demo plants as trees and one grass mesh as an instanced detail layer, with General Standard Lit and General Subsurface Lit. Other TVE shaders and features (elements, seasons, its own interaction) have not been tested. TVE's motion-vector pass has no instancing variant of its own; its copy gets one (its vertex function sets its instance up, see Hand-written shaders), so TVE plants draw TVE's own motion under With Wind, and under As Unity where the terrain asks for per-object motion (measured on TVE's demo tree under both: the motion lay on the tree's pixels only). TVE's motion has not been compared with Unity's: Unity's terrain drew that tree's motion thousands of pixels long (it has no LOD Group, see Motion vectors).

What vendor plants do not get. They move by their own wind systems (TVE's through its manager), not Drop-In Terrain Renderer's, so holding FoliageWind does not hold them. They get no impostors (those need Convert), no bending by grass interaction, no Drop-In Sky weather and no Drop-In Water hooks: those live in Drop-In Terrain Renderer's own shaders.

Updates and failures. Updating a vendor pack needs nothing from you: each re-imported shader's copy is made again. If a vendor shader cannot be patched or compiled (a pack for another URP version, or a vendor script failing on a newer Unity), that terrain's trees or details go back to Unity, with the reason; other terrains are not affected.

Other vendors. Plants from other packs have not been tested. Their URP shaders and Shader Graphs go through the same patching, and the Hand-written shaders rules decide whether a .shader can be patched. A shader that cannot be patched, such as a CG shader or one with its own procedural instancing, leaves its terrain to Unity, with the reason.

Terrain tools. MicroVerse, Gaia and Vista have not been tested. A tool that writes the TerrainData without Unity's change notices needs a call to the component's `Rebuild()` (or Read Again in its Inspector) or `RefreshDetails()`, or Watch Details At Runtime on for players (the editor watches in Edit mode).

Trees changed at runtime. Trees added or removed by script (`TerrainData.SetTreeInstances`, for example to fell a tree) are picked up from Unity's change notice. So are tree prototypes set by script (`TerrainData.treePrototypes`). A player does not compare the prototypes otherwise, since reading them makes garbage: after a change that sends no notice, call the component's `Rebuild()`. The editor also compares their count. In version 1 each change reads the trees of every taken-over terrain again, in one frame: tens of milliseconds and a few megabytes of garbage at 100,000 trees, more beyond (the first read of 115,742 trees on 12 terrains took 63 ms in the editor on our test machine; the cost grows with the tree count). Heightmap edits (`SetHeights`, `SetHeightsDelayLOD`, a sculpting brush) do the same at most four times a second while they go on, and once after the last; they read nothing again when no terrain whose trees Drop-In Terrain Renderer draws changed. Fell trees in batches rather than one per frame. Detail edits that send Unity's change notice (`SetDetailLayer`, for example) are updated where they changed. Updates per terrain are planned for 1.1.

Streaming terrains (additive scenes, tile streamers) have not been tested. In version 1, enabling, disabling, loading or unloading any terrain with Drop-In Terrain Renderer reads every terrain's trees again in that frame (the hitch above), and places every terrain's details again: Unity draws the grass meanwhile, for a few frames, with its CPU cost, and in Extended the grass past Unity's Detail Distance shrinks and grows back. In a player each built scene with Drop-In Terrain Renderer carries its own subset of the shader library, and loading one does the same. Loading or unloading a scene with baked light probes, even one without terrains (an interior, a town), reads every terrain's trees again too. Load or unload tiles and such scenes where a short hitch is hidden (a loading screen, a fade). Updates per terrain are planned for 1.1.

Terrains made at runtime have not been tested. A component added by script in a player starts with empty references, since the editor fills them when the component is added there. Components share them, so it draws while another component set up in the editor is loaded; otherwise copy Shader Library, Culling, Detail Compute, Wind Capture and Interaction Compute from one, or add the component from a prefab that has them. A player cannot patch, and a build carries only the copies its scenes' terrains draw: give such terrains prototypes that a terrain in a built scene also uses.

Grass streaming and garbage

How the near grass is placed. Near the cameras, Drop-In Terrain Renderer draws Unity's own placement: for each detail patch and layer it calls Unity's `TerrainData.ComputeDetailInstanceTransforms`, so every position, size and turn is Unity's. That call runs on the main thread and cannot be split (Unity's own detail drawing does this work on worker threads). So

placing is spread over frames, nearest patches first, within the component's Streaming Budget: 1 ms a frame by default (or 4% of the usual frame time when that is more, up to 8 ms), twice that in Edit mode, and four times that while catching up, that is while a camera or a `PrewarmDetails` point still has grass to place near it (after a jump, a scene load or a prewarm call). For a second after a camera jump, the budget is a quarter of the usual frame time instead when that is more, 16 ms at most. The usual frame time is the frame time outside catch-ups, so the slower frames of a catch-up do not raise their own budget. One call places one layer of one patch. A call predicted to run past what is left of the frame's budget waits for the next frame, unless it is the frame's first call, so a patch larger than the whole budget is still placed, alone. The prediction counts the patch's cells as Unity's placement does, and learns this machine's cost per grass instance from the calls it times. In Edit mode, each repaint places on a budget of its own. Until a camera's grass is all placed, Unity draws that camera's grass (the Inspector says `placing (N patches to go)`), so the image never has holes.

Teleports, cuts and respawns. A jump into dense grass costs main-thread time for a fraction of a second, or a little more, while the grass round the new position is placed (measured below). When your script knows where the camera will be, call `DropInTerrainRenderer.PrewarmDetails` before the jump:

```
using System.Collections;
using HavoilaStudio.DropIn.TerrainRenderer;
using UnityEngine;

public class Teleporter : MonoBehaviour
{
    public Transform player;

    // A scripted jump, started with StartCoroutine(TeleportSoon(target)): the grass round the destination is placed in
    // the 2 s before it, at the catch-up rate (four times the Streaming Budget).
    public IEnumerator TeleportSoon(Vector3 destination)
    {
        DropInTerrainRenderer.PrewarmDetails(destination);
        yield return new WaitForSecondsRealtime(2f); // (the point is wanted for 10 s of real time)
        player.position = destination;
    }

    // Behind a fade, started with StartCoroutine(TeleportBehindAFade(target, 0.5f)): 4 ms more placing in each frame of
    // the fade, on top of the frame's own, then the jump.
    public IEnumerator TeleportBehindAFade(Vector3 destination, float fadeSeconds)
    {
        for (float t = 0f; t < fadeSeconds; t += Time.unscaledDeltaTime)
        {
            DropInTerrainRenderer.PrewarmDetails(destination, 4f);
            yield return null;
        }
        player.position = destination;
    }
}
```

- `PrewarmDetails(worldPosition)` wants the grass round that point for ten seconds of real time, as a camera standing there would. While it has grass left to place, placing runs at the catch-up rate, nearest first: four times the Streaming Budget, 4 ms a frame by default. Up to eight points are wanted at a time; a call within 2 m of a wanted point extends that one. In the demo's densest grass at 60 fps, with Streaming Budget 1 ms, 2 s ahead was enough.
- `PrewarmDetails(worldPosition, maxMilliseconds)` also places at once, for about that many milliseconds of main thread, on top of the frame's own budget (the same rule: after the first call, a call predicted not to fit waits); the next frames place the rest at the catch-up rate. A 300 m jump into the demo's dense grass took about 110 ms of placing in all, so a single 50 ms call places about half of it: behind a fade, call it with a few milliseconds in each frame of the fade, as above. Call it from your scripts (Update, a coroutine, a loading step), not from inside rendering.
- A prewarm point never holds a camera up: the cameras keep their own grass while it is placed.

Measured in the demo (2026-10-19): a Unity 6000.3.25f1 development player (Mono, incremental GC), Ryzen 9 7950X3D, RTX 5080, Direct3D 12; the demo's 2 km valley before its revision (about 4.5 million grass clumps and ferns) in Match Unity, Streaming Budget 1 ms. Teleports and the CPU figures per second were measured with this version, in a setup built so that the main thread sets the frame: 1280x720 at render scale 0.5, no shadows, and 14 ms of stand-in game work every

frame (about 60 fps). Even there, 3 of the 8 jumps landed where the GPU set the frame; the teleport figures are for the other 5. The garbage, collection and 2560x1440 walking figures were measured with the version before, which placed the same calls.

- **Garbage.** Unity's call returns a new array: 24 bytes of garbage per grass instance, plus about 0.3 KB per call. Frames that place carry nearly all of it (92% walking, over 99% at 30 m/s). Frames without placing averaged 40-70 bytes in that version, about the size of two once-a-second reads inside rendering (the trees' prototypes and the scene's lightmaps) that players no longer make. In the demo: 0.03 MB/s walking, 0.08 MB/s at 5 m/s, 1.2 MB/s flying at 30 m/s at ground level, and 2-21 MB per 300 m teleport into dense grass (11.5 MB on average).
- **Collections.** With Unity's default incremental GC, a collection costs 0.8 ms of main thread (median; 2.8 ms at most), about one per 1.6 MB of garbage on the demo's 16 MB heap. Next to 300 MB of game data that never changed (a stand-in), it came once per 92 MB, at 1.4 ms; a game whose large heap keeps changing may pay more. With incremental GC off: 1.8 ms on a 16 MB heap, 54 ms on a 300 MB heap.
- **Walking and flying.** Placing fell on 0.4% of the frames walking and 1.5% at 5 m/s, at 2560x1440: those frames' main thread was 5.1 ms at most walking and 5.9 ms at 5 m/s, where the same paths with Unity's grass peaked at 3.8 and 8.8 ms. Per second, our detail work is less CPU than Unity's while moving (2.4 against 15 ms walking, 13 against 32.5 ms at 30 m/s), but ours is on the main thread, Unity's mostly on worker threads. Teleports are the exception: 95 against 71 ms of CPU per second, 39 ms of ours on the main thread against 6 ms of Unity's.
- **Teleports.** After a 300 m jump into dense grass, Unity drew that camera's grass for 0.54 s on average (1.3 s at most) while ours was placed. Those frames ran 3.4 ms more main thread than with Unity's grass (median; 4.7 ms at the 90th percentile). That is the catch-up budget, here 4 ms (a quarter of the 16 ms frame): placing stayed within it but for 0.1 ms at most, plus any garbage collection that ran inside a call (0.8 ms each, median). The same placing is spread over more frames: the first 60 frames after a jump took about 80 ms more in all (180 ms at most), and the worst frame's main thread was 34 ms against 29.5 ms with Unity's grass.
- **Teleports with PrewarmDetails 2 s ahead:** no catch-up after the jump, and the camera was never handed to Unity. The first 60 frames after a jump took 12 ms less than with Unity's grass on average (4 ms more at most). The placing moves into the 2 s before the jump: 2.8 ms on the frames that place (median; 5.7 ms at most, collections included), and those frames' main thread stayed below Unity's own worst jump frame (21.4 against 29.5 ms).

Generated shader copies

Where they live. Everything Drop-In Terrain Renderer makes goes to Assets/Drop-In Terrain Renderer Generated, outside the package, so updating Drop-In Terrain Renderer keeps it:

- Shaders/: one copy per source shader, named `Hidden/Drop-In Terrain Renderer/<name>` (grass: `Hidden/Drop-In Terrain Renderer/Compat/<name>`), so it never shows in shader menus. It is your shader's code (a graph's generated code, or the .shader file's text) with Drop-In Terrain Renderer's instancing added, compiled with FXC. Do not edit it: it is made again when its source changes.
- Materials/: one material per source material, saved under its source's name (`<name>.mat`; characters other than letters, digits, - and _ become _), and one per grass copy (`Compat_<shader>_<id>.mat`).
- Drop-In Terrain Renderer Shader Library.asset: the record of it all, and of your choices: Keep mine or Convert per prototype, class and wind overrides, impostor settings, and why a material could not be patched. Every terrain's component references it (Shader Library, under Advanced).
- Converted/ and Impostors/: converted materials and impostor bakes, once you use those features.

The folder works only while Drop-In Terrain Renderer is installed: the copies include its files.

When copies are made. Set Up Selected Terrains (or Set Up Every Terrain In The Open Scenes) and the component's Patch Materials button create the folder and library when missing, then patch the terrain's tree and detail materials at once. A material met later (a new tree prototype) is patched right after the frame that met it; until then that terrain is drawn by Unity, with `material X not patched yet`. A component added through Add Component gets the project's library, made when the project has none (one undo removes the component and its library reference; a library made for it stays in Assets/Drop-In Terrain Renderer Generated), and its first frame asks for its terrain's patches.

When copies are made again. You do not need to do anything:

- A source shader or Shader Graph imported again (saved, re-imported, or replaced by a package update) has its copy made again and checked.
- After a URP update, the grass copies are made again at the next script reload, and URP's own shaders that the update re-imports (Lit, for example) have their copies made again.
- After a Unity upgrade, every copy is compile-checked again.
- After a Drop-In Terrain Renderer update that changes its instancing code, every copy recompiles once: a one-time wait. Copies made by an earlier version are made again, one per editor update, in the background, the old copy drawing meanwhile; Set Up waits for them.
- Moving the Drop-In Terrain Renderer package folder rewrites the tree and detail-mesh copies with the new path. A move has not been tested with grass copies: if a Shaders/Compat_ copy then logs an include error, delete it, and it is made again the next time scripts compile.

Patch Materials does not rewrite a current copy: to make one copy again, re-import its source shader. Do not move or rename the Generated folder itself: its path is fixed, and copies elsewhere are no longer made again automatically.

Copies left behind. When a vendor package, a shader or one of your tree materials is removed from the project, Drop-In Terrain Renderer removes what it made from it at its next check, with nothing to do on your part. The check runs when a shader or a package is removed, after a script reload (so also when the editor starts), and first thing in Patch Materials and Set Up. That shader's copies in Shaders/ and the generated materials in Materials/ that nothing else uses go to your system's trash, their records leave the library, and one Console line names the shaders: [Drop-In Terrain Renderer] Removed the copies of N shaders that are no longer in the project (...): C shader copies and M materials from Assets/Drop-In Terrain Renderer Generated, moved to your system's trash. They are made again if those shaders come back. Patch Materials makes a returning shader's copies at once.

Only what is really gone counts: a shader still on disk that has not imported, or failed to, keeps its copies, and so does a package the project still lists but could not resolve (offline, for example). Nothing outside the Generated folder is removed. The automatic check waits for imports, compiles, player builds, package registration and test runs, then for 2 s without asset changes; in Play mode it waits for the return to Edit mode. It does nothing while a shader library has unsaved changes (File > Save Project) or does not load; Patch Materials and Set Up save the libraries first, and say so in the Console when one cannot be checked. A file that cannot be moved to the trash is named in the same line (Could not remove ...: delete it by hand.); the automatic check does not try it again in that editor session, Patch Materials does. A material you gave another shader has its generated material re-pointed at that shader's copy at the same check (the generated material keeps its GUID); until then Drop-In Terrain Renderer draws it with the old copy. An older copy of a shader still in the project that nothing uses is kept, and counted at the end of the line. Batch mode (CI) never removes anything by itself.

Version control. Commit the whole folder. Builds need it, and it holds what cannot be made again:

- your choices in the library (Keep mine or Convert, overrides, impostor settings) and your edits to converted materials;
- the impostor bakes, made by Set Up, "Convert those kinds and bake" and the Impostors section's Bake buttons, never on their own (about 0.8 MB per tree kind at 64 px, 3.3 MB at 128 px);
- one library for the whole team: without it, each teammate's first Patch Materials makes a new library asset, with a new GUID, and writes it into the scene;
- the copies a batch build (CI) uses: in batch mode the copies are not refreshed after a reload or checked, and nothing patches a new material, so the build relies on the folder as committed.

Vendor packs often stay out of version control. On a clone or a branch without a pack the folder has copies of, the first check after the editor starts finds that pack's shaders gone and removes their copies and generated materials (Copies left behind); they are made again, with new GUIDs, once the pack is imported. Import the packs before you open the project, or discard those changes in the Generated folder rather than committing them.

Opening the project in another Unity version checks the copies again and updates their records, so the library asset changes on that machine. The copies contain the full source of the shaders they were made from, vendors' shaders included: keep that in mind for a public repository.

One library. Point every Drop-In Terrain Renderer component at the same library. Terrains loaded together whose components reference different libraries (a duplicated library, scenes from two branches) are drawn from both, and new patches go to the first one that is an asset, with one Console warning: Terrains loaded together use different shader

libraries (...): new patches go to <library>. Point every Drop-In Terrain Renderer component at one library. The Inspector says the same under its status boxes. A build warns once per scene when the scene's library lacks a generated material another library holds (<scene>: its terrains' shader library ... has no generated material for ..., which ... has ...: the build leaves it to Unity.): point the components at one library, or press Patch Materials on those terrains' components, then build again.

A library that does not load (a merge conflict, a damaged file) is never written over, and no second library is made beside it. Patch Materials and Set Up log one error instead: The project's shader library could not be used: the shader library <path> does not load (a merge conflict or a damaged file?): restore it from version control or fix it, then press Patch Materials on a Drop-In Terrain Renderer component. Nothing was written over it.

If you delete the folder. Every terrain's trees and details go back to Unity, with no shader library on the component (press Patch Materials or Set Up, which makes one). Nothing rebuilds it on its own for components already on the terrains: select the terrains and press Patch Materials. Conversions and converted materials are lost (convert again), and impostors must be baked again.

Removing Drop-In Terrain Renderer. Remove the Drop-In Terrain Renderer components from your terrains: Unity draws them again at once, and your terrains, prefabs and materials were never changed. Then delete Assets/Drop-In Terrain Renderer Generated, then the package folder. Deleted the other way round, the copies log shader errors and the terrains show a missing script. On Unity 6.3 to 6.5, each scene without baked occlusion data in a player or AssetBundle built with the package holds the hidden "Drop-In Terrain Renderer Occlusion Mark" object (Tree casters per cascade), so build your AssetBundles again after removing the package, and do not load bundles built with it into a player built without it: each such scene would log a missing script when it loads (nothing else changes).

Builds. No Always Included Shaders setup is needed. Each component references the library, so the generated materials, their shaders and the instancing variants for the keywords the generated materials hold go into the player (see "Edit your original materials" for a keyword changed later). Each scene gets only what its terrains draw (their materials, their conversions, their impostors that are on and baked for the tree kinds of terrains with Draw Trees and Impostors on, and the grass copies; a terrain with either switch off ships none of its kinds' impostors unless another terrain asks for them, so turning a switch on from a script in a player leaves those trees ending at their last LOD); the scene and library assets are not changed. Scenes built separately and loaded together are merged. This build path was tested in Windows players with D3D12, D3D11 and Vulkan; what each of them drew is under "Graphics APIs".

- Before a build, a warning names any grass copies older than their sources: Open the project in the Unity editor once, which remakes them, then build again. A build without a graphics device cannot make them again.
- The same warning names copies left behind (see above) and materials drawn through a stale or out-of-date copy: the build leaves those materials' terrains to Unity wherever its scenes use them, and deletes nothing. Fix: press Patch Materials on a Drop-In Terrain Renderer component, which first removes the stale copies and re-points the rest, then build again (for a command-line build, open the project in the Unity editor once first).
- Tree and detail copies are not checked before a build, and a player cannot patch: a material never patched in the editor leaves its terrain to Unity in the player. The build names such materials and grass shaders, per scene, in one warning, with the components that have no shader library (a checkout without Assets/Drop-In Terrain Renderer Generated, for example). After adding prototypes, open the scene or press Patch Materials before you build.
- **What a build leaves out.** A player carries only the shader variants something in it can draw. URP first removes what your URP assets never use, as it does for every shader. Then Drop-In Terrain Renderer removes the variants of its own shaders that nothing draws:
 - Tree copies and grass copies keep only the variants of Drop-In Terrain Renderer's own draws (procedural instancing, single-pass instanced XR's included; none with DOTS instancing): nothing else draws them. So a tree copy adds less build time and size, and a vendor shader's compile errors in variants we never draw are not repeated under the copy's name.
 - The Impostor shader keeps only those too: only Drop-In Terrain Renderer draws impostors.
 - Drop-In Leaves, Bark and the Impostor shader leave out the lightmapped variants of our draws: trees and the detail meshes we draw take their light from light probes (and Terrain Light), never from a lightmap of their own. Drop-In Grass keeps its baked-lightmap variants, since converted grass textures on a lightmapped terrain use the terrain's lightmap; it leaves out only those for realtime GI (Enlighten) lightmaps, which none of our draws use.

- What Unity draws keeps every variant: Drop-In materials on your prefabs and GameObjects, Unity's own draws of a terrain it takes back, reflection probes.
- Your own shaders, URP's and other packages' are never touched; only our copies of them are trimmed.

This does not depend on URP's Strip Unused Variants setting. The build log says it once: [Drop-In Terrain Renderer] The build leaves out the shader variants nothing draws (README, Builds).

- **Build size.** Measured on the demo (a Windows Direct3D 12 development player, Unity 6000.3.25f1, lightmapped terrains): Drop-In Leaves and Bark each come out about 18 % smaller (16 % fewer variants), from 19.2 to 15.8 MB and from 15.6 to 12.7 MB of compressed shader code (XR plug-in installed, which adds single-pass instanced XR's variants); Drop-In Grass is unchanged there (20.2 MB: with no realtime GI in the demo, Unity's own stripping already leaves out what this would). The player is 6.6 MB smaller in all (761 to 754 MB, under 1 %). The Drop-In shaders are larger than URP Lit or Terrain Lit (2.1 and 11.9 MB in the same player): besides Unity's plain and GPU-instanced variants for your materials on prefabs and GameObjects, each carries the procedural instancing variants of our draws. Impostors add about 0.8 or 3.3 MB per tree kind (64 or 128 px frames); the runtime code is 0.69 MB (Mono, measured in that development build).
- **Keep every variant.** Add `DROPIN_TERRAIN_RENDERER_KEEP_VARIANTS` to Project Settings > Player > Scripting Define Symbols, or to the active build profile's, then build again; the build log then says `This build keeps every shader variant`. A define that only a build script passes (`BuildPlayerOptions.extraScriptingDefines`) is not read. If trees or grass look wrong in a player but not in the editor (pink, or plants missing or piled up), try this first and tell us which shader and which view: it means we left out a variant your project draws. With it, builds take longer and are larger, and a vendor shader's compile errors in variants we never draw can show again under its copy's name.
- The Impostor material draws only through Drop-In Terrain Renderer. On a Mesh Renderer it shows in the editor, not in a player.
- AssetBundles and Addressables have not been tested.

Reflections

In Reflections (on the component, off by default) decides who draws the trees in Reflection cameras (reflection probes):

- Off: reflection probes show Unity's own trees, at Unity's cost.
- On: Drop-In Terrain Renderer draws its trees into them.

The image is the same either way. On a small test bench (80 trees without LOD Groups, two grass layers, a Reflection camera rendering a 256 px cubemap as a realtime probe does, wind held) the cubemaps differed by 0.007/255 on average, and the main-thread time per cubemap was the same: 7.6-8.0 ms with it on, 7.8 ms off. GPU time was not measured. Leave it off unless a realtime reflection probe that updates often sees many LOD Group trees. Then turn it on and compare the main thread in the Profiler: the two settings were compared on this small bench only. Grass does not depend on this setting: a reflection camera near a camera whose grass is placed gets Drop-In Terrain Renderer's grass, others get Unity's.

Planar reflections and other nested cameras (rendered inside another camera's rendering, such as a water mirror or a portal) follow their camera's type. One left as a Game camera, the usual case, gets Drop-In Terrain Renderer's trees whatever In Reflections says, and its grass when it is near a camera whose grass is placed. One its script marks as a Reflection camera follows In Reflections. A mirror camera under a water plane matched Unity's image. Extended mode (the component's Detail Mode that thins details with distance to reach further) draws its far details only for top-level cameras, so nested cameras show details only out to the Exact Radius (where Unity's own placement ends). Cameras nested two deep or more get Unity's own.

Reflection probes outside Forward+. In Forward and Deferred, Unity picks a reflection probe per tree renderer, where Drop-In Terrain Renderer makes one draw per tree kind. Whether reflections on trees differ there has not been tested; the test is planned for a later version, with per-cell probe draws. In Forward+, reflections on the URP terrain demo's trees matched Unity's (in the editor; the SpeedTree leaf difference in Known issues aside).

Known issues

- **SpeedTree 8 leaves.** Drop-In Terrain Renderer's draws show a fine speckle along every leaf edge and in the leaves' ground shadows, compared with Unity's (the leaves cover about 1.7% fewer pixels; trunks match). The LOD cross-fade band is not the cause. A fix is planned for 1.1. Until then no workaround keeps SpeedTree's own shading: Convert

changes the look, and turning the component off gives the terrain back to Unity.

- **One prototype hands back the whole terrain.** The fallback is per terrain (see "Why does Unity draw my trees?"). A per-kind fallback is planned for 1.1.
- **Hand-written shaders without URP's instancing setup** are drawn by Unity, with the whole terrain (see the item above). The reason names the lines to add (see Hand-written shaders).
- **Streaming terrains and runtime tree edits rebuild every terrain.** A tree or height edit by script, a terrain lit by baked probes that stops moving, a scene with baked light probes loaded or unloaded (even one without terrains), or a terrain enabled, disabled, loaded or unloaded reads every terrain's trees again in one frame (tens of milliseconds and a few megabytes of garbage at 100,000 trees, more beyond), and a terrain added or removed also places every terrain's details again (see Trees changed at runtime and Streaming terrains). Updates per terrain are planned for 1.1.
- **Tree edits in the editor rebuild every terrain too.** A tree edit in Edit mode (a stroke of the Paint Trees brush, for one) reads every terrain's trees again at the next frame drawn, as a runtime edit does. In the demo, with 522,116 trees, such a rebuild took about 1 s in the Unity 6.3 editor (Direct3D 12, our test machine): most of it Drop-In Terrain Renderer's own work, about 0.1 s the trees' light probes, the rest the material copies made again.
- **Animate Cross-fading is not read.** Trees inside the Cross Fade Band stay dithered, where Unity fades them only while they switch. Fix: set Cross Fade Band to 0 (LODs then pop).
- **Reflection probes outside Forward+ and Adaptive Probe Volumes** have not been compared with Unity's image (see above).
- **Shadowed point and spot lights.** Several of them in dense foliage cost more than in Unity (+0.65 ms for grass with 4 shadowed point lights at 4K; the trees' share has not been measured). Punctual Shadow Radius limits what casts into them only while the main light casts no shadows: under a shadowed sun, the sun's casters (those whose shadow falls in view) are drawn into each shadowed spot and point light too (Unity culls casters per light). Turn off the shadows of spot and point lights that do not need them. Per-light shadow buckets are planned for a later version.
- **Billboard grass, and Grass-mode detail meshes with GPU instancing off,** cost a little more GPU time than Unity's own draw (+0.38 and +0.57 ms on a 4K bench); grass cards and Vertex Lit meshes cost the same as Unity's. These were measured before version 1's grass vertex changes and have not been measured since. Instanced detail meshes in Match Unity cost about Unity's GPU time: 0.95x on the MeadowLite bench (2560x1440, D3D12 development player; see "Grass in Match Unity", under A small or negative result).
- **SSAO on Drop-In Grass and Leaves.** The depth-normals pass uses the meshes' own normals, not the grass's ground lean or the leaves' crown normals, so SSAO sees other normals than the lit image.
- **Light through leaves in Deferred.** Drop-In Leaves and Grass let only the main light and the ambient light through in Deferred. Point and spot lights shine through them in Forward and Forward+ only (Additional Lights Transmit, off by default).
- **Weather and water hooks** (Drop-In Sky's wetness and snow, Drop-In Water's shore band and hiding under water) work only on Drop-In Terrain Renderer's own shaders. Fix: Convert the prototype. Drop-In Water's surface map covers about 48 m round the camera: beyond it, plants under water stay drawn, as Unity draws them.
- **Impostors cast no shadows** in version 1. The Inspector warns where one starts inside URP's shadow distance.
- **An editor in the background patches slowly.** Patching runs when the editor gets to it: `material X not patched yet` can stay for minutes while the editor is not focused. Fix: focus the editor.
- **"Ran out of Graphics Ring Buffer space" in the editor.** In the D3D11 or Vulkan editor (not D3D12), Unity can log this once in the first frame of a view: when a large scene opens, when Play mode starts and when it ends. In that frame Unity's render thread has one-time work of its own (its detail batches, the editor's fonts) while the trees are drawn for the first time, and it falls behind. The main thread waits for it (about 60 ms in the demo, in a first frame of over a second); later frames are not affected. In the D3D11 editor, starting it with `-gfx-ring-buffer-size 50331648` (48 MB: enough for the demo at 3840x2160) silences it; this was not measured in the Vulkan editor.
- **Undo after Delete converted** brings back the conversion's record, not the deleted material (it is in your system's trash).
- **Undo across automatic patches.** Undoing "Remove Stale Shader Copies" (Patch Materials' first step), a bake or an impostor edit after Set Up or an automatic patch has written to the library can drop the records written since: a tree kind loses its impostor (bake it again), or a second generated material appears for one of yours (`Bark 1.mat` ; the first stays unused in Assets/Drop-In Terrain Renderer Generated/Materials and can be deleted).
- **Extended mode's far field** briefly holds a readable copy of every grass texture while it builds its atlas: a one-time memory spike at load.

Planned for 1.1.

- the SpeedTree 8 leaf fix;
- a per-kind fallback, so that one prototype Unity draws no longer hands back the whole terrain;
- updates per terrain: a tree or height edit, a moved lit terrain or a streamed tile reads only its own terrain again, and the other terrains keep their placed details.

Planned later (no version yet).

- the reflection probe test outside Forward+, with per-cell reflection probe draws;
- a render check under Adaptive Probe Volumes;
- per-light shadow buckets, for shadowed point and spot lights in dense foliage;
- a Drop-In wind and weather sub graph for your own Shader Graphs;
- DOTS instancing variants of the Drop-In shaders, for the GPU Resident Drawer;
- impostor shadow casters;
- impostors for trees whose last LOD is a SpeedTree billboard or that have 8 LODs (the impostor replacing the last LOD);
- impostors baked from your own shaders, if a test shows the capture works for them.

SpeedTree 9 wind is not scheduled yet: it waits for SpeedTree 9 test content.

How this was tested

Each test scene was rendered by Unity and by Drop-In Terrain Renderer from the same camera in Windows D3D12 players, and the two images were compared pixel by pixel: the mean difference over the vegetation's pixels, on a 0-255 scale, at eye level and from above. Unity rendering the same scene twice gives its own run-to-run noise, the floor for any comparison. The content kept its own materials (Keep mine). For wind, both drew with wind on and time running, in the same frame; the motion ratio is how far Drop-In Terrain Renderer's plants move against how far Unity's do (1.00: the same). No run logged an error or warning naming Drop-In Terrain Renderer's code. All tests ran on our test machine, with an NVIDIA RTX 5080; the timings in Known issues are at 3840x2160 on it unless they say otherwise. AMD and Intel GPUs have not been tested. The benchmark's results are under "Did it help? Measuring your scene".

Content	Unity 6.3: Drop-In Terrain Renderer against Unity (Unity against itself)	Unity 6.6: Drop-In Terrain Renderer against Unity (Unity against itself, where measured)	Wind: motion ratio, 6.3 / 6.6
URP Lit and Simple Lit trees and detail meshes	0.92 / 0.21 (0.88 / 0.18)	0.91 / 0.22 (0.89 / 0.20)	1.00 / 1.00
Unity's grass Shader Graph (detail meshes)	0.33 / 0.42 (0.31 / 0.42)	0.32 / 0.41 (0.30 / 0.41)	not measured alone (part of the SpeedTree 8 scene)
SpeedTree 8 (Unity's URP terrain demo)	3.96 / 5.65 (0.33 / 0.05)	3.96 / 5.65 (0.33 / 0.05)	1.02 / 1.02
NatureManufacture beeches	0.16 / 1.84 (0.07 / 0.02)	0.17 / 1.85 (0.08 / 0.02)	0.99 / 1.00
The Visual Engine, manager in the scene	0.89 / 0.30 (0.82 / 0.11)	0.88 / 0.33 (0.82 / 0.12)	1.00 / 1.00
Unity-style grass textures, one view each: billboards; cards	0.68; 1.99 (cards: 0.80)	0.68; 1.99 (cards: 0.79)	not measured / 0.99; 1.05
Drop-In Bark, Leaves and Grass, against Unity drawing the same materials	Bark 0.006 / 0.002; Leaves 0.140 / 0.073 (0.135 / 0.070); Grass 1.24 / 1.25	Bark 0.006 / 0.002; Leaves 0.148 / 0.080 (0.146 / 0.079); Grass 1.22 / 1.25 (0.81 / 1.21)	not measured

Values are eye level / from above, out of 255. With wind running, SpeedTree 8 differed from Unity's by 4.0/255; a still Drop-In Terrain Renderer frame against Unity's moving one differs by 19.5, so the trees move as Unity's do. The Visual Engine: 0.88 against 34.5. The NatureManufacture trees barely move in this scene. Unity 6.6 gave the same: SpeedTree 8 4.0 against 19.5, The Visual Engine 0.89 against 34.6.

Earlier tests: the Drop-In shaders in D3D12 players on Unity 6.3 and 6.6, and in XR (Unity's Mock HMD, single-pass instanced and multi-pass; see XR above). An earlier build ran instanced detail meshes and Unity-style grass textures in D3D11, D3D12 and Vulkan players: the detail meshes within Unity's own noise on all three, grass cards and billboards on Vulkan above the D3D12 numbers (cards 2.48 against 1.99, where Unity against itself is 0.80); that build came before version 1's grass vertex changes. Forward and Deferred were checked in the editor. Main-thread savings were measured on LOD Group (SpeedTree) trees; NatureManufacture and The Visual Engine content was compared for the image only.

Quality levels and options menus

Your quality levels. Drop-In Terrain Renderer follows the active quality level. Switch it from your menu with `QualitySettings.SetQualityLevel(index, true)`, as usual: there is nothing else to call.

- LOD Bias and Maximum LOD Level are read by each camera, every frame.
- Terrain Setting Overrides: a level that overrides Tree Distance, Detail Distance, Detail Density, Billboard Start or Fade Length draws the level's value, as Unity's terrain does. Pixel Error is used too (the ground under the shadows takes the larger of the level's and the terrain's). Basemap Distance shades the terrain itself.
- Detail Density, the terrain's or the level's, thins only the detail layers whose Affected by Density Scale is on (the layer's setting; `DetailPrototype.useDensityScaling`, off for a layer made by script unless set). The other layers keep their density, in Unity's terrain and ours alike. Every detail layer of the demo has it on, and the demo's valley terrain has a Detail Density of 0.7.
- Billboard Start and Fade Length (the level's or the terrain's) end the trees without a LOD Group, as in Unity's URP terrain, which draws no legacy billboard: they are drawn to Billboard Start plus Fade Length, cast shadows to Billboard Start, and end there (or at Tree Distance, if nearer; see Trees without a LOD Group). LOD Group trees do not use them. Max Mesh Trees is not read: it changed nothing in Unity's URP terrain in our tests. To draw trees without a LOD Group farther, raise Billboard Start: Unity's terrain follows it too, so the comparison stays like for like.
- The level's URP asset (shadows, LOD Cross Fade, render path) is read every frame.

What a switch does (as designed and checked by the package's editor tests; how long each takes has not been measured in a player yet):

- A new LOD Bias, Tree Distance, Billboard Start, Fade Length or Tree LOD Bias Multiplier, or a lower Detail Distance: nothing is built again (the terrains' small records are uploaded again).
- A higher Detail Distance: the grass of the new ring is placed over the next frames, as after a jump. Unity draws that camera's grass meanwhile.
- A new Detail Density on a terrain with a layer Affected by Density Scale, or a level that switches Unity's legacy detail distribution (`QualitySettings.useLegacyDetailDistribution`): the details are placed again from the start, nearest first, as when a terrain is loaded (see Streaming terrains). Unity's terrain draws them meanwhile, and in Extended the grass past Unity's Detail Distance shrinks and grows back. Placing everything again also makes the details' buffers again, which a jump does not; for the placing alone, a 300 m jump into the demo's dense grass left a camera's grass to Unity for 0.54 s on average (see Grass streaming and garbage). A terrain with no such layer places nothing again.
- A new Maximum LOD Level with impostors in reach: the trees are read again once, in one frame (see Trees changed at runtime).

Tree Distance and Detail Distance overrides. With a Tree Distance override, Drop-In Terrain Renderer can hide a terrain's trees from Unity only together with its details. So Unity draws that terrain's trees whenever its grass is not ours: for a moment after a jump, a higher Detail Distance or a new Detail Density. A terrain with no details for Unity to draw is not affected (by design, checked in the editor). The same holds the other way round for a Detail Distance override. LOD Group trees ignore Tree Distance anyway, so we recommend not overriding it, or using the distance scale below.

Sliders in an options menu. Three scales act on every terrain with an enabled Drop-In Terrain Renderer component:

Scale	Range	Acts on	Cost of a change
<code>DistanceScale</code>	0.1 to 2	Detail Distance; Tree Distance and Billboard Start (trees without a LOD Group); Max Distance and Extended's Detail Reach and Exact Radius when set, whatever the quality level	Lower: nothing built again. Higher: the new ring is placed, as above. Above 1 there is more grass all along, ours and Unity's: at 2, up to four times as much to draw, keep in Detail Memory and place while moving.
<code>DensityScale</code>	0 to 1	Detail Density: the detail layers Affected by Density Scale (at 0, none of them is placed)	On a terrain with such a layer, its details are placed again (as above; in Extended the far grass shrinks and grows back). Elsewhere nothing. A slider that calls it at each step places them twice: at its first step, and once it has rested for 0.25 s (steps further apart: at each).
<code>LodScale</code>	0.1 to 4	Tree LOD Bias Multiplier: where every tree switches LOD and impostor	Nothing is built again.

- Set them with `SetDistanceScale`, `SetDensityScale`, `SetLodScale`, or all three with `SetQualityScales`, on `DropInTerrainRenderer`.
- A density of 0 turns off the detail layers Affected by Density Scale: Unity's terrain places none of them either, so neither do we, and a terrain left with no layer has no details to draw. The other layers keep their density. In the demo every layer has it on, so the density scale thins all of its grass and plants.
- They write each terrain's Detail Distance, Tree Distance, Billboard Start, Detail Density and Tree LOD Bias Multiplier: its own value times the scale (Billboard Start with Tree Distance, so the distance scale moves the end of the trees without a LOD Group too). Unity's terrain reads the same values wherever it draws instead (a frame or a camera Drop-In Terrain Renderer leaves to Unity, Measure's Unity side).
- Each terrain's own values are kept: they come back when a scale is 1 again, when the component is disabled, and when Play mode ends. In Play mode the component's Inspector shows the scales and the terrain's own values. A terrain open in Prefab Mode is never scaled.
- A script that reads these terrain values in Play mode sees the scaled ones. One that writes one keeps its value, and the scale applies to it from the next call. So a terrain that a script sets up after enabling its component (a tile streamer writing its distances) stays unscaled until then: set a terrain up before enabling its component, or call `SetQualityScales` again once it is streamed in.
- A quality level that overrides a value decides that value. The scale applies again on a level without the override.
- They work in Play mode and players only. Each Play session and each launch starts at 1, so restore the player's choice at startup, as the example does once the first scene is loaded. The scales then stay set across scene loads: terrains loaded or enabled later take them, with nothing to call again. Called during rendering (a camera callback), a change applies at the next frame.
- To compare with Unity's terrain at the same settings, use Measure or the demo's `Space`, which keep the scales. Disabling the component instead puts its terrain's own values back.

```
using HavoilaStudio.DropIn.TerrainRenderer;
using UnityEngine;
using UnityEngine.UI;

public class VegetationOptions : MonoBehaviour
{
    public Slider grassDistance; // 0.25 to 2: the grass, and the trees without a LOD Group
    public Slider grassDensity; // 0 to 1: the detail layers Affected by Density Scale
    public Slider treeDetail; // 0.25 to 2
    public Camera gameCamera;

    // Each launch starts at 1: the player's saved choice, once the first scene is loaded. The scales stay set across
    // scene loads, and terrains loaded or enabled later take them.
    [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.AfterSceneLoad)]
    static void Restore() =>
        DropInTerrainRenderer.SetQualityScales(PlayerPrefs.GetFloat("grassDistance", 1f),
                                                PlayerPrefs.GetFloat("grassDensity", 1f),
                                                PlayerPrefs.GetFloat("treeDetail", 1f));

    void OnEnable()
    {

```

```

        grassDistance.SetValueWithoutNotify(DropInTerrainRenderer.DistanceScale);
        grassDensity.SetValueWithoutNotify(DropInTerrainRenderer.DensityScale);
        treeDetail.SetValueWithoutNotify(DropInTerrainRenderer.LodScale);
        GreyOut(0, QualitySettings.GetQualityLevel());
        QualitySettings.activeQualityLevelChanged += GreyOut;
    }

    void OnDisable() => QualitySettings.activeQualityLevelChanged -= GreyOut;

    // A quality level that overrides a value decides it: grey its slider out. (With LOD Group trees the distance scale
    // moves only the grass, and Max Distance when set; with trees without a LOD Group, keep the distance slider while
    // the level leaves TerrainQualityOverrides.TreeDistance and BillboardStart free.)
    void GreyOut(int previous, int current)
    {
        var overrides = QualitySettings.terrainQualityOverrides;
        grassDistance.interactable = (overrides & TerrainQualityOverrides.DetailDistance) == 0;
        grassDensity.interactable = (overrides & TerrainQualityOverrides.DetailDensity) == 0;
    }

    // The menu's Apply button (not each slider step: a new density places the grass again).
    public void Apply()
    {
        bool toPlace = grassDensity.value != DropInTerrainRenderer.DensityScale
            || grassDistance.value > DropInTerrainRenderer.DistanceScale;
        DropInTerrainRenderer.SetQualityScales(grassDistance.value, grassDensity.value, treeDetail.value);
        PlayerPrefs.SetFloat("grassDistance", grassDistance.value);
        PlayerPrefs.SetFloat("grassDensity", grassDensity.value);
        PlayerPrefs.SetFloat("treeDetail", treeDetail.value);
        // Grass to place (a new density, a longer distance): about 50 ms of it now, behind the menu; the rest over the
        // next frames.
        if (toPlace) DropInTerrainRenderer.PrewarmDetails(gameCamera.transform.position, 50f);
    }
}

```

Avoiding a hitch.

- Apply on the menu's confirm, not on every slider step. A slider that sets the density at each step places the grass twice: at its first step, and once it has rested for a quarter of a second.
- Placing goes on behind a paused menu: it runs on unscaled time.
- `PrewarmDetails(position, 50f)` right after a new density or a raised distance places about 50 ms of the grass at once, behind the menu, and the next frames place the rest (a 300 m jump into the demo's dense grass took about 110 ms of placing in all: see Grass streaming and garbage). After a lowered distance or a new LOD scale there is nothing to place.
- While the menu is open, a larger Streaming Budget on the components places faster (the first enabled component's value is used).
- A distance scale above 1 means more grass all along, ours and Unity's (up to four times as much at 2): raise Detail Memory with it, or Unity draws the grass that does not fit.
- Each placement makes Unity's 24 bytes of garbage per instance (see Grass streaming and garbage).

Why does Unity draw my trees?

When Drop-In Terrain Renderer cannot draw something, Unity keeps drawing it: nothing vanishes. The terrain's Inspector then says "Unity draws this terrain's trees" (or details), then the reason and its fix. For a material, the reason names the prototype, the material and its shader: Oak: material Bark uses Legacy Shaders/Diffuse, a built-in shader (no source to patch). Fix: ... When Unity would draw nothing either (no trees, a Tree Distance of 0), the Inspector says "No trees to draw" instead.

Per terrain. One prototype that cannot be drawn hands that terrain's trees (or details) back to Unity, all of them: they still draw, as Unity draws them, only without Drop-In Terrain Renderer's savings. Other terrains are not affected, except by the project-wide reasons below. Fix the material, and the terrain is drawn by Drop-In Terrain Renderer again. When its shader is URP Lit, Simple Lit, Unity's SpeedTree 8 or 9 or Drop-In Sky's Park Foliage, converting the prototype also works (the Look section): Convert draws those with Drop-In Terrain Renderer's own shaders. Convert keeps any other shader as yours, so it does not help there: the Inspector's fix names Convert only for the materials Convert takes. The status names the first reason found; the Tree kinds section (inside Trees) gives each tree kind's own reason.

The reasons and their fixes.

- ..., a built-in shader (no source to patch): one of Unity's built-in shaders (Standard, Legacy Shaders, Nature/Tree Creator, which do not draw correctly in URP anyway). Fix: give the material a URP shader (URP Lit, for example); Window > Rendering > Render Pipeline Converter upgrades Standard materials.
- ..., made by another importer from <file> (no .shader text to patch): a shader that no .shader or .shadergraph file holds, such as one a tool builds from its own file type. Fix: use a URP shader or a Shader Graph.
- ..., a CG shader (no HLSLPROGRAM): shaders written in CG (CGPROGRAM blocks, made for the built-in pipeline) are not patched. Fix: give the material a URP shader (URP Lit, for example).
- ..., which has no HLSL program of its own (fixed-function or UsePass passes only): nothing in the shader to patch. Fix: use a URP shader.
- ..., whose HLSLPROGRAM block at line N has no ENDHLSL line of its own: an ENDHLSL that follows code on its line (return o; } ENDHLSL). Fix: put ENDHLSL at the start of its line.
- ..., set up for another renderer's instancing (procedural:X): a shader, or a Shader Graph (Graph Settings' custom pragmas on Unity 6.6), made for another renderer's procedural instancing, whose pragma Unity reads outside any #if. Fix: use a version of the shader without that setup.
- ..., which failed to import (Unity draws it pink): the shader (or Shader Graph) is still in the project but has errors, or its files (a sub graph, an include) are missing. The reason names the shader's file, since Unity has replaced the shader with its error shader. Fix: fix the shader, then press Patch Materials. A shader imported again later (a sub graph arriving, for example) is patched again by itself.
- ..., which is no longer in the project: the material's shader file was deleted (a vendor package removed, for example). Fix: give the material another shader, or bring the shader back.
- ..., a Shader Graph whose generated code could not be read: this Shader Graph version changed what Drop-In Terrain Renderer reads. Fix: update Drop-In Terrain Renderer (for Unity's SpeedTree 8 or 9 graphs, Convert also works).
- ..., whose source has no Shader "name" line: the shader file has no Shader "name" declaration. Fix: add a Shader "name" line, or use a URP shader.
- ..., custom HLSL without instancing setup (pass X, vertex function Y): a hand-written shader whose vertex function does not set up the instance ID; drawn by Drop-In Terrain Renderer, every tree would stand at the terrain's origin. Fix (as the reason words it): add UNITY_VERTEX_INPUT_INSTANCE_ID to the vertex input struct and call UNITY_SETUP_INSTANCE_ID(input) first in the vertex function, as URP's own shaders do (see Hand-written shaders above).
- ..., whose pass X has no instancing variant (no #pragma multi_compile_instancing): a pass Drop-In Terrain Renderer draws (untagged, forward, forward-only, G-buffer, shadow caster, depth, depth normals) cannot draw instances. A pass whose vertex function is seen to set its instance up gets the pragma in its copy; this reason means it could not be seen. Fix: add the pragma and the instance setup to that pass.
- ..., whose pass X comes from UsePass "...": another shader's pass, which the copy cannot patch. Fix: write that pass in the shader itself.

- `...`, whose patched copy has errors (`...`) or whose patched copy does not compile for instanced drawing (pass X, `...`): the shader compiles on its own but not with Drop-In Terrain Renderer's instancing (per-renderer data, for example). The message names the first error; `;` its source asks for DXC (`#pragma use_dxc`) and its copy compiles with FXC follows it when the shader asks for DXC (code that needs DXC does not compile as an FXC copy), and `;` its source also holds another renderer's procedural instancing (`procedural:X`) when an optional one sits under an `#if`. The compile check runs in the background after the copy is made, so the terrain can be drawn by Drop-In Terrain Renderer for a moment before this shows. Fix: follow URP's own shaders' instancing setup (see Hand-written shaders above). If it appears after a Unity or URP update, first re-import the source shader or Shader Graph (right-click it, Reimport): its copy is made again and checked, and the terrain goes back to Drop-In Terrain Renderer if it passes. A Drop-In Terrain Renderer update makes its copies again by itself.
- `material X not patched yet`: the copy is being made (a moment after the material first appears); it goes by itself once the editor is focused. A player cannot patch: a material never patched in the editor leaves its terrain to Unity in the build (the build's warning names it), so press Patch Materials after adding prototypes.

Tree prototypes.

- `<prefab>`: more than 8 LODs: Drop-In Terrain Renderer draws LOD Groups of up to 8 LODs. Fix: reduce the LOD Group to 8 LODs.
- `<prefab>`: a SkinnedMeshRenderer (not drawn by Drop-In Terrain Renderer yet), or another renderer type such as a BillboardRenderer: only MeshRenderers with a MeshFilter are drawn. Fix: build the prefab from MeshRenderers.
- `<prefab>`: no mesh renderer to draw: the prefab has no MeshRenderer with a mesh.

Terrain and Quality settings.

- `Draw Trees is off`: the component's own switch, in the header of its Trees section (Draw Details: the Grass and Details section's).
- `the terrain's Preserve Tree Prototype Layers is on (...)`: each tree would need its prefab's layer. Fix: turn it off in the Terrain's settings.
- `the quality level overrides the terrains' Tree Distance (Quality settings, Terrain Setting Overrides): ...` (or Detail Distance): with that override, Drop-In Terrain Renderer can hide a terrain's trees only together with its details (or the other way round), so Unity draws them whenever Drop-In Terrain Renderer does not draw both. A terrain with no details (or no trees) for Unity to draw is not affected. Fix: remove the override, or let Drop-In Terrain Renderer draw that terrain's trees and details both (see Quality levels and options menus).
- "No trees to draw" (or details), where Unity draws none either: the terrain has no TerrainData, no trees or no details; the Terrain component is disabled; its Draw Trees And Foliage is off; its Tree Distance or Detail Distance (or the quality level's) is 0; its Detail Density (or the quality level's) is 0 and every detail layer is Affected by Density Scale; a terrain tool hides them during its stroke (editor only).

Detail layers.

- `more than 255 detail layers`.
- `the terrain's heightmap texture is not available here` (or `is not created`, or has another format): the details are placed from the terrain's GPU heightmap.
- `<prefab>`: no mesh or material on the prefab's root or `<prefab>`: its mesh has no triangles: an instanced detail mesh is drawn from its prefab root's MeshFilter and MeshRenderer, as Unity draws it. `<prefab>`: cannot be drawn: the prefab's own reason is shown on the first terrain that uses it.
- Unity-style grass is drawn through adapted copies of URP's own terrain detail shaders. Its reasons: `layer N (...): <shader> could not be adapted (...)` (is being adapted while the copy is made and compile-checked: about 15 to 30 s after a first setup, or after a URP or Drop-In Terrain Renderer update that remakes the grass copies), `layer N (...): the pipeline's detail shader X is not URP's own` (a pipeline asset with its own terrain detail shaders) or `the pipeline has no terrain detail shader for it` (fix: restore URP's own terrain detail shaders, or turn GPU instancing on for mesh layers, which do not use them), `layer N (...): a mesh in <mode> mode` (fix: use Vertex Lit or Grass mode, or turn GPU instancing on), `layer N's texture could not be read`, `the detail atlas could not be packed`. A grass copy that failed on one graphics API counts as failed on every machine until URP is updated, a Drop-In Terrain Renderer update changes its grass copies, or someone presses Patch Materials.

- the terrain is lit by realtime GI alone (Enlighten): ..., or its lightmap N is missing (bake the lighting again, or load the scene that owns it): for Unity-style grass on a lightmapped terrain.
- the details round the cameras need more than the N MB of Detail Memory (raise Detail Memory on the component), or ... more than the lists can address (lower the terrains' Detail Density or Detail Distance). While a camera's details are being placed (placing N patches to go), after a jump or a scene load, Unity draws them for it.

Project-wide (every terrain).

- no shader library on the component (press Patch Materials or Set Up, which makes one) (only when a material needs a generated copy: a component whose trees and details all use Drop-In Leaves, Bark and Grass needs no library, as the demo's), or no culling compute shader (DropInTerrainRendererCull.compute) on the component (or the detail compute shader). Add Component gives a new component the project's library (made when there is none), as Set Up and Patch Materials do; a library lost later (its folder deleted, the reference cleared) is not made again on its own. Fix: press Patch Materials, which makes and assigns them.
- compute shaders and indirect instanced draws are needed, vertex shaders cannot read five buffers here, or DropInTerrainRendererCull.compute did not compile here (details: compute shaders and indirect instanced draws reading five buffers are needed, DropInTerrainRendererDetails.compute did not compile here): this GPU or graphics API lacks what Drop-In Terrain Renderer needs.
- too many trees: more than 4,194,240 trees across all taken-over terrains together. Fix: remove the component from some terrains; Unity draws those.
- too many trees × LODs for the lists' float start: the trees of every taken-over terrain, each counted once per LOD of its kind, reach 8,388,608 (about a million trees with 8 LODs, or 2.1 million with 4). Impostors are dropped first: when the count with them crosses the limit, the trees keep only their mesh LODs. Fix: use fewer LODs, or remove the component from some terrains (Unity draws those).
- nothing to draw, no terrain's trees can be drawn yet (or no detail kind can be drawn yet): no terrain can be drawn yet, and each terrain's own reason says why. While the patch of materials met for the first time is still to come (after Add Component, new prototypes or a setup), the Console waits for it, and says this only if it stays.
- the GPU records' sizes are wrong: an internal error. Report it with the Console log.
- An error during a frame shows as its exception and message; Unity draws, and Drop-In Terrain Renderer builds again a second later.

Cameras. Some cameras get Unity's own trees and details while the others keep Drop-In Terrain Renderer's: see Cameras and Reflections under Compatibility. When it happens for a lighting reason, the Inspector says "Some cameras: ...".

Impostor and Convert problems never hand a terrain back: they show in the Impostors and Look rows ("Drawn as yours: ..."), and the tree keeps drawing.

A material's reason is kept with the project (in the shader library), and the material is tried again when its shader changes (another shader chosen, or the shader or one of its sub graphs imported again), after a Unity upgrade or a Drop-In Terrain Renderer update that changes its instancing code, or when you press Patch Materials in the Drop-In Terrain Renderer component. In the Console, material reasons start with "left to Unity" (after Patch Materials, with the terrain's name first), and project-wide reasons end with "Unity draws the trees" (or details). A terrain's own reasons show in its Inspector, in the stats overlay's Terrains list (see Stats overlays), and through the component's `Problem` and `DetailProblem` properties.

Did it help? Measuring your scene

The benchmark measures Unity against Drop-In Terrain Renderer on your own view, from your camera: in Play mode in the editor, or in a development build. Unity draws, then Drop-In Terrain Renderer, then Drop-In Terrain Renderer, then Unity (ABBA). Unity's side is Drop-In Terrain Renderer paused: every terrain goes back to Unity with nothing rebuilt, and what Drop-In Terrain Renderer still costs while paused is measured and shown. When a terrain has both its trees and its details drawn by Drop-In Terrain Renderer, a trees-only and a details-only block follow, if they fit in the run's 40 s. A report then says whether it helped, why, and what to do next.

In the editor. Open the Benchmark section of a Drop-In Terrain Renderer Inspector and press Enter Play Mode and Measure (in Play mode: Measure), or choose `Tools > Havoila Studio > Drop-In Terrain Renderer > Benchmark > Measure In Play Mode`. From Edit mode it enters Play mode, starts 2 s after Play mode begins, measures for about 15 to 40 s, leaves Play mode and opens the report window (see Reading the report). While it runs, the section shows one progress line with the seconds left and Cancel, and the Inspector shows only the status boxes and this section. The section's settings:

- Camera: "Camera.main when the run starts" (the default: the enabled camera tagged MainCamera, found even when your game spawns it), or one of the open scenes' enabled screen cameras. With no camera to measure, the report says `No Game camera renders to the screen: Measure needs one`, with why (for the default: `(no enabled camera is tagged MainCamera)`): tag your camera MainCamera, or pick it here.
- Freeze game time while measuring (on): `Time.timeScale` is 0 while measuring, so camera scripts and animations stop and both sides draw the same view.
- Longer run (off): Unity, ours, ours, Unity twice over (ABBAABBA), for a noisy view.

The Game view's size is the resolution measured: set it to the resolution you ship (the Game view's aspect menu, or Play Maximized) before you press Measure. Keep the Game view visible and the editor focused. While the editor is in the background with Run In Background off (Unity's default: Unity then stops Play mode or slows it to about 10 fps), or paused, the run pauses: that time is not counted, and the progress line asks you to keep the editor focused. With Run In Background on, Unity keeps drawing in the background and the run keeps measuring. Either way the frames where the focus changes are left out. A slow frame is measured as it is: a view where Unity's side runs at 3 fps is not taken for a lost focus. Hiding the Scene view helps: frames in which it renders are left out. Entering Play mode cancels impostor bakes still running (impostors bake in Edit mode only), so let them finish first: the Last setup box's Measure in Play Mode is unavailable until they finish, but this button and the menu are not. The run first waits for compiles and Deep Profiling to end (60 s at most), then for Drop-In Terrain Renderer to be ready (10 s at most; then it measures anyway and says so). The section keeps the last result for the editor session, in one line (`2.2x faster · 23 -> 51 fps; (not like for like)` after the speed-up, and see `Careful` or the blocks a cancelled run finished at the end, when they apply) with Open report, which opens the report window again; the header of the section shows the same verdict. One Console line repeats the verdict and the first reason. Editor numbers include the editor's own work on both sides; a build shows your game alone. Use the editor Measure to compare settings quickly. For numbers to keep, or to compare with Measured results below, measure in a development build: the editor's GPU readings were less steady on our test machine.

In a build. Add the Benchmark component (`Tools > Havoila Studio > Drop-In Terrain Renderer > Benchmark > Add Benchmark` to the Open Scene, or the Benchmark section's In a build fold), turn on Development Build, build as usual and run. Start Delay seconds after the scene starts (5 by default) it measures from Camera.main (or its Target Camera), game time frozen, and puts the report:

- on screen: a small panel says `Drop-In Terrain Renderer benchmark: measuring starts in N s`, then `Measuring Drop-In Terrain Renderer: game time frozen, about N s left` with Cancel (it hides itself in the measured frames), then shows the report's first screen with Show all, Copy, the file's path and Close (in XR, on the desktop window only);
- in the Player log, as one `[Drop-In Terrain Renderer] Benchmark` block;
- in a file: `Application.persistentDataPath/Drop-In Terrain Renderer Benchmark/<date>_<time>_<scene>.txt`, and the same report as a page beside it (`<date>_<time>_<scene>.html`: the report window's layout, self-contained, opens in any browser, offline). Write File off writes neither.

Without the component, start a development build with `-dropInTerrainRendererBenchmark`: it measures once a scene with Drop-In Terrain Renderer is loaded (a menu scene first is fine). Options: `-dropInTerrainRendererLonger` (Unity, ours, ours, Unity twice over, for a noisy view), `-dropInTerrainRendererDelay <seconds>`, `-dropInTerrainRendererOut <folder>` (where

the file goes), `-dropInTerrainRendererQuit` (quits once the report is written). A scene that has the component with Run On Start uses it, with those options. The component also measures when you enter Play mode in the editor. Keep the build's window in front: with Run In Background off (Unity's default), the game stops while it is in the background, and the run pauses with it; with it on, the run keeps measuring. Either way the frames where the focus changes are left out.

Release builds do not measure: the component logs one line and turns itself off (no per-frame cost), and the command line is ignored. Every build that contains an enabled Benchmark with Run On Start gets a warning in its build log: untick Run On Start or remove the component before shipping or QA builds. No project setting is needed or changed: development builds read the GPU timers without the Frame Timing Stats setting.

Settings: Target Camera (none: Camera.main when the run starts), Run On Start (on; off: only `Run()` measures), Start Delay (5 s), Freeze Time (on), Longer Run (off), Show Results (the panel; on), Write File (on), Quit When Done (off; ignored in the editor). There is no hotkey: the panel uses ImGui and needs no input package.

Scripting (`HavoilaStudio.DropIn.TerrainRenderer.DropInTerrainRendererBenchmark`):

- `Run()` : measures now; false when it cannot (a release build, Play mode not running, no Drop-In Terrain Renderer, another measurement going), with the reason in the log.
- `Cancel()` ; `IsRunning` ; `LastReport` (the report's text; its last line is the data line, see Reading the report); the `Finished` event (the report's text, when a run ends).
- `DropInTerrainRendererBenchmark.RunNow(camera)` : measures now from a temporary object, removed when its panel is closed or its scene unloads.

What a run changes, for the run only, restored however it ends (finished, Cancel, the end of Play mode, an error):

`Time.timeScale` (0 with Freeze game time), `Application.targetFrameRate` (-1, no cap) and, in builds only, `QualitySettings.vSyncCount` (0). The editor never changes VSync: frames held at the display's refresh rate are flagged instead, and the report says the frame is capped.

Reading the report

The report window opens when Measure ends, and from the Benchmark section's Open report. It leads with the speed-up: `2.2x faster`, `13% slower` or `No measurable difference`, with the fps and the frame time beside it (`23 -> 51 fps · frame 43.4 -> 19.5 ms`). Then:

- one context line: `Like for like` or `Not like for like` (see Compared as set below), the Detail Mode when grass is compared, the terrains Unity draws on both sides (`Tile_4's trees drawn by Unity on both sides`: the speed-up covers the rest), Unity's own noise (how far apart Unity's own blocks read: a smaller difference is not called), and the bottleneck on each side (`bottleneck: main thread -> GPU`); under it, where it ran;
- three cards, Unity against Drop-In Terrain Renderer: the frame time, the CPU main thread (URP's render and culling on the main thread, with Drop-In Terrain Renderer's update: the text report's render main thread) and the GPU. The bars are to scale (the larger of the two fills its card), a gain in green, a cost in amber, each with its change (`33x less`, `7% more`). A number that could not be read says `not readable here`, with why, never 0; the other side's number then shows without a bar (one value has nothing to scale against), as do two GPU numbers read from different sources;
- one Why line (More reasons folds the others; when the frame did not move, Why says what that means: the CPU main thread's change and what sets the frame, or the cap that holds both sides) and the Next steps, numbered in the order to try them;
- folded sections: Trees and grass apart (the trees-only and details-only blocks, whenever they ran), Compared as set, Per terrain and kind, and How it was measured (every total with its source, the notes, the run's header and blocks).

Its buttons: Copy (the whole text report, below), Open as HTML (the same report as a self-contained page, written to a temporary file and opened in your browser: no network, nothing to install, easy to send), Save as... (an `.html` page, or the text when the name ends in `.txt`) and Measure again. It follows the editor's light or dark theme; the page follows the browser's.

The text report is the same in Copy, a saved `.txt`, and a build's panel, Player log and file. Its first screen, at most 14 lines, answers "did it help, why, and what now":

1. **The verdict.** Every comparison has one format: `0.45x Unity's frame time (41.3 -> 18.6 ms; 24 -> 54 fps)`. The ratio is Drop-In Terrain Renderer's time over Unity's (under 1x, Drop-In Terrain Renderer takes less), then Unity's value -> Drop-In Terrain Renderer's. The verdict is one of:

- `Faster: ...x Unity's frame time (...)`;
- `Slower: ...x Unity's frame time (...)`. Most of it: `<part>`, ...; `<what to change>` (All of it: `<part>`, the only part compared; ... when only trees or only grass were compared);
- `No measurable difference in frame time (...; within ±N%: Unity against itself)`, with the render main thread's change when it moved, and what that means (see A small or negative result);
- `Not measured: <reason>`, when the run could not start or ended before both sides were measured; its Next line says what to do;
- `Frame time not comparable (...)`, with the reason;
- `Headset-paced (XR): ...` (see XR, under Not measured below).

When the two sides do not draw the same work, a tag follows: `Drop-In Terrain Renderer draws more than Unity here` (see Compared as set). (or less, more and less, or with other shaders).

2. **Careful:** at most two lines that qualify the verdict: a cancelled run (its numbers come from the blocks it finished), blocks still over 20% flagged, a camera that moved while measuring (the numbers mix views), Drop-In Terrain Renderer not ready after 10 s, a setting your code changed during the run, a Profiler recording.
3. **Why:** then **Also:** one to three reasons, ordered by what sets the frame. For example: Unity culling LOD Group trees one by one on the main thread, Unity drawing detail meshes in patches of up to 1,023, the GPU's change and its cause, a part measured in its own block, the batches. In a slower result, what was still gained reads `Also (still gained):`.
4. **Next:** one to three steps, each an action on a named setting (see A small or negative result).
5. **The context:** where it ran (Editor Play mode (includes the editor's own work) or Development build), the resolution, the camera, the blocks and frames, and `Unity against itself ±N%`, the run's noise. Then `Bottleneck: ...` for each side: the GPU, the main thread, the render thread, VSync or a frame cap, or not determined.

When a difference counts. Each side is measured in two blocks (four with Longer run); after it settles, a block measures about 1.5 s of frames (30 to 120). A side's number is the mean of its block medians, and the spread between its blocks is the noise. A difference is called only when it is larger than that noise and than 2%; otherwise the report says "no measurable difference", with the threshold and what set it. Frames in which something else happened are left out: the Scene view rendered, a garbage collection, a compile, the camera moved (1 cm or 0.1 degree), the focus was lost or came back (Unity's focus events; in the editor, its Pause button too), the camera did not render. A slow frame is not left out, however long. A block more than 20% flagged is measured again once.

The rest of the text report (in the report window's folded sections, Copy and Save as...):

- **Totals,** `Unity draws (Drop-In Terrain Renderer paused) -> Drop-In Terrain Renderer`, each number with its source:
 - Frame time: median, p95 and hitches (frames over twice their side's median), from `Time.unscaledDeltaTime`.
 - Render main thread: URP's whole render on the main thread (every camera's culling, Unity's tree culling and LOD picking, Drop-In Terrain Renderer's render hooks), plus Drop-In Terrain Renderer's Update. It holds no wait for the GPU: read it first for the CPU.
 - Main-thread total: in builds, `FrameTimingManager's` main-thread time; in the editor, the player and editor loops less the main thread's waits (it may hold the GPU's wait there). Shown only when it passes its check, else `not separable from the GPU wait here`.
 - Render thread: `FrameTimingManager's`.
 - GPU: URP's camera sampler plus Drop-In Terrain Renderer's four GPU samplers, with `FrameTimingManager's` GPU time beside it when it reads at least as much. With no GPU timer: `GPU-bound frame time` where a side is known GPU-bound, else `not readable here`.
 - Batches, Draw Calls and SetPass Calls: Unity's counters (Draw Calls includes Drop-In Terrain Renderer's draws); Drop-In Terrain Renderer's own draws; Unity's foliage batches, by difference.
 - Drop-In Terrain Renderer CPU, while paused -> drawing: what it costs on Unity's side, and its own cost.
 - The bottleneck on each side, and each module block (trees only, details only) against Unity's side.

A number that cannot be read says why, never 0.

- **Compared as set:** each setting under which Drop-In Terrain Renderer draws other work than Unity: more work (impostors past a LOD Group's last LOD, Extended grass past Unity's Detail Distance, a LOD Bias above 1), less work (a LOD Bias below 1, Max Distance, Extended's thinning inside Unity's distance), or other shaders (converted kinds, grass textures drawn as Drop-In Grass). `Like for like` when there is none.
- **Per terrain**, then **per tree kind** (its class; in view by LOD, impostors and shadow casters; its draw commands; Unity's estimated renderers, "at least": each shadow cascade is not counted, and in shadows the trees whose bounds meet one of URP's cascade culling planes, Unity's rule for its own renderers, not Drop-In Terrain Renderer's shorter list (where the trees' reach pass does not run: its casters as before it); why it gains) and **per detail kind** (how it is drawn; in view, in shadows and far; its draw commands), at most 12 kinds each (they are counted, not timed: see Not measured).
- **Notes and How it was measured:** the Unity version, graphics API, threading mode, GPU, CPU, quality level; one line with the package version (`Drop-In Terrain Renderer 1.0.0`), the render path URP runs (Forward, Forward+, Deferred, Deferred+), the GPU's vendor, vendor and device IDs and API version string (on Vulkan and OpenGL it holds the driver's version, in brackets; on Direct3D Unity gives scripts none: the Player or Editor log names it), the window mode and size, the XR plug-in's display (such as `OpenXR Display`: the plug-in, not the headset model), the LOD Bias, the Maximum LOD Level when not 0, the quality level's terrain overrides and the quality scales when set, each terrain's Tree LOD Bias Multiplier when not 1, the camera's vertical field of view and SpeedTree 8 or 9 trees; the URP asset, VSync, the camera with its vertical field of view, each source's validity on each side, and each block.

For scripts: the data line. The text report ends, after a blank line, with one line for scripts and spreadsheets; the `.html` page carries the same line in `<pre id="dtr-benchmark-data" hidden>`:

```
dtr-benchmark format=1 version=1.0.0 unity=6000.3.25f1 build=development outcome=finished end=na api=Direct3D12
res=2560x1440 frame=0.06862 frame_unity=255.0412 frame_ours=17.5021 frame_called=yes gpu=0.95616 ... speedtree=none
driver=na
```

After the `dtr-benchmark` prefix come `key=value` pairs separated by spaces. Every value is percent-encoded (RFC 3986: decode it with any URL decoder). Numbers use a `.` in every language and are not rounded for display (up to 4 decimals, ratios 5). `na` marks a value not read, never 0. Ratios are Drop-In Terrain Renderer's time over Unity's, as in the report; times are in ms.

Keys: `format`, `version` (the package), `unity`, `build` (editor, development, release), `outcome` (finished, cancelled, refused), `end` (why it was cancelled or refused), `api` (the graphics API), `res` (the camera's pixels), `frame`, `frame_unity`, `frame_ours`, `frame_called` (past the noise), `gpu`, `gpu_unity`, `gpu_ours`, `gpu_called`, `gpu_source` (cameras, frametiming, gpuframe, mixed), `main`, `main_unity`, `main_ours` (the render main thread), `noise` (Unity against itself, %), `like` (like for like), `path`, `path_asked` (as URP runs it and as the renderer asks it: Forward, ForwardPlus, Deferred, DeferredPlus), `vendor`, `device` (the GPU's PCI IDs, hex), `gfx` (the API version string), `window` (ExclusiveFullScreen, FullScreenWindow, MaximizedWindow, Windowed, editor), `screen` (the window's size, which can differ from `res`), `hz` (the display's refresh rate), `xr` (none, or the XR plug-in's display, such as OpenXR Display: not the headset model), `xr_camera` (yes: the measured camera renders to a headset), `fov` (vertical, degrees; ortho), `lod_bias`, `tree_lod` (one value, or each terrain's in the Per terrain order, comma-separated), `speedtree` (none, 8, 9, 8,9), `driver` (the driver's version where the API version string carries it: Vulkan, OpenGL; else na), `overrides` (the quality level's Terrain Setting Overrides: none, or Unity's names joined by commas, such as DetailDensity,DetailDistance), `o_tree_distance`, `o_detail_distance`, `o_detail_density` (the level's value where it overrides it, else na), `scale_distance`, `scale_density`, `scale_lod` (the quality scales, 1 unless set), `max_lod` (the Maximum LOD Level), `legacy_detail` (yes, no: Unity's legacy detail distribution).

`format` changes only if a key's meaning or unit does. New keys may be added at the end, so read keys by name and skip the ones you do not know.

A small or negative result

Drop-In Terrain Renderer takes the trees' culling and LOD picking off the main thread and draws in fewer calls. Where the main thread did not set the frame, the frame rate does not change: the report says so, and why. The verdicts and their next steps:

- **The GPU sets the frame.** No measurable difference in frame time (...); render main thread ...x (...): the GPU sets the frame on both sides, so the saving is free main-thread time for your game, not frame rate. The GPU draws the same work on both sides; the main thread's saving is time your game can use. When the GPU sets the frame

on Drop-In Terrain Renderer's side, a next step names one GPU lever: a LOD Bias below 1 draws simpler LODs nearer and costs less GPU when the trees cost more than Unity's in their own block (when Drop-In Terrain Renderer draws more trees than Unity there, the step names that instead: the impostors, or a LOD Bias above 1), else Detail Mode: Extended (the component's Grass and Details section) thins far grass and usually costs less GPU when the grass is in Match Unity. Both change the look, so compare. Without either (a terrain with trees only runs no module blocks, for example), the report names no GPU lever.

- **A frame cap.** ...: VSync or a frame cap holds both sides at N fps, so the saving cannot show in the frame rate; lift the cap to see it (meanwhile it is free main-thread time for your game). Fix: in the editor, turn off the Game view's VSync or the cap; in a build, turn off VSync in the GPU's control panel, or run windowed.
- **Grass in Match Unity: a gain in draw calls and main-thread time.** In Match Unity, Drop-In Terrain Renderer draws Unity's own placement, distance and meshes, each kind in one indirect draw instead of patches of up to 1,023 draw calls. Its gain is draw calls and main-thread time, not GPU time: grass in Match Unity draws Unity's own work, so its gain is main-thread time and draw calls, not GPU time. Its GPU time is close to Unity's: on our test machine (RTX 5080, D3D12 development players at 2560x1440), 0.95x on the MeadowLite bench (14.8 -> 14.1 ms, see Measured results) and 0.95-0.97x in the demo's densest grass views. Before version 1's lean fragment path for patched grass graphs, the same bench view read 1.09-1.12x (14.87 -> 16.24-16.67 ms), and equal or lower at 1280x720, in D3D11 and in Vulkan. Other scenes may differ: measure yours. Grass textures and non-instanced meshes are drawn as Unity draws them, for coverage and fewer draw calls, at about Unity's GPU time (billboard grass, and Grass-mode meshes with GPU instancing off, cost a little more: see Known issues). When the GPU now sets the frame, the next step is Detail Mode: Extended (the component's Grass and Details section) thins far grass and usually costs less GPU; it changes the look, so compare. (The Inspector's Extend Reach button sets Detail Mode to Extended.)
- **Plain trees.** Unity instances trees without a LOD Group but still culls them one by one on the main thread, and stops them at Billboard Start (or Tree Distance, if nearer); Drop-In Terrain Renderer culls them on the GPU, so the main thread gains here too. (20,000 plain trees on our test machine: 14.3 -> 0.73 ms on the main thread; see Measured results.) With an impostor (Look: Convert, then Bake), a plain tree's mesh hands over to a flat card at the impostor's From height (3% of the screen's height by default), and the card draws out to Billboard Start (or Tree Distance), where Unity draws the full mesh. The card replaces work rather than adding it. When plain trees outnumber LOD Group trees in view, that reason goes on: With an impostor their far part is drawn as cards up to there, where Unity draws full meshes: less GPU time.; otherwise, when the counts were read, a reason line counts the cards as a GPU saving (N far trees without a LOD Group drawn as cards (2 triangles each) where Unity draws their full meshes: less GPU work). Either way they are never counted as Drop-In Terrain Renderer drawing more. When plain kinds without an impostor lead Unity's work, Next: Impostors (Look: Convert, then Bake) draw the far part of these trees as cards up to Billboard Start (or Tree Distance, if nearer): less GPU time.
- **Few trees in view.** With fewer than 100 LOD Group trees in view: Few trees in view: measure from a view of the forest.
- **Impostors and Extended draw more than Unity.** Impostors past a LOD Group's last LOD draw far trees where Unity draws none (a tree without a LOD Group: see Plain trees), and Extended draws grass past Unity's Detail Distance. The verdict then carries Drop-In Terrain Renderer draws more than Unity here, and Compared as set says what: the frame compares more work against less, a farther forest and farther grass against Unity's. To compare like for like, turn the Impostors switch off (on every terrain that plants those kinds), set Detail Mode to Match Unity, and set LOD Bias to 1. Extended's next step says how the look changes: Extended changes the look (<terrain>: thinner from X m, farther to Y m): compare it in the Game view; Detail Mode: Match Unity measures like for like.
- **Slower.** Slower: ...x Unity's frame time (...). Most of it: <part>, ...; <lever>. The part comes from the trees-only and details-only blocks when they ran; otherwise it is estimated from what Drop-In Terrain Renderer draws more. When only trees or only grass were compared, it reads All of it: <part>, the only part compared; <lever>. The levers, in the report's words: for impostors, raise Impostor Min Height, or turn the Impostors switch off to compare like for like; set LOD Bias to 1; for Extended, set Detail Mode to Match Unity ... to compare like for like; when spot and point lights cast realtime shadows into the grass, lower Punctual Shadow Radius; otherwise turn off Draw Trees on <terrain>'s Drop-In Terrain Renderer component (or Draw Details: the component's switch, not the terrain's own settings), which gives that part back to Unity. Handing a part back comes last, after the steps that keep it: with grass in Match Unity where the GPU sets the frame, Detail Mode: Extended ... first, the Punctual Shadow Radius step next, then If Extended's look is not wanted, or grass and details still cost more: turn off Draw Details on <terrain>'s Drop-In Terrain Renderer component ...
- **A terrain Unity draws.** Next: <terrain>'s trees are drawn by Unity: <reason>. <fix>. Both sides draw it Unity's way, so it adds nothing to the result. Fix the reason (see "Why does Unity draw my trees?"), then measure again.

- **The run's own conditions.** Measure again with the Scene view hidden. (the editor's bottleneck could not be determined, or the Scene view rendered in measured frames); Turn on Freeze game time (or stop the camera's script), then measure again. (the camera moved); Measure again once Drop-In Terrain Renderer is ready (it was not after 10 s: see Careful).

Measured results

All measured on our test machine, with an NVIDIA RTX 5080 GPU and an AMD Ryzen 9 7950X3D CPU; AMD and Intel GPUs have not been tested. Each row gives its bench, resolution, GPU and date. Your scene, settings and hardware differ: measure your own scene with the benchmark. The bench names are test scenes; none ships with the package.

A worked example. The test scene Bench is a 2 km terrain with 150,000 SpeedTree 8 trees with LOD Groups and a grass layer. The benchmark ran from the command line (`-dropInTerrainRendererBenchmark`) in a Windows development player, D3D12, 2560x1440 windowed, on 2026-10-16:

- The verdict: Faster: 0.45x Unity's frame time (41.3 -> 18.6 ms; 24 -> 54 fps).
- Why: render main thread 0.03x (34.1 -> 0.90 ms). Unity culls the 150,000 trees and picks their LODs one by one on the main thread; Drop-In Terrain Renderer does it on the GPU and draws them in 112 indirect draws. Batches 0.03x (13,414 -> 400).
- Main-thread total 0.03x (41.4 -> 1.2 ms): Unity's side was held by its main thread.
- GPU 1.07x (17.0 -> 18.2 ms), the camera samplers and FrameTimingManager agreeing: the GPU did a little more work on Drop-In Terrain Renderer's side, and it set that side's frame (18.2 ms of an 18.6 ms frame), as the first screen's next step says.
- The trees-only block read 0.45x, and the details-only block no measurable difference: the gain is the trees'.
- Unity against itself $\pm 0.2\%$; no frame flagged.

Read it as: the frame time fell by more than half because Unity's main thread no longer handles the trees one by one. The GPU now sets the frame, so a further gain would come from the GPU's side (see "The GPU sets the frame", under A small or negative result), not from the CPU.

Where the speedup is. Unity against Drop-In Terrain Renderer, one benchmark run per row. Ratios are in the report's format, and each pair is labelled main thread or GPU.

Measured 2026-10-16 (the two MeadowLite rows 2026-10-18, after the grass fragment fix) in development players on our test machine: an NVIDIA RTX 5080 with an AMD Ryzen 9 7950X3D, Direct3D 12, 2560x1440 windowed, Unity 6.3. AMD and Intel GPUs have not been tested: measure your own scene with the benchmark.

What	Scene	Frame time	Main thread (render)	GPU time
LOD Group trees (150,000 SpeedTree 8 trees and a grass layer)	Bench, from a ridge	0.45x Unity's (43.4 -> 19.5 ms)	0.03x (35.8 -> 1.0 ms)	1.07x (17.9 -> 19.2 ms)
Grass, Match Unity (detail meshes)	MeadowLite, at ground level	0.96x Unity's (15.1 -> 14.4 ms)	0.78x (0.84 -> 0.66 ms)	0.95x (14.8 -> 14.1 ms)
Grass, Extended (not like for like: grass out to 400 m against Unity's 160 m, thinned past 100 m)	MeadowLite, at ground level	0.80x Unity's (15.1 -> 12.1 ms)	0.79x (0.85 -> 0.67 ms)	0.80x (14.9 -> 11.9 ms)
Unity-style grass billboards (grass textures)	TextureGrass, billboards	1.18x Unity's (2.6 -> 3.1 ms): slower	no measurable difference (0.66 -> 0.66 ms)	1.18x (2.4 -> 2.8 ms)
LOD Group trees at a lookout (115,742 trees on 12 terrains; not like for like: converted kinds, 5,172 far trees as impostors where Unity draws none)	WildForest, the Lookout	0.15x Unity's (32.6 -> 4.9 ms)	0.14x (20.2 -> 2.8 ms)	no measurable difference (3.3 -> 3.3 ms)

What	Scene	Frame time	Main thread (render)	GPU time
Trees without LOD Groups (20,000 plain trees over 2 km, Billboard Start at the 2 km Tree Distance; impostors on, as shipped)	PlainBench	0.53x Unity's (17.2 -> 9.1 ms)	0.05x (14.3 -> 0.73 ms)	0.83x (10.8 -> 8.9 ms)

Read it as: trees are where Drop-In Terrain Renderer pays off, by taking Unity's per-tree culling off the main thread (with or without LOD Groups: Unity instances plain trees but still culls them one by one). Grass in Match Unity saves main-thread time and draw calls (batches 1,047 -> 76) and draws the same meadow at about Unity's GPU time (0.95x at 2560x1440 in Direct3D 12); Extended saves more GPU, with a different look. Unity-style grass billboards cost a little more GPU than Unity's own draw.

What each feature costs. The benchmark compares Unity against Drop-In Terrain Renderer only. To find what a feature costs in your scene, Measure twice, once with it on and once with it off, and compare the two ratios. Each run measures Unity again, so a change in your scene between the runs shows on both sides and mostly cancels out. The rows below were measured that way: the cost is the ratio with the feature on minus the ratio with it off, also given in milliseconds (main thread and GPU, labelled).

Feature	Scene	GPU time cost	Main thread (render) cost
Impostors on (against off; 244 far trees drawn as impostors where Unity draws none)	ImpostorBench, eye level	no measurable cost (1.06x -> 1.08x Unity's)	no measurable cost
Drop-In Leaves' light through the leaves (transmission)	LeavesBench, eye level	no measurable cost (1.03x -> 1.05x Unity's)	no measurable cost (-0.01 ms)
Extended against Match Unity (not like for like)	MeadowLite, at ground level	-2.3 ms (0.95x -> 0.80x Unity's)	no measurable cost (0.79x -> 0.79x)
Impostors at the Lookout (5,172 far trees; the kinds converted in both runs)	WildForest, the Lookout	no measurable cost (1.01x -> 1.01x Unity's)	+0.18 ms (0.13x -> 0.14x)
Impostors on plain trees (5,122 far trees)	PlainBench	-2.2 ms, a saving (1.03x -> 0.83x Unity's)	no measurable cost

Not measured

- **XR.** The benchmark has not been run in XR. In version 1 it labels an XR run: frame time is "set by the headset", the bottleneck is not judged, and the verdict leads with the render main thread, Drop-In Terrain Renderer's own CPU and the draw calls. URP takes no camera sample in XR, so the GPU number can only come from FrameTimingManager, and a GPU-bound frame time never stands in for it. The build panel and the stats overlays draw on the desktop window only.
- **Other platforms.** The benchmark and the stats overlays have run on Windows only. Consoles, Metal (macOS), Linux and WebGPU have never been built (see Graphics APIs, under Compatibility). Mobile (Android, iOS, Quest) is not supported in version 1 (see Platforms, under Compatibility).
- **Graphics APIs.** The benchmark ran in D3D12 development players. Its timing sources were also checked in D3D11 and Vulkan development players: on Vulkan, FrameTimingManager reads no GPU time, and the camera samplers do.
- **Unity versions.** The benchmark and the stats overlays ran on Unity 6.3. On Unity 6.6 they compile, but have not run.
- **The editor against a build.** Editor and build numbers have not been compared on the same view. The numbers here come from development players; editor numbers hold the editor's own work on both sides.
- **Per kind and per terrain.** A tree kind, a detail kind or a terrain cannot be timed on its own in version 1: Drop-In Terrain Renderer's draws sit inside URP's passes. The report gives their counts, draws and Unity's estimated renderers, and times trees and details apart when the module blocks fit.
- **Several cameras.** Drop-In Terrain Renderer's GPU number holds the culling of every camera it drew for, Unity's side one camera's; the report says so when it happens. Other screen cameras' cost is in both sides (listed under Notes). Split screen and camera stacks are not measured apart.

- **Some features' costs:** motion vectors With Wind, the motion-vector passes of your materials under As Unity beyond the one SpeedTree 8 forest measured (see Motion vectors), the Drop-In Sky and Drop-In Water hooks, and In Reflections' GPU time (measure them in your scene as described under What each feature costs).

Stats overlays

The stats overlays show, for one camera, what Drop-In Terrain Renderer draws and why Unity draws the rest: what is in view, Drop-In Terrain Renderer's draws, its CPU time and its GPU memory. They do not compare frame times: the benchmark does.

In the Scene view. Press Stats in the Drop-In Terrain Renderer Inspector, or choose `Tools > Havoila Studio > Drop-In Terrain Renderer > Show Stats Overlay`. It is a Scene view overlay (Drop-In Terrain Renderer in the Scene view's Overlays menu), hidden until you show it. It holds:

- Camera: This Scene view, or Game camera (Camera.main), each at its last render;
- the summary (below);
- Terrains: up to 10 terrains, those Unity draws something of first, each with its reason and fix;
- Kinds: up to 10 lines, the tree kinds then the detail kinds, each ranked by what is in view, with why each gains;
- where the numbers come from;
- Show in the Game view (Play mode): the runtime overlay in the Game view while in Play mode, for this editor session. Nothing is saved in the scene. The Game view cannot show overlays in Edit mode.

In Play mode and in builds. Add the Stats Overlay component to any object (`Add Component > Havoila Studio > Drop-In Terrain Renderer > Stats Overlay`). F8 shows and hides it; it reads the key from ImGui's events, so it works in a project that uses the Input System alone. It leads with what is on screen, in large type: the trees in the camera's view (impostors counted in) and the grass and details (the far field counted in), then in Play mode Unity's batches and SetPass calls, the number that jumps when Unity's terrain draws instead (the headline, below). Show details unfolds the rest: the camera, the summary below, Detail's list and where the numbers come from. Settings:

- Toggle Key: F8 by default. None: no key, and your code calls `Toggle()` or sets `Visible`.
- Corner: top right by default.
- Target Camera: none follows the last Game camera Drop-In Terrain Renderer drew for (Camera.main before the first).
- Detail: what Show details lists besides the summary: Compact (nothing more), Terrains (each terrain) or Kinds (the kinds).
- Show Details: off by default (the headline alone). The overlay's Show details button switches it in Play mode (a left click; in a bottom corner the button sits under the details, so it stays where you clicked); a game whose cursor is locked sets `showDetails` from code.
- Visible At Start: on.
- Refresh Interval: the seconds between reads of the numbers (0.5 by default, never under 0.25). The text is rebuilt only when a number changed.
- Development Builds Only: on by default. A release build then turns the overlay off and logs `[Drop-In Terrain Renderer] Stats Overlay: off in this release build (Development Builds Only is on)` once. Off: it works in release builds too.

What they cost. An overlay listens only while it is shown (the Scene view's: shown and not collapsed). Then the visible counts are read back from the GPU's culling for the shown camera only, at most four times a second and without stalling, and in Play mode Unity's own counters are recorded. Hidden, it reads nothing. While the benchmark measures, no count is read back, and the runtime overlay hides.

The headline (the runtime overlay):

- `6,912 trees`, and under it `5,678 of them impostors (far cards)`: the trees drawn in the camera's view (a tree inside a cross-fade band counted once), its impostors counted in.
- `165,345 grass and details`, and under it `45,000 of them in the far field` (Extended): those past the far field's capacity are in the summary.

- In amber under a row, when Unity draws for a lasting reason: `Unity draws the trees for this view: <reason>` when this camera is left to Unity, else `Unity draws <terrain>'s trees: <reason>` (; N more: set Detail to Terrains to list them, or listed under Show details). Not for a passing state: a pause (the demo's Space, a measurement), details still being placed after a jump, a module not built yet.
- `Trees: Unity draws them (not counted here)` when Unity draws them for this camera: the counts come from Drop-In Terrain Renderer's own culling. `Trees: none to draw` when no terrain has trees Unity would draw. A count not read yet says why (`Trees: waiting for the GPU's count` , and the reasons below), never 0. With no enabled component, the one line below and no Show details.
- In the demo, its planted counts (a line of the demo's).
- `Unity's counters: N batches · K SetPass calls` (Play mode): Unity's render counters for the last frame, every camera.

The summary, line by line (the Scene view overlay, and the runtime overlay's Show details):

Line	What it says
<code>Trees: Drop-In Terrain Renderer, N of M terrains (Unity draws <terrain>'s trees: <reason>)</code>	Of the terrains that have trees to draw, how many Drop-In Terrain Renderer draws. A terrain whose Trees switch is off, or with no enabled Drop-In Terrain Renderer component, counts as Unity's. The first one Unity draws is named with its reason; N more: see the Terrains fold counts the others (in the runtime overlay: see Terrains below, or set Detail to Terrains to list them). When it draws none: <code>Unity draws them on the terrain (<terrain>: <reason>)</code> (or both terrains , all N terrains): fix the reason (see "Why does Unity draw my trees?"). <code>none to draw</code> when no terrain has trees Unity would draw; <code>no render of this view seen yet</code> until the camera renders. For a camera Drop-In Terrain Renderer leaves to Unity (a reflection camera, a camera nested two deep, the per-object light limit), it reads <code>Unity draws them for this view: <reason></code> .
<code>in view: N trees, M impostors (far cards); S shadow casters</code>	The trees drawn in this camera's view (a tree inside a cross-fade band counted once), its impostors apart, and the trees drawn into its shadows. From the GPU's culling, read back 2-4 frames late.
<code>Grass and details: ... and its in view: line</code>	The same for details: instances near the camera, in the far field (Extended), those past the far field's capacity (not drawn), and shadow casters.
<code>Drop-In Terrain Renderer draws for this view: N draw calls submitted (a shadow draw counted once): trees A, grass and details B</code>	Drop-In Terrain Renderer's draw calls for this camera, its view and its shadows. Unity's Draw Calls count a shadow draw once per cascade and light, so the two do not compare one to one.
<code>Drop-In Terrain Renderer CPU: N ms a frame, all cameras together (main thread; replaces Unity's tree and grass work: Measure compares)</code>	Drop-In Terrain Renderer's own main-thread time for every camera, averaged over 0.5 s (a Stopwatch round its hooks). It replaces Unity's tree and grass work, which the overlay cannot see: the benchmark compares the two (in a build, the label names the Benchmark component).
<code>Streaming: caught up</code>	Or the detail patches still to place and the Extended pages still to read, after a jump or a scene load.
<code>GPU memory: N MB (trees ..., grass and details ..., impostors ..., bend field ...)</code>	What Drop-In Terrain Renderer holds on the GPU, from its buffer and texture sizes. The bend field shows while grass interaction holds memory.
<code>Unity's counters, last frame, every camera: N batches, M draw calls (ours included; ...), K SetPass calls</code>	Play mode only: Unity's own render counters for the last frame. <code>starting</code> until a frame is recorded, <code>not readable in this build</code> where they cannot be read.

A count not read yet says why, never 0: `waiting for the GPU's count` , `waiting for this camera's next render` , `not available on this platform (no asynchronous GPU read-back)` or `paused while the benchmark measures` . With no enabled component, the summary is one line: `No enabled Drop-In Terrain Renderer component in the open scenes: Unity draws`

every terrain. The runtime overlay's details start with its camera (Camera "<name>"); F8 hides this sits beside its Show details button (with no enabled component, on a line of its own).

Terrains lines are the benchmark report's per-terrain lines: each module Drop-In Terrain Renderer's (its trees, its details in Match Unity or Extended) or Unity's with the reason, then `Fix: ... (Fix (trees): ... Fix (details): ...` when they differ). A switch you turned off is your choice: its line says so, with no fix. A terrain with no enabled component reads `<terrain>: no enabled Drop-In Terrain Renderer component: Unity draws its trees and details. Fix: Add Drop-In Terrain Renderer to it (Tools > Havoila Studio > Drop-In Terrain Renderer > Set Up Selected Terrains).`

Kinds lines give each tree kind's class (LOD Group or no LOD Group, impostor last, SpeedTree wind, converted), what of it is in view (or how many are planted, before the first count), its draw commands for a view and why it gains: `Culled and LOD-picked on the GPU, where Unity does it tree by tree on the main thread, Culled on the GPU, where Unity instances these but culls them one by one on the main thread (and stops them at Billboard Start or Tree Distance), far trees drawn as cards where Unity draws none` (a kind without a LOD Group: `their far part drawn as cards where Unity draws full meshes`). Each detail kind gives how it is drawn (instanced mesh; grass texture or grass billboard, as Unity draws it; non-instanced mesh, as Unity draws it; grass texture as Drop-In Grass), what is in view, its draw commands and why: `One indirect draw, where Unity draws patches of up to 1,023 (a draw call each)`. A kind Unity draws reads `Unity draws it: <reason>`. Same cost as without Drop-In Terrain Renderer; nothing is lost.

The last line says where the numbers come from, and where frame and GPU time are compared: Measure in the editor, the Benchmark component in a development build, and in a release build `development builds and the editor only`.

Grass interaction

Characters, creatures and vehicles press Drop-In Grass as they move through it, and the grass rises again on its own.

Setup. Add a Foliage Interactor to your character (`Add Component > Havoila Studio > Drop-In Terrain Renderer > Foliage Interactor`). It fits itself to a CharacterController, CapsuleCollider or SphereCollider on the same object; Fit to Collider does it again later. Grass Interaction is on by default in the terrain's Drop-In Terrain Renderer component, and it costs nothing while nothing presses the grass.

Settings (the Drop-In Terrain Renderer component's Grass Interaction section; its header switch turns it off). One bend field serves every terrain, with the values of the first enabled component that has it on.

- Area: the square round the camera where grass bends: 32, 64 (default) or 128 m.
- Resolution: texels per side of that square: 128, 256 (default, 1 MB of GPU memory) or 512 (4 MB).
- Recovery: how long pressed grass takes to rise again (s, the time constant; back at rest after five times this).
- Max Angle: how far a blade bends at most, at its tip (70 degrees by default).
- Preview in Edit Mode: interactors bend the grass in Edit mode too, round the Scene view's camera. Off: Play mode only.

The section's status says what the field does now (how many interactors press it, round which camera) or why nothing bends. Each Foliage Interactor's Inspector says whether it is inside the bent area, outside it, or past the limit.

Foliage Interactor. Shape (a sphere, or a capsule along the object's up axis), Center, Radius (times the object's largest scale), Height (capsule), Strength (0-1), and Push From Motion: how far the push leans the way it moves (a walker sweeps the blades forward). The 64 interactors nearest the camera press the grass; others are ignored.

Material. Drop-In Grass's Bend (0-1, Grass interaction group) scales how far a plant bends: lower it for stiff flowers, 0 never bends.

Scripting (`HavoilaStudio.DropIn.TerrainRenderer.FoliageInteraction`):

- `AddImpulse(position, radius, strength)`: a sphere pressing the grass for one frame (a landing, a blast, a helicopter's downwash); call it every frame for a lasting push.
- `SetEnabled(false)` / `SetEnabled(true)`: switches grass interaction off and on for everything (`Enabled` reads it).
- `SetFocus(transform)`: the Transform the square follows (null: Camera.main, the head in XR; `Focus` reads it).
- `Hold()` / `Release()`: keeps the field as it is (holding FoliageWind holds it too).

- `Clear()` : forgets every bend (teleports, camera cuts).
- `IsActive` : whether grass reads a bend field this frame.

What bends. Grass drawn by Drop-In Terrain Renderer:

- Drop-In Grass: detail mesh layers drawn with a Drop-In Grass material (converted or authored), and grass texture layers converted to Drop-In Grass. Their shadows bend with them.
- Unity-style grass textures (grass cards and billboards). They bend over keeping their length, with URP's own wave on top. They have no Bend setting, so they bend fully: convert a layer to Drop-In Grass to make it stiffer.

The far field of Extended mode bends inside the square. The Inspector's Detail layers rows say for each layer whether it bends.

What cannot bend.

- Anything Unity draws: the cameras Drop-In Terrain Renderer leaves to Unity (preview cameras, reflection cameras it does not cover, cameras nested two deep or more, the per-object light-limit hand-back), terrains not taken over, layers it cannot draw, and lightmapped billboards in Deferred.
- Detail mesh layers drawn with your own shader (Keep mine): convert them to Drop-In Grass (the Look section) to bend them.
- Mesh layers with GPU instancing off (drawn as Unity draws them).
- Trees, including shrubs painted as trees.
- Plain renderers using Drop-In Grass outside the terrain.
- Plants hidden under water.
- Grass outside the square (the bend fades out at its edge), and everything in Edit mode unless Preview in Edit Mode is on.
- A second view in split screen: there is one square, round the focus.

Motion vectors (TAA, STP, motion blur). Bent grass draws its motion. While interaction is active, Drop-In Terrain Renderer's runtime copies of Drop-In Grass materials whose Bend is above 0 turn their motion-vector pass on and draw their motion (wind and bend, wind-still grass included, out to 60 m from the camera; with the component's Motion Vectors set to As Unity, Drop-In Grass draws per-object motion only then); once the field is at rest they go back to the material's own setting. Your materials are never changed. Unity-style grass textures draw their bend's motion too: with the component's Motion Vectors set to With Wind they draw their motion (wave and bend) everywhere; set to As Unity, they draw it only while interaction is active and only inside the square (the camera's motion stands in outside it). While the field is held or the game paused, the bend draws no motion.

Support

Questions and bug reports: support@havoila.com, or our Discord server, discord.gg/2ahVw3N9SX. Havoila Studio: havoila.com.

Please include your Unity and URP versions, the target platform and any Console errors. The Copy button of the Last setup box, or of the report window after a Measure, gives the details.