UNITY FOR GAMES E-BOOK

UNITY FOR TECHNICAL ARTISTS

KEY TOOLSETS AND WORKFLOWS

2021 LTS EDITION

Unity

Contents

INTRODUCTION

An urban environment created in the High Definition Render Pipeline

This guide offers an overview of the toolsets and systems in Unity, or supported by it, that technical artists (TA) can use to help their teams meet the visual requirements in their game production.

Technical artists often function as the bridge between artists and programmers on a team because they can provide solutions for both technical and artistic requirements.

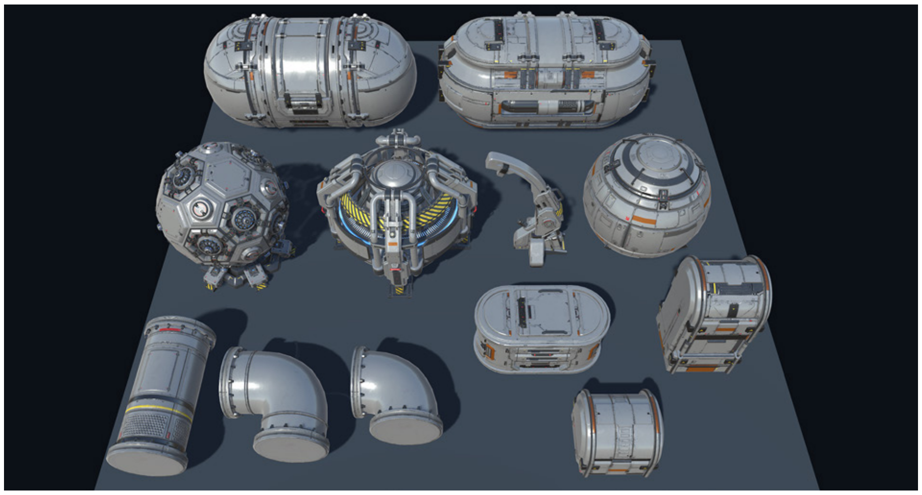
They have a broad understanding of what's possible to achieve on various target platforms with the Digital Content Creation tools and game engine that they're using. This enables them to inform the art director and artists of any limitations and opportunities surrounding the target hardware.

Many TAs work directly on their team's most complex artistic needs, from character rigging to writing shaders, or proposing new workflows and creation tools to accelerate their processes. By working closely with the game engine technology, and observing the final rendering result in real-time, TAs play a key role in ensuring that the visual quality of a game or other application meets the standard set by their team.

Our TAs have years of experience working in large game productions, with deep knowledge of lighting, animation, cinematics, visual effects, and shader and graphics programming. This guide was assembled with input from some of Unity's most experienced TAs from R&D, plus the demo team based in Stockholm – the creators of *Adam*, *Book of the Dead*, *The Heretic*, and *Enemies*.

Read this guide for an introduction to the authoring tools and APIs available in Unity 2021 LTS and newer versions, and learn how to set up an efficient content creation pipeline that supports everyone on your team to deliver their best result.

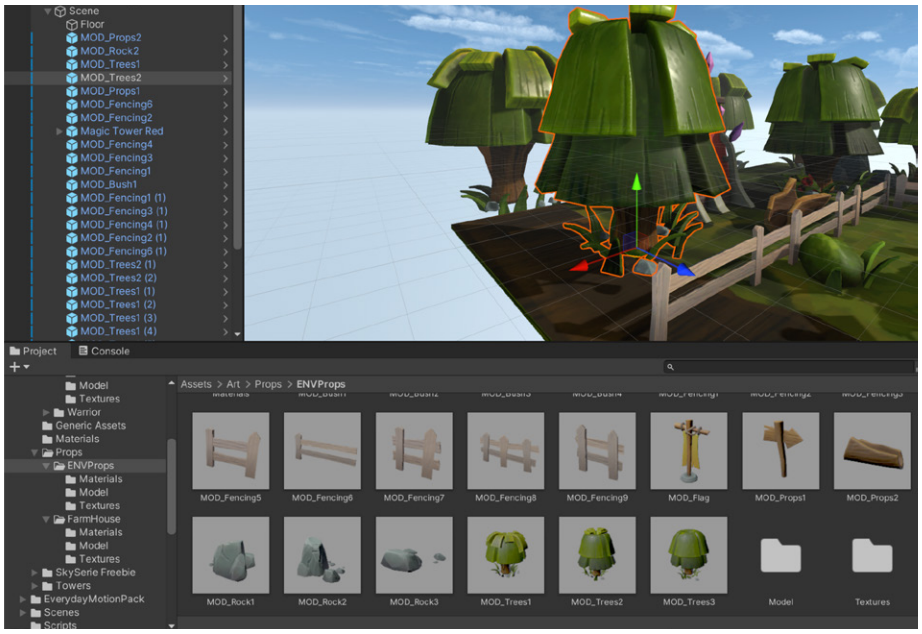
PREFABS



An example of Prefabs that are building blocks for an environment

Use Unity's Prefab system to create, configure, and store a **GameObject** as a reusable Asset, complete with all its components, property values, and child GameObjects. The **Prefab Asset** acts as a template from which you can create new Prefab instances in Scene view. These assets can then be shared between scenes, or even other projects, without having to be configured again.

Prefabs are useful for objects that will be used many times, such as platforms or collectible items in a game. To create a Prefab, drag an object from the **Hierarchy** window into the **Project** window.



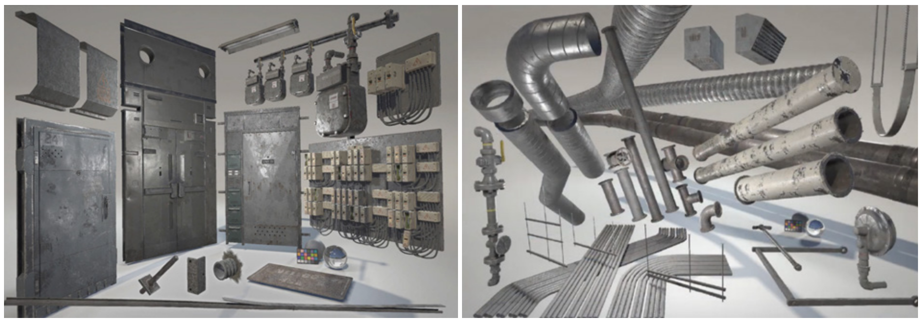
The Prefab MOD_Trees2 shown in the Project window, in a two-column view (bottom) and one-column view (left)

Like all assets in Unity, Prefabs are editable. You can edit a Prefab on a per-object basis, where a single instance of a Prefab is changed for an individual purpose, or changes can be applied to all instances of the Prefab. This makes it more efficient to fix object errors, swap out art, or make other stylistic changes.

Nested Prefabs allow you to parent Prefabs to one another in order to create a larger Prefab; for instance, a building that's composed of smaller Prefabs, such as those for the rooms and furniture. This makes it easier to split the development of your assets over a team of multiple artists and developers, who can all work on different parts of the content simultaneously.

A **Prefab Variant** allows you to derive a Prefab from other Prefabs, much like inheritance in object-oriented programming. To affect the Variant, you must override certain parts without impacting the original. You can also remove all modifications and revert to the base Prefab at any time.

Alternatively, to change all of your Prefab Variants at once, you can apply changes directly onto the base Prefab itself.



Larger Nested Prefabs can consist of smaller Prefabs. For example, the Prefab panel on the left includes many power transformer Prefabs, whereas the one on the right showcases Variants of a pipe Prefab with different materials.

More resources

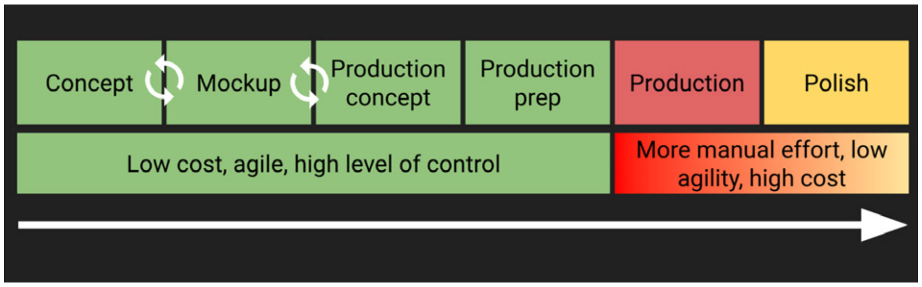
Introduction to Nested Prefabs
Improved workflows for editing Prefabs

WORKING WITH ASSETS

Production pipeline

A foundational system of any game engine is a robust **Asset Pipeline**, which determines the state of assets in a project and updates dependent systems accordingly. The Asset Pipeline can also be extended by a user-defined production pipeline.

A solid Asset Pipeline for your project reduces the risk of artists adding assets to the project that don't comply with technical requirements (e.g., 3D models that have a higher tri-count than what was budgeted or planned for). A good Asset Pipeline helps your team follow standards for naming conventions, file paths, Prefabs, and other settings, thereby avoiding issues such as asset duplication, broken dependencies, or mistakes when bundling assets. This section introduces the tools available in Unity that you can use to create a safe and efficient Asset Pipeline.


Concept
Mockup
Production concept
Production prep
Production
Polish
Low cost, agile, high level of control
More manual effort, low agility, high cost

A detailed preproduction plan allows your studio to parallelize the work when multiple artists are active in the project, minimizing unused assets or other issues that can impact productivity and performance.

A non-destructive Asset Pipeline

As part of the preproduction planning, your team should agree on aspects of the asset workflow, such as the list of required assets to be created, the budget for them based on your project's technical requirements, and the development cost.

One way you can group required level elements is the following:

— **Large elements:** These elements can't be authored with unique textures but must use tiling or modular elements, such as walls, the ground, cliffs, roofs of buildings, and so on.

— **Medium-sized props:** These are props that have unique texture sheets, like barrels, crates, rocks, and doors.

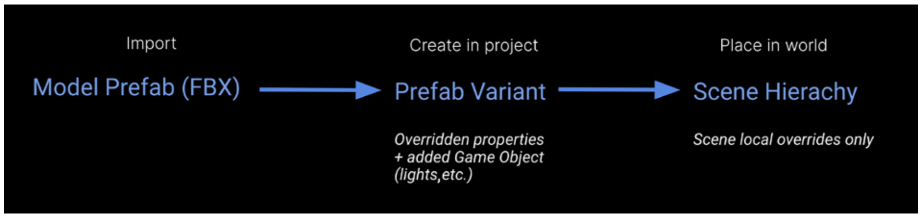
— **Highly repetitive, small elements:** Think of items used to ground objects or convey scale, including pebbles, leaves, screws, and bolts.

Budget the time, polycount, and texture resolution of each asset, and prioritize the tasks accordingly. Then create non-essential assets and reduce the number of essential assets to the bare minimum.

Another critical aspect of the asset import workflow is the creation of Prefabs from 3D models. Directly converting the FBX model into a Prefab during production isn't recommended as it will break the Prefab if changes are made to the FBX file, such as hierarchy changes in the geometry. This will require you to replace all the instances of it in the scene with an updated version. The solution to this problem lies in Prefab Variants.

Prefab Variants enable the scene-independent production of your art assets, making them ideal to use while prototyping. They are recommended for collaborative iteration while building the game world. They offer the benefit of regular Prefabs, like making variations of the original model, but also act as a "weak" reference to the FBX model Prefab. This allows you to conveniently add, split, or remove model elements without having to recreate the Prefab. Production artists can make changes to the FBX models, and the Prefab Variant will update accordingly. Prefab Variants also allow artists to see art changes in context without altering the project itself.

At the same time, consider using Nested Prefabs to facilitate collaboration on the scene.

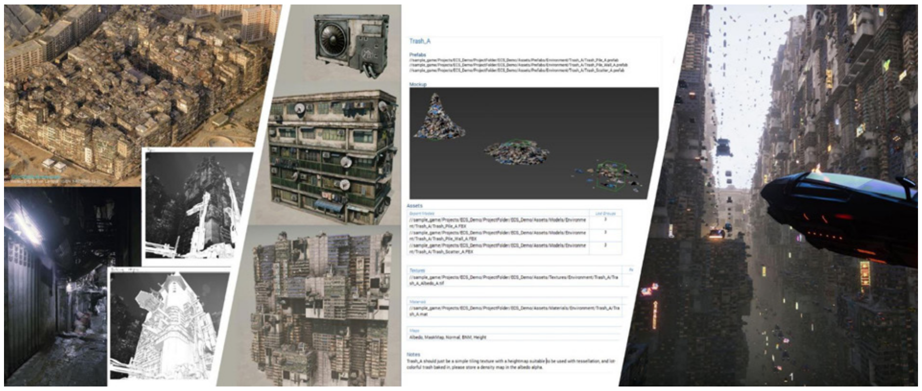


A recommended Prefab Variant workflow for productions with many collaborators

Automating Asset settings with the AssetPostProcessor

In a production involving thousands of assets, avoid relying on manually configuring the specifications of each asset from the Inspector.

To automate the process of validating Asset files, use the **AssetPostProcessor API**. This helps you hook into the import pipeline and run scripts prior to, or after, importing assets; just add assets for the scripts to make the necessary changes. You can change the settings from a single script that will automatically reapply the changes to the assets in the project.



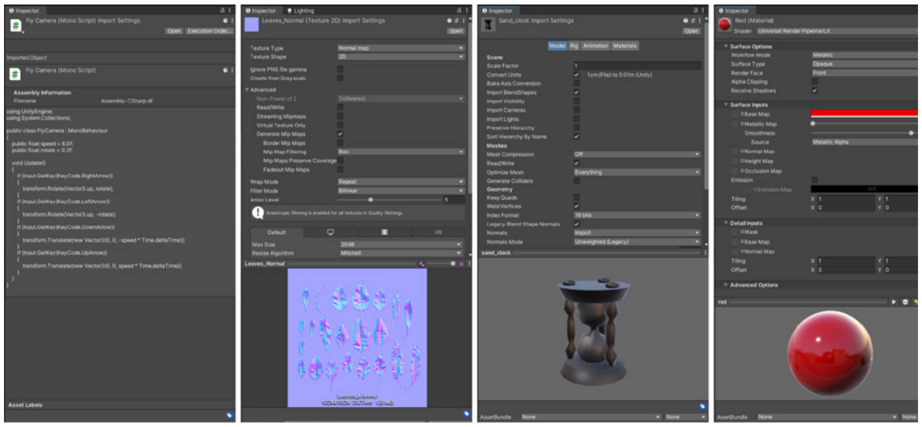
From left to right: Examples of the mood board, concept art, Prefab Assets, directory and naming standards, and final visuals for the *Megacity* demo by Unity

Importing assets

You can bring assets created in DCC tools directly into the Assets folder of your project where Unity can read them. Unity detects when you save new changes to the file and reimports files as needed.

When you modify the settings of an asset in Unity, whether from a script or from the Inspector, the original source file remains unchanged. Instead, Unity creates a game-ready representation of your asset internally that matches your chosen Import settings. Always move or rename your assets from within the Unity Editor, as it does the same for the corresponding meta files.

The Import settings look different depending on the type of asset at hand.



```
Assembly Information
Filename                Assembly-CSharp.dll
```

```
using UnityEngine;
using System.Collections;

public class FlyCamera : MonoBehaviour
{
  public float speed = 6.0f;
  public float rotate = 0.2f;

  void Update()
  {
    if (Input.GetKey(KeyCode.RightArrow))
    {
      transform.Rotate(Vector3.up, rotate);
    }
    if (Input.GetKey(KeyCode.LeftArrow))
    {
      transform.Rotate(Vector3.up, -rotate);
    }
    if (Input.GetKey(KeyCode.DownArrow))
    {
      transform.Translate(new Vector3(0, 0, -speed * Time.deltaTime));
    }
    if (Input.GetKey(KeyCode.UpArrow))
    {
      transform.Translate(new Vector3(0, 0, speed * Time.deltaTime));
    }
  }
}
```

```
Texture Type                  Normal map
Texture Shape                 2D
Ignore PNG file gamma
Create from Grayscale
Advanced
  Non-Power of 2              ToNearest
  Read/Write
  Streaming Mipmaps
  Virtual Texture Only
  Generate Mip Maps           ✓
    Border Mip Maps
    Mip Map Filtering         Box
    Mip Maps Preserve Coverage
    Fadeout Mip Maps
Wrap Mode                     Repeat
Filter Mode                   Bilinear
Aniso Level                   5
Anisotropic filtering is enabled for all textures in Quality Settings.
Max Size                      2048
Resize Algorithm              Mitchell
```

```
Scene
Scale Factor                  1
Convert Units                 ✓  1cm (File) to 0.01m (Unity)
Bake Axis Conversion
Import BlendShapes            ✓
Import Visibility
Import Cameras
Import Lights
Preserve Hierarchy
Sort Hierarchy By Name        ✓
Meshes
Mesh Compression              Off
Read/Write                    ✓
Optimize Mesh                 Everything
Generate Colliders
Geometry
Keep Quads
Weld Vertices                 ✓
Index Format                  16 bits
Legacy Blend Shape Normals    ✓
Normals                       Import
Normals Mode                  Unweighted (Legacy)
```

```
Shader                        Universal Render Pipeline/Lit
Surface Options
  Workflow Mode               Metallic
  Surface Type                Opaque
  Render Face                 Front
  Alpha Clipping
  Receive Shadows             ✓
Surface Inputs
  Base Map
  Metallic Map
    Smoothness
    Source                    Metallic Alpha
  Normal Map
  Height Map
  Occlusion Map
  Emission
    Emission Map              HDR
  Tiling                      X 1    Y 1
  Offset                      X 0    Y 0
Detail Inputs
  Mask
  Base Map
  Normal Map
  Tiling                      X 1    Y 1
  Offset                      X 0    Y 0
Advanced Options
```

Previsualization and settings for different asset types in the Inspector: C# script, texture, FBX asset, and material

Unity supports many asset types along with their built-in importers, including the most common 3D models, textures, and audio files. See the complete list in the documentation.

You can also create your own importer in C# with **Scripted Importers**. Scripted Importers enable you to work with files that are not natively supported in Unity. Examples can be found in the Package Manager, including **Alembic** and **PSD Importers**.

The Asset Database

For most types of assets, Unity needs to convert the data from the asset's source file to a format that can be used in a game. It stores these converted files and their associated data in the Asset Database.

The conversion process is required because most file formats are optimized to save storage space. In a game, however, the asset data needs to be in a format that is ready for hardware or immediate use, such as the CPU, graphics, or audio hardware. For example, when Unity imports a .PNG image file as a texture, it does not use the original .PNG formatted data at runtime. Instead, when you import the texture, Unity creates a new representation of the image in a different format, stored in the **Library folder > Project window**. The **Texture** class in Unity uses this imported version, and Unity uploads it to the GPU for real-time display.

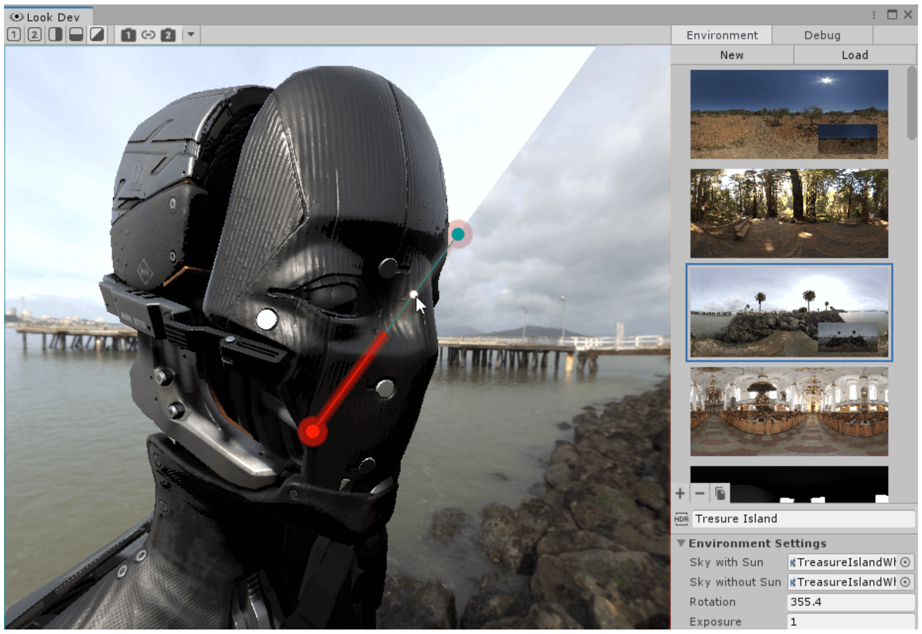
If you subsequently modify the source file of an asset you've already imported (or change any of its dependencies), Unity reimports the file and updates the imported version of the data. See Refreshing the Asset Database for more information on this process.

The Asset Database also provides an **AssetDatabase API** that you can use to access assets and control or customize the import process.

Visualizing 3D models in context

During a typical production, there are many times when a new visual asset will need to be reviewed holistically and validated by different stakeholders, such as the art director. Testing how materials and assets behave beforehand is the only way to ensure that they will work well in the scene.

Unity's Look Dev solution, available under **Window > Render Pipeline > Look Dev**, is an image-based lighting tool that comes with the **High Definition Render Pipeline (HDRP)**. It contains a viewer for you to check and compare assets to verify how they work in various lighting conditions. You must either create a new **Environment Library** or load one the first time you use Look Dev – but make sure it's a **High Dynamic Range Image (HDRI)** to better simulate real-world lighting.



Tresure Island

Environment Settings

Sky with Sun	TreasureIslandWI
Sky without Sun	TreasureIslandWI
Rotation	355.4
Exposure	1

An HDRI image in Look Dev

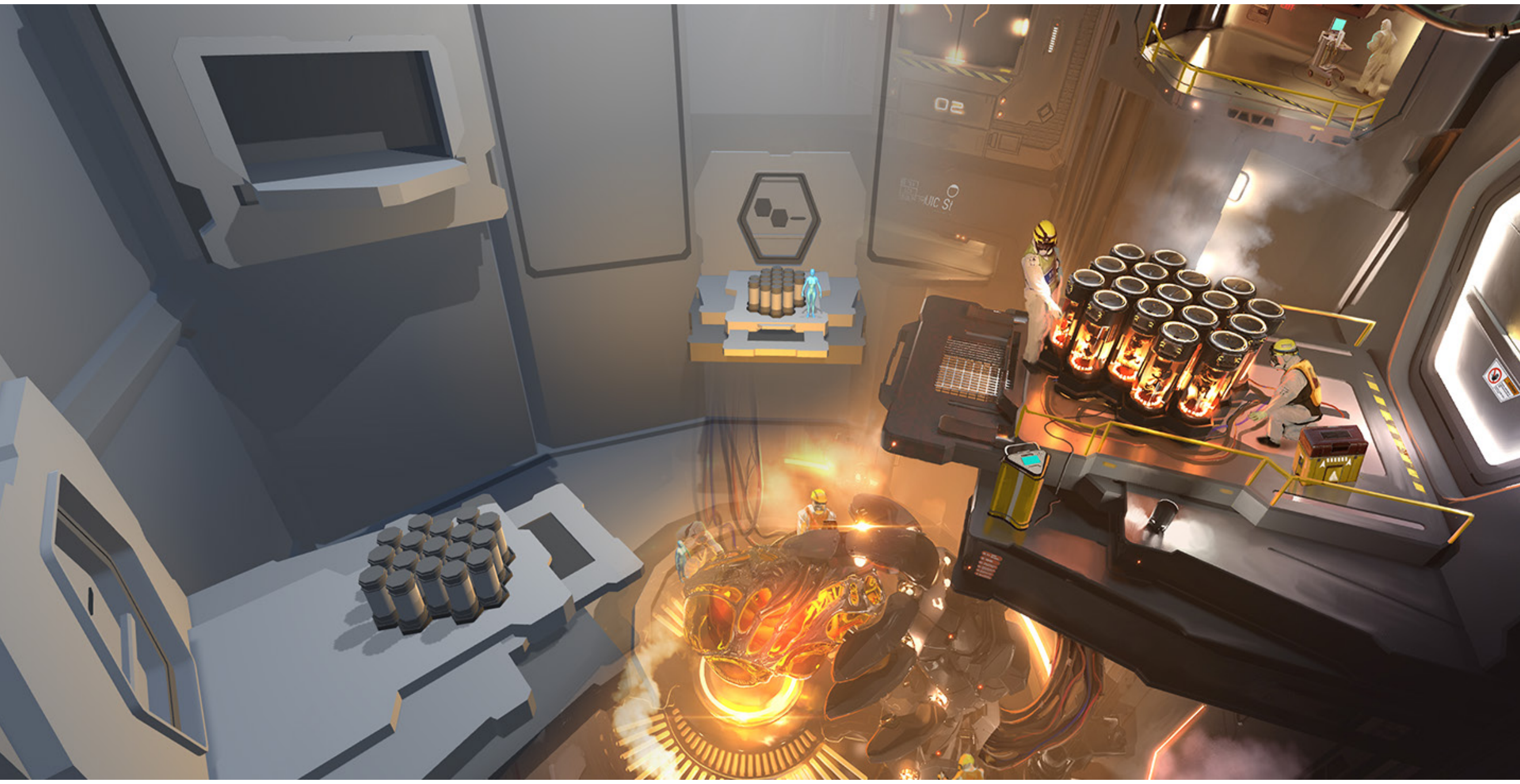
FBX Exporter

The FBX Exporter package is one of the products developed collaboratively by Unity and Autodesk. It enables a smooth roundtrip workflow for transferring 3D models between Unity and Autodesk® Maya®, Autodesk® Maya LT™, or Autodesk® 3ds Max®.

With the FBX Exporter, you can export Unity scenes to FBX files and import them into Maya, Maya LT, or 3ds Max using an artist-friendly interface. More specifically, you're able to export Unity-ready FBX geometry and animation, and safely merge your changes back into those assets to continue working in Unity.

For example, you can block out a level in Unity or prototype an animation or cutscene that artists and animators can then load and iterate on in their preferred DCC tool. The Unity FBX Exporter supports hierarchies, lights, meshes, materials, textures, and camera parameters.

However, if you're working on a performance-heavy or graphically demanding game, such as an open world, you'll want to ensure early on that the game can handle this complexity. The prototyping assets can be created with the final specs of the production assets. For instance, a mockup asset can have the same tri-count as the final asset budget or the same shader setup.



The grey-boxed assets on the left side were created with Unity ProBuilder, whereas the final assets on the right side were completed in DCC software and imported back into Unity.

Working with other DCC tools

Unity uses the FBX file format internally, so it's recommended that you also use it wherever possible in your production, and avoid proprietary model file formats. One exception is if you work with DCC software such as Blender. In this case, you can save your 3D assets in Unity as .Blend files inside the project's Asset folder. However, this requires that everyone on your team installs and uses the same version of DCC software.

Unity will import the mesh with all nodes in their saved position, rotation, and scale. Pivot points and names will also be imported – along with vertices, polygons, triangles, UVs, normals, bones, skinned meshes, and animations. As you iterate on assets in other supported DCC software, Unity will update the corresponding GameObject and reflect your changes in the Unity Editor every time you save the file.

Beyond roundtripping with Blender and other DCC software, it's recommended that you use FBX files to maximize compatibility with Unity features like **Cloud Build**. If you prototype or grey-box scenes in Unity, your artists can then work with those assets in their DCC tool once the assets are exported as FBX. Go to Unity's **Project settings** to adjust your default **Export** to **FBX options**.

More resources

Artist workflows with Maya and Unity
Roundtrip easily between Unity and Autodesk
Tips for working more effectively with the Asset Database

RENDER PIPELINES

The flexible graphics features in Unity provide you with a refined level of control for crafting sharply optimized visuals across a range of platforms, from mobile to desktop and high-end consoles. A render pipeline performs a series of operations to take the contents of a scene and display them onscreen. At a high level, these operations are culling, rendering, and post-processing.

Unity provides three render pipelines for various purposes: the older **Built-in Render Pipeline**, and two more recent **Scriptable Render Pipelines (SRPs)** – the **Universal Render Pipeline (URP)**, which will become the default rendering pipeline, and the **High Definition Render Pipeline (HDRP)**. You can also create custom SRPs. Your choice of render pipeline depends on your target platforms and the level of visual fidelity and customization you seek.

Rendering paths

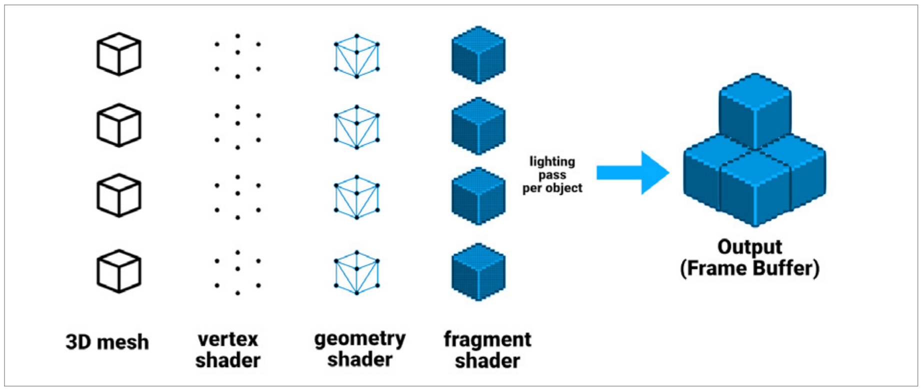
A rendering path is a series of operations used to render lighting and shading. Different rendering paths have different capabilities and performance characteristics. The following rendering paths are available for Unity's render pipelines:

Forward rendering

Forward rendering is the default rendering path in the Built-in Render Pipeline and URP. It is a general-purpose rendering path that limits the number of real-time lights that can affect an object's pixels.

Each light affecting the object has a cost. When they are above a certain number, the lights will affect the object with an approximation (in URP all lights are calculated per-pixel and the limit can be changed). If your project does not use a high number of real-time lights (mobile or VR projects), then this rendering path might be a good choice because fullscreen passes are not done for every light in the scene, which is costly for tile-based GPUs.

However, the tile/cluster architecture in HDRP still allows for a number of lights to affect each object in Forward rendering, and generally, offers feature parity between the two rendering paths. It also provides a slightly more accurate rendering quality than the Deferred G-buffer rendering, making it suitable for HDRP projects where high visual fidelity is prioritized over performance. Read more in HDRP's rendering path documentation.

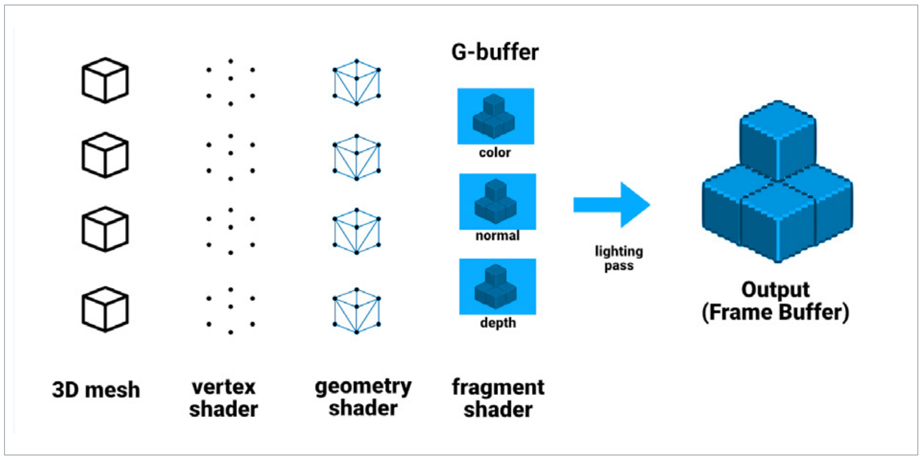


Forward rendering path

Deferred Shading rendering

Deferred Shading rendering requires a higher level of support for GPU features, typically those found on desktops, consoles, and high-end mobile devices (but has limitations on some rendering features). If your project demands several real-time lights and a high level of lighting fidelity while targeting high-end mobile, PC, or console platforms, this rendering path might be the right choice when used with the Built-in Render Pipeline or URP.

Deferred Shading rendering is the default path in HDRP, since HDRP projects are expected to have a high number of real-time lights and fullscreen effects. In HDRP, Deferred Shading maintains a proper balance between performance and quality.



Deferred Shading rendering path

Universal Render Pipeline

The URP provides the most optimized graphics performance on a broad range of platforms including mobile, PC, console, WebGL, and VR. This is possible due to some trade-offs around lighting and shading. Certain restrictions are applied, and features that aren't supported on low-end devices are disabled. This way, developers can spend less time worrying about how to optimize their work, and focus more on developing projects that will reach a larger audience.

Another benefit is that URP is scalable. Advanced developers can customize it to control the rendering for specific purposes in their project. As noted earlier, URP will replace the Built-in Render Pipeline as the default render pipeline in Unity.

A Unity demo in URP

URP renders in a light loop by default, making it possible to perform Forward rendering in a single pass. In comparison, Forward rendering in the Built-in Render Pipeline performs an additional pass per-pixel light within range. This means that using URP results in fewer draw calls.

URP is ultimately the best solution for 2D game development in Unity because it provides a 2D Renderer, 2D Lights and lighting effects, and the 2D Pixel Perfect Camera for implementing a pixelated visual style in your project.

To get started with URP, check the documentation and go to the **Unity Hub** in Unity 2021 LTS or newer.

High Definition Render Pipeline

The HDRP is a high-fidelity SRP used for targeting modern platforms that are compatible with compute shaders. Its features enable artists and developers to achieve high-definition visuals at speed. Use HDRP for AAA-quality games, automotive demos, architectural applications, virtual production, film, or any other content that requires high-fidelity graphics.

I Am Fish by Bossa Studios Ltd., created with HDRP, is available on Steam and Xbox.

HDRP follows three main principles: physically based rendering (PBR), unified and coherent lighting, and rendering path independence. It builds upon all of these with new rendering processes and shaders, and vast improvements to lighting within the scene.

The pipeline uses a hybrid Deferred / Forward – Tile / Cluster renderer, so that the lighting scales better than it would with any other render pipeline. There is feature parity in HDRP between Deferred and Forward rendering with only small variations in the quality and accuracy of effects. HDRP can be customized using **Custom Pass** and **Custom Post Processing** effects.

HDRP is designed to be accessible to smaller development teams seeking the highest quality visual fidelity for 3D PC and console graphics. It is not suitable for mobile platforms.

LEGO Builder's Journey by Light Brick Studios was made with HDRP and ray tracing with NVIDIA DLSS for its PC and console versions.

Read the documentation and watch this presentation for an in-depth introduction to HDRP.

Creating a custom render pipeline

For more control over rendering in your project, you can create a custom SRP or customized versions of the URP and HDRP.

To create a custom SRP, you'll need to use the **SRP Core API** provided by Unity. It contains reusable code to help you make your own render pipeline, including boilerplate code for working with platform-specific graphic APIs, utility functions for common rendering operations, plus the shader library that URP and HDRP both use.

A custom SRP requires writing C# scripts to configure and schedule rendering commands. The **ScriptableRenderContext** class acts as an interface between the C# scripts and Unity's low-level graphics code.

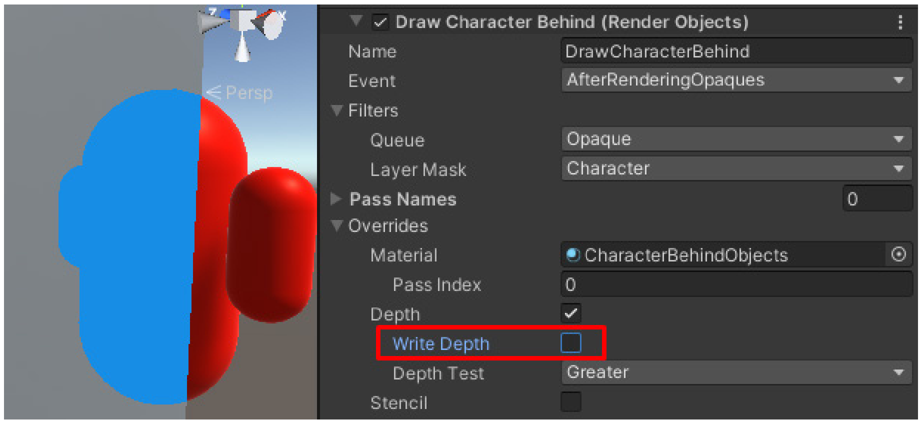
SRP rendering operates through delayed execution. First use ScriptableRenderContext to build up a list of rendering commands, and then have Unity execute them before the low-level graphics architecture sends instructions to the graphics API.

Unity documentation provides instructions for creating custom rendering, including custom versions of URP and HDRP (see next section). SRP source code is also available on GitHub.

Custom render passes in URP

In URP, you can use **Layer Masks** to define how to render a specific set of objects with custom render passes.

In Unity 2021 LTS and newer versions, the **Render Objects Renderer** for URP provides you with access to custom render passes based on layers that filter the GameObjects to render. This allows artists to create advanced visual effects or gameplay mechanics, such as rendering the silhouette of a character hidden from the camera behind a wall, without writing code. Watch this video tutorial to see it in action.



Draw Character Behind (Render Objects)

Name	DrawCharacterBehind
Event	AfterRenderingOpaques
Filters	
Queue	Opaque
Layer Mask	Character
Pass Names	0
Overrides	
Material	CharacterBehindObjects
Pass Index	0
Depth	✓
Write Depth	☐
Depth Test	Greater
Stencil	☐

In the above example, a different material is applied to the objects under the Opaque Layer Mask.

Custom Passes are done in a slightly different way in HDRP. You can find more information about them in the documentation and sample projects.

Dynamic resolution and upscaling methods

A popular performance optimization for games is dynamic resolution, available in Unity as a Camera setting for the Built-in Render Pipeline, URP, and HDRP. When the application's frame rate is about to drop as a result of being GPU-bound, the resolution will gradually scale down to keep a constant frame rate. You can also manually trigger it via a script.

Take advantage of more advanced upscaling methods with HDRP, such as **NVIDIA Deep Learning Super Sampling (DLSS)** for supported NVIDIA GPUs only, **AMD FidelityFX™ Super Resolution (FSR)**, and **Temporal Anti-Aliasing (TAA) Upscale**. **AMD FSR** is also supported on URP.

NVIDIA DLSS for NVIDIA RTX GPU and Windows

NVIDIA DLSS is natively supported by HDRP in Unity 2021 LTS and newer versions. It uses advanced AI rendering to produce image quality that's comparable to native resolution, while only conventionally rendering a fraction of the pixels. With real-time ray tracing and NVIDIA DLSS, you can create beautiful worlds running at higher frame rates and resolutions on NVIDIA RTX GPUs. DLSS also provides a substantial performance boost for traditional rasterized graphics. Learn more about it on NVIDIA's Unity developer page and blog, as well as in our documentation.

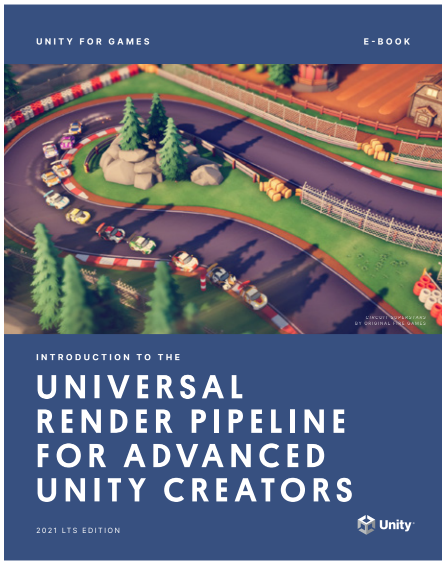
Read this blog for more on the inner workings of the DLSS technology that enables *Naraka: Bladepoint* by 24 Entertainment to run at 4K.

AMD FidelityFX Super Resolution (cross-platform)

AMD FidelityFX Super Resolution (FSR) is available in Unity 2021 LTS. It includes built-in FSR support for HDRP and URP alike. To use FSR, enable **dynamic resolution** in HDRP assets and cameras, then select **FidelityFX Super Resolution 1.0** under the **Upscale filter** option.

FSR is an open-source, high-quality solution for producing high-resolution frames from lower resolution inputs. It uses a collection of algorithms with a particular emphasis on creating high-quality edges, which significantly improves performance – especially compared to rendering at native resolution directly. In other words, FSR enables "practical performance" for costly render operations, such as hardware ray tracing. Learn more about it on AMD's Unity web page and forum.

See how the AMD FidelityFX Super Resolution (FSR) technology is used in the Unity demo *Spaceship*, available on Steam.



Introduction to the Universal Render Pipeline for advanced Unity creators

This guide supports both new and experienced Unity users who want to leverage URP in their projects. Topics covered include setting up URP, Renderer features, shader coding, visual authoring, lighting techniques, performance considerations, and more.

Download the e-book

More resources

Level up your game graphics with these URP tutorials
Harnessing light with URP and the GPU Lightmapper
Create jaw-dropping graphics with these HDRP tutorials
Explore, learn, and create with the new HDRP template

SHADERS

In Unity, shaders are divided into three broad categories:

— **Shaders that are part of the graphics pipeline:** These are the most common type, as they perform calculations to determine the color of pixels onscreen (see Surface Shaders, for example).

— **Compute shaders:** These perform calculations on the GPU outside of the regular graphics pipeline.

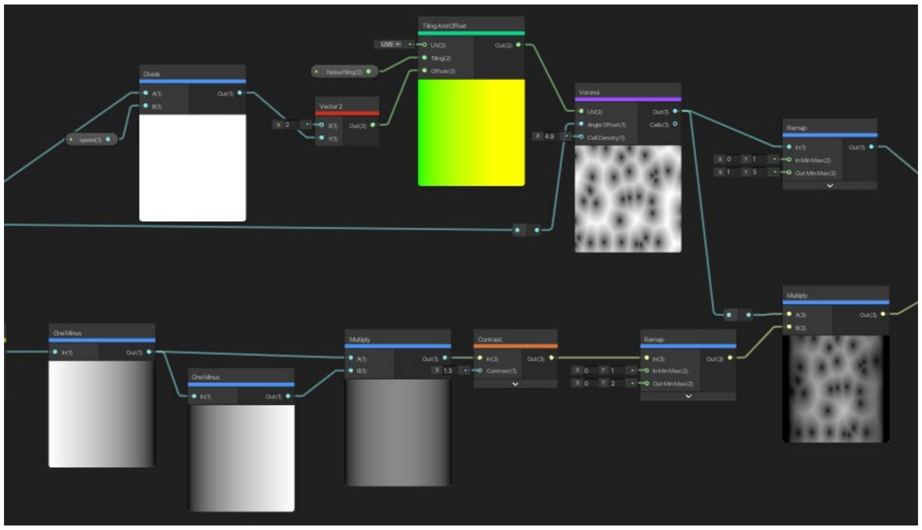
— **Ray tracing shaders:** These are mainly used to generate lighting.

Shader visual authoring: Shader Graph

Unity shaders are written in a Unity-specific language called **ShaderLab**, but you can also create shaders visually with Shader Graph. Instead of writing code, saving, compiling, and testing in-Editor, you can create and connect nodes in Shader Graph's framework, while observing what occurs to the material in real-time, so you can make changes and experiment on the fly.

The **Shader Graph Asset** provides preconfigured options for different materials. The nodes in Shader Graph represent data about the objects to which the material is applied; think of their mathematical functions and procedural patterns.

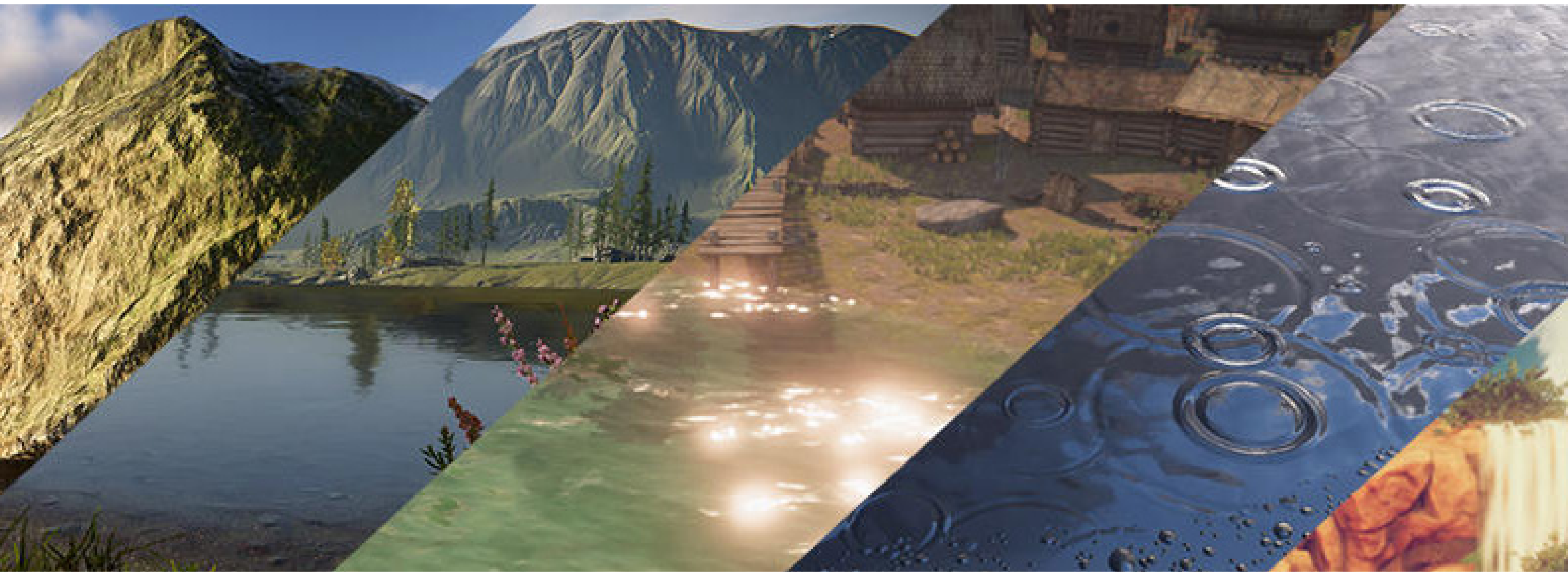
In addition, the Shader Graph system is extensible, meaning that programmers can develop custom Shader Graph nodes. Use Shader Graph to create shaders that work for URP and HDRP (as well as for the Built-in Render Pipeline in 2021 LTS or newer).



Node-based shader creation in Shader Graph

The **Master Stack** is the end point of a Shader Graph that defines the final surface appearance of a shader. It helps users visualize the relationship between operations that take place in the vertex stage – when attributes of the polygon's vertices are calculated – and the fragment stage, when calculations are made to see how the pixels between the vertices look.

Users who are experienced in writing HLSL code can create **Custom Function** blocks in Shader Graph.


Shader Graph enables shader authoring for artists and other team members via connecting nodes in a graph network.

Compute shaders in SRPs

Compute shaders run on the GPU outside of the normal rendering pipeline. They can be used for massively parallel GPU algorithms or to accelerate parts of rendering.

To use them effectively requires in-depth knowledge of GPU architectures and parallel algorithms, DirectCompute, OpenGL Compute, CUDA, or OpenCL. These shaders have better compatibility and versatility, and can be implemented in all Unity render pipelines.

Surface Shaders for the Built-in Render Pipeline

As their name implies, Surface Shaders define the physical characteristics of materials. They calculate the final color of each pixel within a material and perform the light calculations that define the shading of each pixel on the surface. Most Surface Shaders in Unity are extensions of the default **Standard Surface Shader**, which makes the creation process more intuitive and gives artists more freedom to define the look of their surfaces.

More resources

Latest visual effects guide for artists
Shader Graph Master Stack
Experimenting with Shader Graph: Doing less with more
Normal Map compositing using the surface gradient framework in Shader Graph

LIGHTING IN UNITY

Lighting effects with Global Illumination

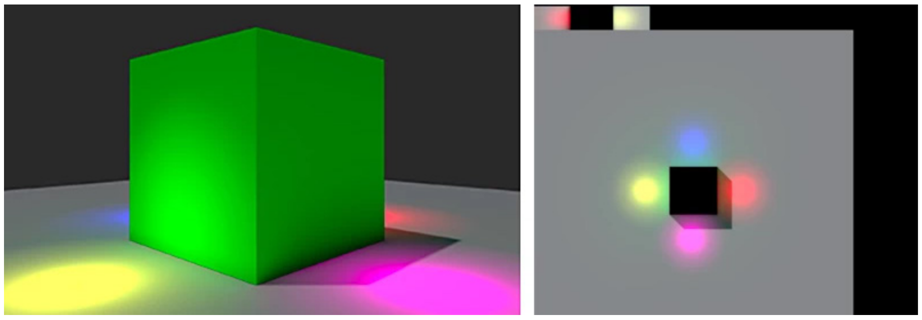
Global Illumination

Lighting in modern games makes great use of Global illumination (GI). GI covers a range of techniques and mathematical models that attempt to simulate the complex behavior of light as it bounces and interacts with the world. Simulating Global Illumination accurately is challenging and can be computationally expensive. For this reason, games often use a range of approaches to handle these calculations beforehand, rather than during gameplay.

Generally speaking, lighting in Unity can be either real-time (direct lighting) or precomputed, though both approaches can be combined to create immersive lighting.

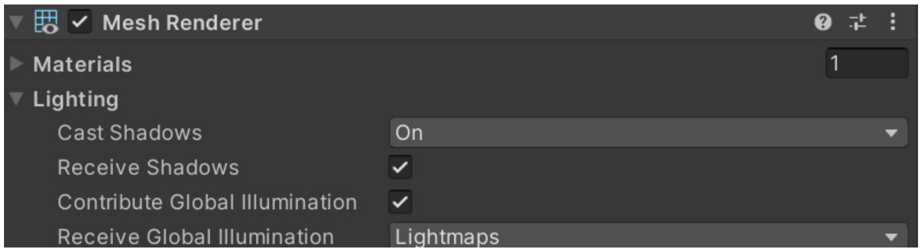
Baked Global Illumination

When baking a lightmap, the effects of light on static objects in the scene are calculated, and the results are written to textures that are overlaid on top of scene geometry to create the effect of lighting.


The lightmap of a static object

In Unity, precomputed lighting is either generated automatically in the background or initiated manually. It's possible to continue working in the Editor while these processes run behind the scenes, as long as you don't move or edit objects contributing to the lightmaps (otherwise the baking process will restart).

The GI system will only consider objects that have the **Contribute Global Illumination** property selected in their **Mesh Renderer** component. To be lit by a lightmap, select the **Lightmaps** option for the **Receive Global Illumination** property. Otherwise, the object will be lit by Light Probes.


Mesh Renderer
Materials 1
Lighting
Cast Shadows: On
Receive Shadows: ✓
Contribute Global Illumination: ✓
Receive Global Illumination: Lightmaps

The main options for GI can be found in the Mesh Renderer.

If a GameObject is set to receive GI from Light Probes, the user must create a **Light Probe Group** object in the scene, and duplicate its probes to cover the scene so that an accurate representation of the lighting is captured. The Light Probe Group captures the lighting from a multitude of points (probes) in space. Their lighting data is stored on disk; then at runtime, each probe-lit object is lit using a lighting blend from the four probes closest to it. Light Probes are particularly useful for dynamic objects, in addition to small objects that do not require high-quality lighting.

To reduce baking time and the amount of memory reserved for lightmaps, we recommend that you maximize the usage of probe-lit objects in your scenes. Only large static objects or objects that require a high lighting fidelity should receive GI from lightmaps.

Progressive Lightmapper

The Progressive Lightmapper is a path tracing-based lightmapping system that delivers progressive updates for baked lightmaps and Light Probes in the Editor. It allows artists to iterate quickly, as it progressively refines and displays the lightmaps in the Scene or Game view within the Editor. Baking times are then made more predictable because the Progressive Lightmapper provides an estimated time while it bakes.

The Progressive Lightmapper bakes GI at the lightmap resolution for each texel individually, without upsampling schemes or relying on any irradiance caches or other global data structures. It is robust in that it allows you to bake selected portions of lightmaps for faster testing and iteration.

The Progressive Lightmapper is available for the Built-in Render Pipeline, URP, and HDRP.

CPU and GPU Lightmappers

You can choose between two backends for the Progressive Lightmapper. The Progressive CPU Lightmapper uses a computer's CPU and system RAM while the Progressive GPU Lightmapper uses the GPU and VRAM.

The Progressive GPU Lightmapper is currently in Preview, which means that it's under active development and subject to change. Check the documentation for the latest development status.

Real-time Global Illumination

Due to its temporal nature, Real-time GI is useful for lights that change slowly and have a high visual impact on your content, such as the sun moving across the sky, or a slowly pulsating light in a closed corridor. Real-time GI is inefficient for fast moving lights or special effects due to performance cost and latency. It's suitable for games on mid-level to high-end PC systems and consoles, as well as high-end mobile devices when used for small scenes with low resolution for real-time lightmaps.

Enlighten

Enlighten is the backend for Real-time GI in Unity via the Lighting window. This system requires the precomputing of a scene for static objects, but similarly supports the seamless addition of lights and materials in real-time. Once a scene has been precomputed, lighting iteration times can be drastically reduced.

The Built-in Render Pipeline offers Enlighten, and as of Unity 2021 LTS, HDRP and URP also support it.

Ray-traced Global Illumination

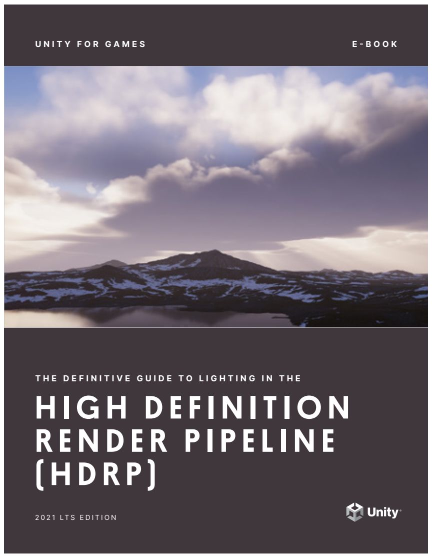
Another form of real-time GI can be attained through ray tracing. Currently, only HDRP offers this capability, which requires a GPU compatible with ray tracing.

The advantage of Ray-traced GI compared to Enlighten or traditional baked lightmaps is that it does not require any precomputing. Furthermore, the lighting is per-pixel: It does not depend on a given texel resolution, nor does it require any specific UV layout to ensure optimal lighting. While iteration times are incredibly short, the GPU requirements for Ray-traced GI remains very high.

That's why this technique can only be used on particular hardware that supports hardware-accelerated ray tracing.

LEGO Builder's Journey by Light Brick Studio for PC platforms, rendered in real-time using HDRP with ray tracing and Deep Learning Super Sampling (DLSS)

To learn more about all of the ray tracing effects available in Unity, such as Ray-traced GI and Reflections, check out this video.


UNITY FOR GAMES
E-BOOK
THE DEFINITIVE GUIDE TO LIGHTING IN THE
HIGH DEFINITION RENDER PIPELINE (HDRP)
2021 LTS EDITION
Unity

The definitive guide to lighting in the High Definition Render Pipeline (HDRP)

Lighting is one of the most complex topics in game development. Learn how to leverage the features of HDRP to create highly optimized, photorealistic games. Concepts covered include cameras, lights, exposure, real-time lighting effects, and refining your shadows.

Download the e-book

More resources

Get acquainted with HDRP settings for enhanced performance
Harness the ray tracing capabilities of Unity's HDRP with NVIDIA
Configuring Global Environment lighting
Introduction to lighting and rendering
Configuring lightmaps

WORLDBUILDING

Unity provides a complete set of tools for building and designing rich and scalable 3D and 2D worlds. This section covers the 3D tools that can help you move from prototyping to polished art.

Many teams, such as the Professional Artistry group, are continuously developing new toolsets to support your most common gamedev needs. Read more about this endeavor at the end of the book.

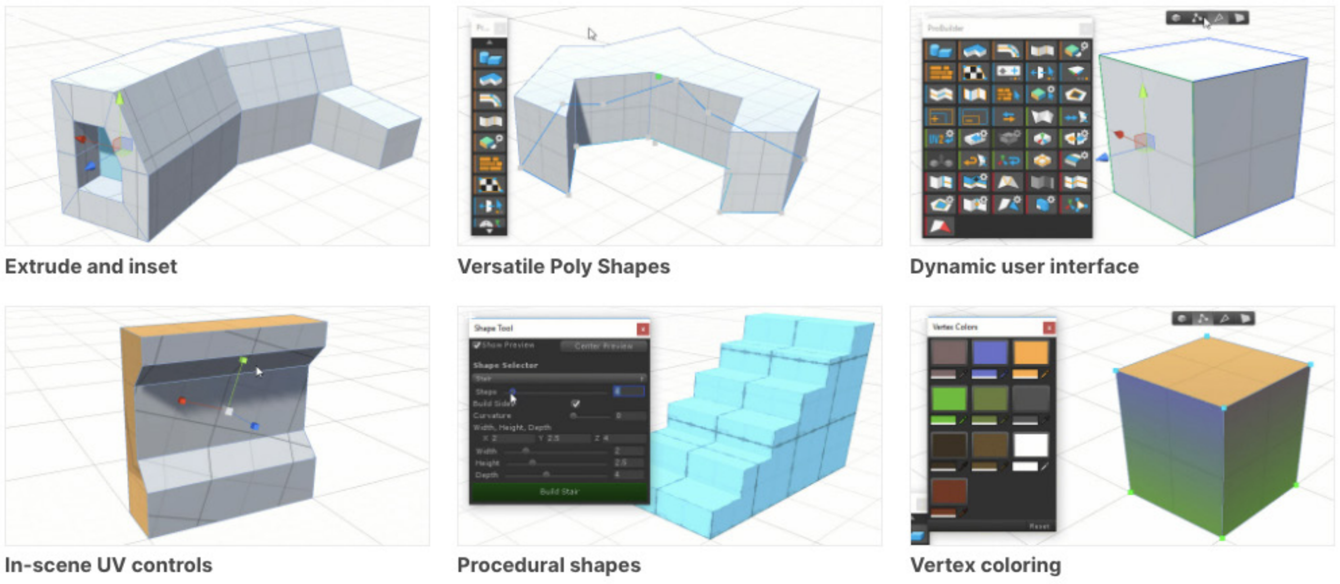
Prototyping

ProBuilder

ProBuilder is a unique hybrid of 3D modeling and level design tools, optimized for building simple geometry with detailed editing features and UV unwrapping.

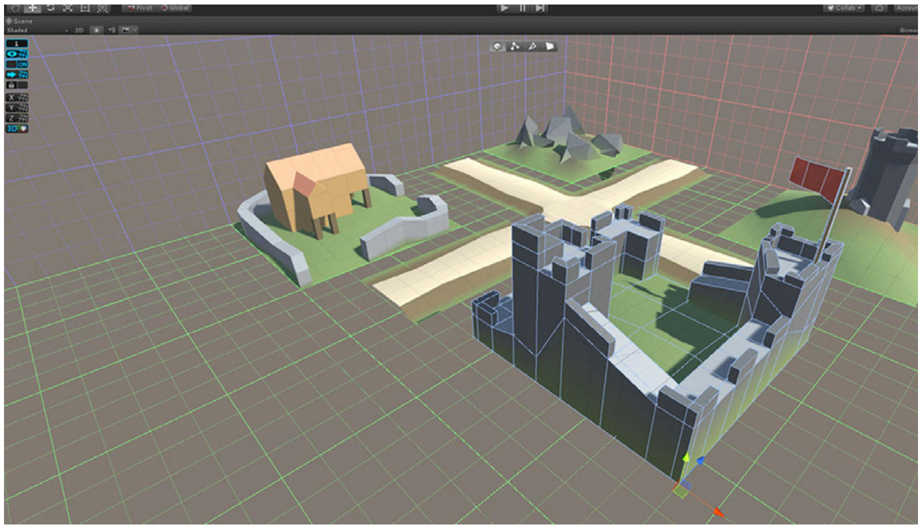
Available via the Package Manager, ProBuilder enables you to prototype structures, complex terrain features, vehicles, weapons, custom collision geometry, trigger zones, and nav meshes at speed.

With its own export options, ProBuilder can also utilize Unity’s roundtripping capabilities with the FBX Exporter. This way, 3D artists can finalize assets from the prototypes in their DCC software.



Some of ProBuilder’s functionality for creating 3D meshes

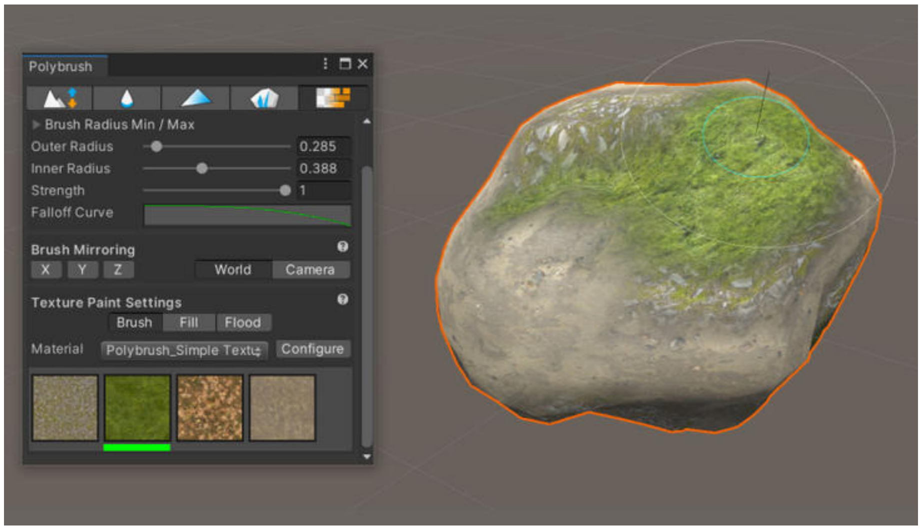
With your preferred tools, use the shape and size reference from the prototype to add more detail and polish to your models. You can create predefined shapes with the Shape tool, which includes a library of both standard and complex geometric shapes corresponding to common objects in level-building. You can also create a 2D shape and extrude it, or use the Bezier Shape tool to create curved meshes that wrap around the line of the spline.


Prototype of a level made in Unity with ProBuilder

Additionally, you can create new meshes with ProBuilder or modify existing ones. Apply a material to the entire mesh, or on selected faces, to create more realistic-looking surfaces during gameplay or while grey-boxing. For example, you might decide to use tiles on the floor or brick on some walls for gameplay testing. Leverage the UV Editor inside the tool to unwrap the UVs or edit them.

Polybrush

Polybrush, also available via the Package Manager, allows you to blend textures and colors, sculpt meshes, and scatter objects directly in the Unity Editor. Combined with ProBuilder, Polybrush constitutes a complete in-Editor level design solution.


Polybrush

Brush Radius Min / Max
Outer Radius 0.285
Inner Radius 0.388
Strength 1
Falloff Curve

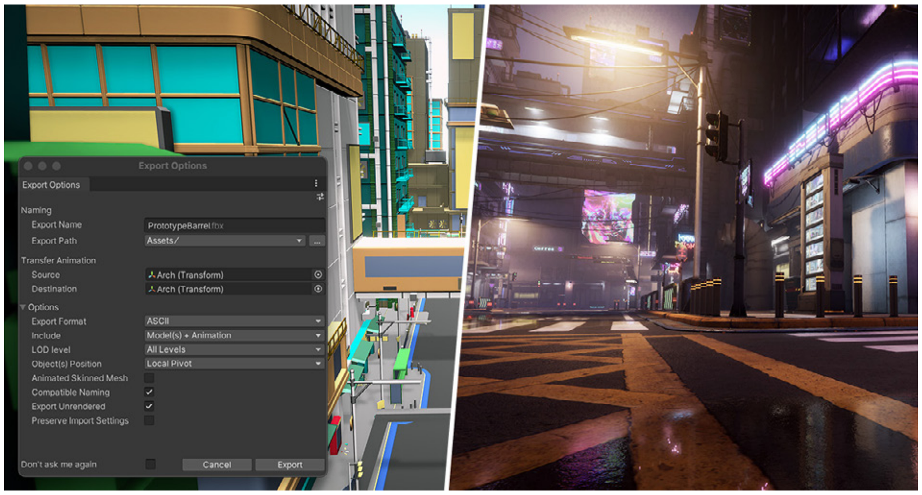
Brush Mirroring
X Y Z World Camera

Texture Paint Settings
Brush Fill Flood
Material Polybrush_Simple Textu Configure

Use Polybrush to add details to your assets made with ProBuilder.

Using the FBX Exporter for refinement

Combine ProBuilder, Polybrush, and the FBX Exporter in your workflow to grey-box models and levels, and facilitate both prototyping and functionality testing. You can use the FBX Exporter to tailor assets to the required dimensions before exporting them to a DCC software for refinement.



Naming
Export Name: PrototypeBarrel.fbx
Export Path: Assets/
Transfer Animation
Source: Arch (Transform)
Destination: Arch (Transform)
Options
Export Format: ASCII
Include: Model(s) + Animation
LOD level: All Levels
Object(s) Position: Local Pivot
Animated Skinned Mesh: ☐
Compatible Naming: ✓
Export Unrendered: ✓
Preserve Import Settings: ☐
Don't ask me again: ☐

A prototype made with ProBuilder on the left, with the FBX Exporter panel open, and the final environment by the artist on the right

A terrain created with Unity Terrain Editor

Terrain sculpting tools

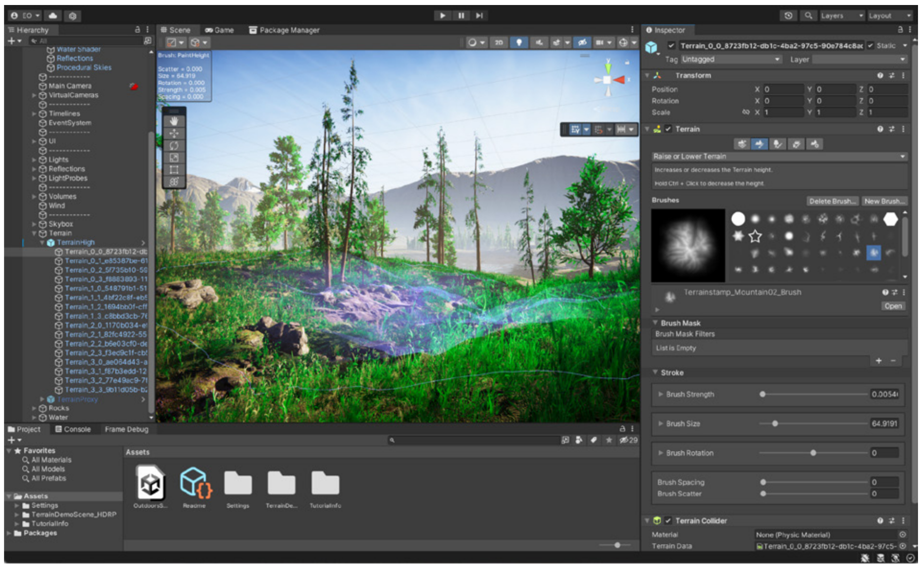
The **Unity Terrain Editor** enables you to create detailed, realistic, and highly optimized terrains. To use the Terrain tools, create or select a **Terrain** object in the **Hierarchy** window.

Discover the new demo scene for URP and HDRP, and learn how it was made in this blog post.

The Terrain component provides certain brushes you can use to raise or lower the terrain, wherever you paint the heightmap of the terrain with the Paintbrush tool. Use it to hide portions of the terrain, add a stamp brush on top of the current heightmap, or simply polish the terrain.

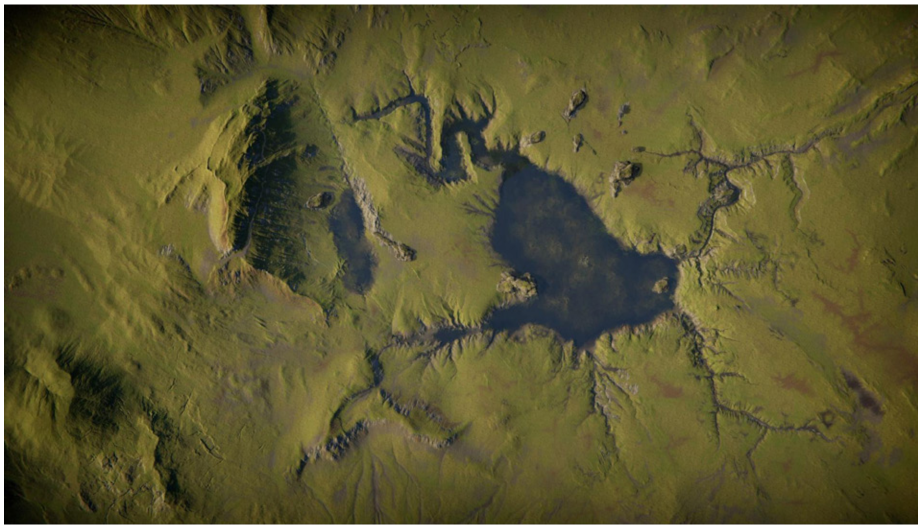
You can leverage the component to paint textures that apply surface changes to the geometry of the terrain. The Terrain Editor includes settings to define draw distances of vegetation, trees, or terrain, plus settings for controlling the terrain mesh resolution, lighting, and wind effects.

The Terrain Tools package adds additional terrain sculpting brushes and tools to your project for creating vivid terrain assets, while easing your workflows. As with other Unity tools, the **Terrain API** helps you make custom Editor tooling so you can use the feature in a way that best suits your team.



Multiple stamps that come with the Terrain Tools package

Check out this series of Unity Learn tutorials for more on how to create and customize your terrain using specific tools and techniques.


Create a terrain with brushes and stamps.

Trees and vegetation

Unity provides brushes as part of the Terrain component. The brushes let you design tree areas and terrain details in-Editor, which is useful for creating large, detailed forests and jungles. The addition of color and size variations make the environment look more organic.

The detail brush works in a similar way to the tree painting tool. It's mainly used for adding details; think simpler meshes with one material, such as grass patches or stones. The tool is also compatible with **SpeedTree**, so you can create trees with advanced visual effects, including smooth LOD transitions, fast billboarding, and natural wind animation.

An instanced mesh with Terrain details

Unity recognizes and imports **SpeedTree Assets** in the same way that it handles other assets. If you're using SpeedTree Modeler 7, make sure to save your `.spm` files once again, using the Unity version of the **Modeler**. If you're using SpeedTree Modeler 8, save your `.st` files directly in the **Unity Project** folder.

SpeedTree Modeler 8 leaves working with Terrain

To create the effect of wind on your terrain, you can add one or more GameObjects with **Wind Zone** components. Trees, grass, and other vegetation within a wind zone bend in a realistic, animated fashion, and the wind itself moves in pulses to create natural patterns of movement among the trees.

The Terrain system brushes and tools help you create these effects, with many options available to find the right balance between the performance and aesthetic that suits your target platform and art direction.

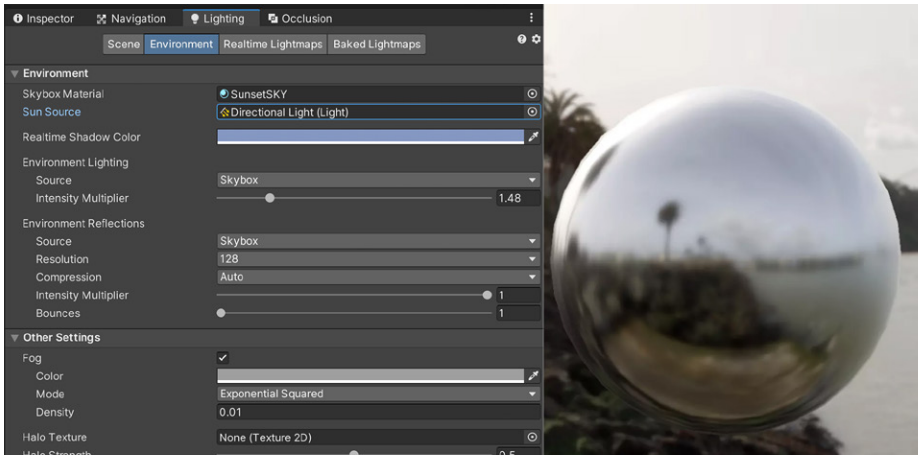
Sky, fog, and clouds

The SRPs include tools and configurations to set up atmospheric effects.
URP priotizes general performance while HDRP aims for photorealistic graphics.

URP

In the **URP Lighting Settings > Environment tab**, you'll find a set of tools to create atmospheric effects.

URP uses **Skybox shaders** to project an environment texture or procedurally generated image for ambient lighting in the scene. In the same window, you can control the **Fog**, **Realtime Shadow Color**, and **Intensity**.



Environment	
Skybox Material	SunsetSKY
Sun Source	Directional Light (Light)
Realtime Shadow Color	
Environment Lighting	
Source	Skybox
Intensity Multiplier	1.48
Environment Reflections	
Source	Skybox
Resolution	128
Compression	Auto
Intensity Multiplier	1
Bounces	1
Other Settings	
Fog	✓
Color	
Mode	Exponential Squared
Density	0.01
Halo Texture	None (Texture 2D)

See the Environment settings in URP's Lighting menu; note that Halo effects are now achieved with Lens Flare (SRP) instead.

HDRP

Through the **Volume framework** in HDRP, you can override the scene property values, which by default, use the settings in the **HDRP Global Settings** section of the **Project** settings.

Volumes can be **Global** or have **Local** boundaries. For instance, use **Local Volumes** to change **Environment** settings, such as fog color and density to alter the mood of different areas in your scene. Use the **Visual Environment** override to define the sky and general ambiance of the scene.

HDRP includes three different techniques for generating skies: HDRI Sky, Gradient Sky, and Physically Based Sky.

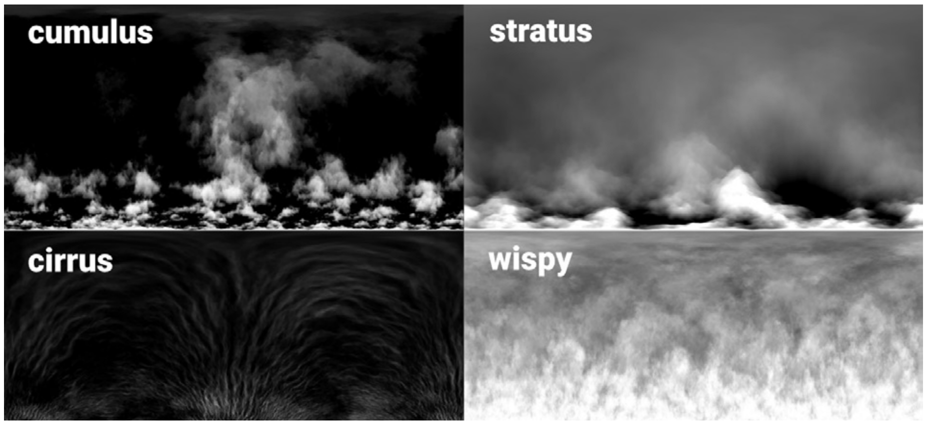
A procedurally generated sky

HDRP also implements **Global Fog** as a Fog override, and features **Volumetric Fog** to light the fog with nearby lights. Volumetric Fog Distance sets the distance (in meters) from the Camera's near clipping plane to the back of its volumetric lighting buffer. This fills the atmosphere with layers that simulate airborne material, partially occluding GameObjects within range. If you want more detailed fog effects than the Fog override can provide, HDRP offers Local Volumetric Fog.

Volumetric Fog in HDRP

In Unity 2021 LTS or newer, the **Cloud Layer** system generates natural-looking clouds to complement your Sky and Visual Environment overrides.

The Cloud Layer is a 2D texture that can be animated with a flowmap, which uses the red and green channels to control vector displacement. In Play mode, the clouds can add some slight motion to your skies, making the background more dynamic. The Cloud Layer sits in front of the sky, with an option to cast shadows on the ground.



The 2D texture uses a cylindrical projection, wherein the RGBA channels all contain different cloud textures (cumulus, stratus, cirrus, and wispy clouds, respectively).

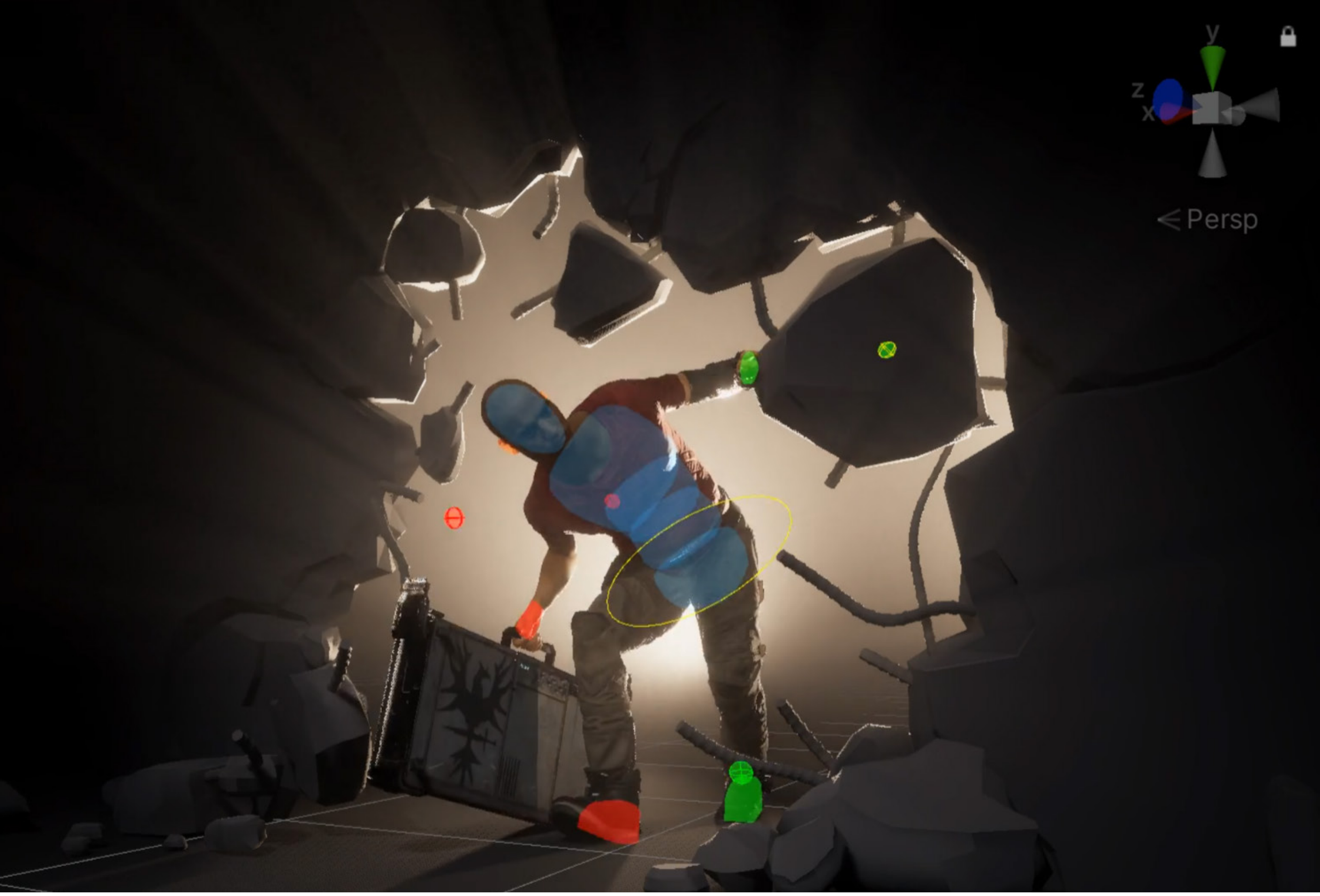
Volumetric Clouds produce realistic clouds with a thick texture, while reacting to lighting and wind.

If your clouds need to interact with light, use **Volumetric Clouds**. These can render shadows, receive fog, and create volumetric shafts of light. Combine them with Cloud Layer clouds or add them separately.

More resources

The Unity game designer playbook
Faster level design with ProBuilder and Polybrush
Asset management with FBX Exporter, ProBuilder, and Polybrush
Experience the new Unity Terrain Demo scenes for HDRP and URP
New Lighting features: Volumetric Clouds, Lens Flare, and Light Anchor
The definitive guide to lighting in HDRP 2021 LTS
Making of *The Heretic*: Environment art

ANIMATION



The character's rig reacts to the environment with a series of constraints.

Note: If you are familiar with Unity's animation system, then you can skip this section and go right ahead to Animation Rigging.

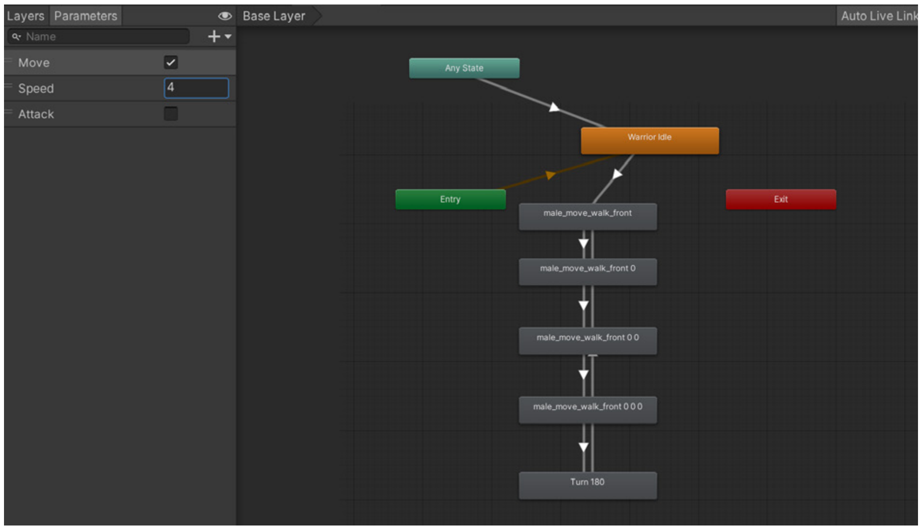
The animation system

Animations in Unity projects are typically created from motion capture, or by using software such as Blender, Autodesk Maya, or Autodesk 3ds Max. Unity's animation system provides tools to modify, refine, procedurally adapt, and blend such animations to bring them to life in the game or interactive experience that you are creating.

Unity's animation system is based on the concept of **Animation Clips**, which contain information about how certain objects should change their position, rotation, or other properties over time. Each clip can be thought of as a single linear recording. Animation Clips from external sources, such as those mentioned above, are brought into Unity (see section on roundtripping) and then organized into a structured flowchart system called an **Animator Controller**.

Animation State Machine

In Unity, the Animator Controller allows you to arrange and maintain a set of Animation Clips and associated **Animation Transitions** for a character or object. In most cases, it's normal to have multiple animations and switch between them when certain game conditions occur. For example, you could switch from a walk Animation Clip to a jump Animation Clip whenever you press the spacebar. The Animator Controller has references to the Animation Clips used within it, and manages the various Animation Clips and Transitions between them using an Animation State Machine. The State Machine could be thought of as a flowchart of Animation Clips and Transitions, or a simple program written in a visual programming language within Unity.



The flow of Animation Clips and parameters in the Animator Controller

Blend trees are effective for hiding complexity. A blend tree doesn't have state, nor does it call back out into code. It simply blends between the different clips based on the parameters that you define. This is significant because you can iterate on blend trees without worrying about breaking the rest of your game. Even more, you can hide a complex web of states to prevent bugs down the road, since you can't tie behavior to most of the animations in a blend tree.

Unity offers **Animation Layers** for managing complex state machines. For instance, use Animation Layers to create a lower-body layer for walking and jumping, and an upper-body layer for throwing objects and shooting. In addition to visual animation, Animation states can trigger sound effects or C# code.

Animation Window

The Animation Window allows you to create and modify Animation Clips directly in Unity. It is designed to act as an alternative to external 3D animation software, or to create simple animations as needed during development. It provides the standard set of tools required for animation like **Keyframes**, **Playhead**, **Animation Timeline**, and **Animation Curves**.

Besides animating movement, you can animate variables of materials and components (almost any GameObject property) and augment your Animation Clips with **Animation Events**, which are functions called at specific points along the Timeline, in-Editor.

Unity's Animation Window also enables you to animate:

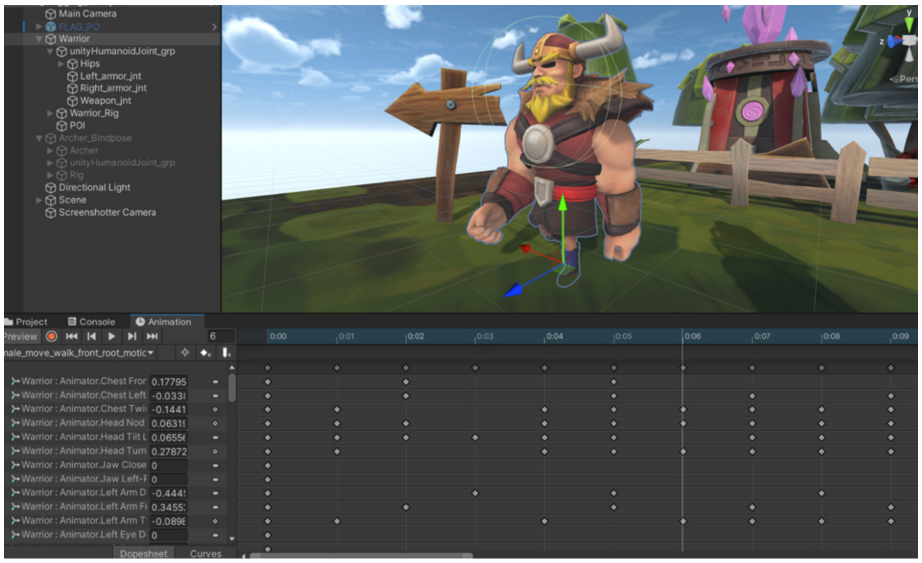
— The position, rotation, and scale of GameObjects

— Component properties including material color, light intensity, and sound volume

— Properties within your scripts, such as Float, Integer, Enum, Vector, and Boolean

— The timing of calling functions within your own scripts

The generated Animation Clips can be used by the Animator Controller or Animation Rigging, and harnessed during gameplay or cinematics with Timeline.



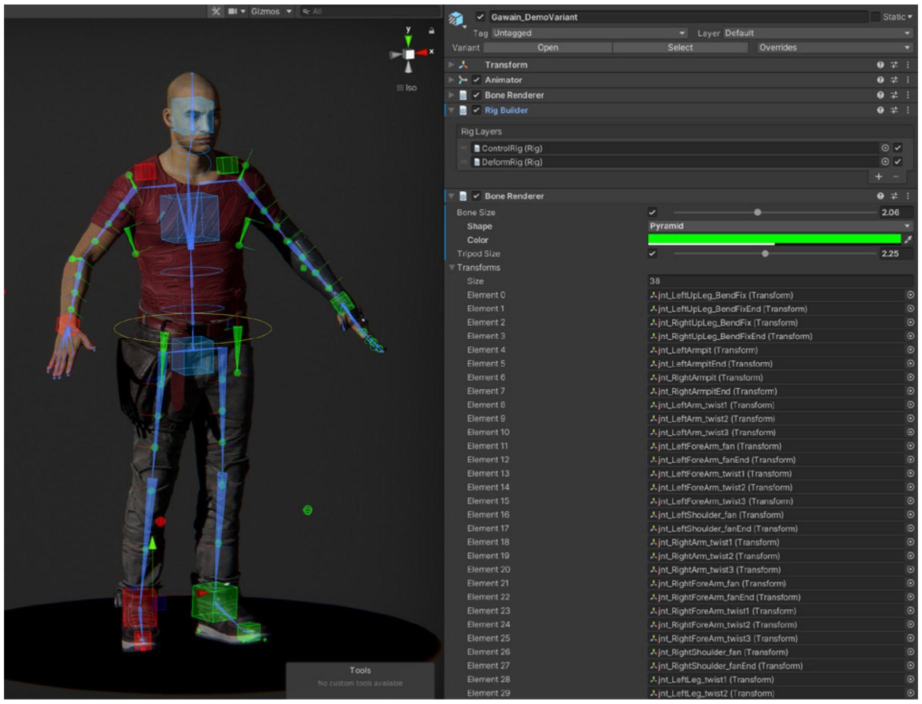
Use Animation tools like Keyframes and Curves for your project.

Animation Rigging

Use the Animation Rigging package to set up procedural motion on animated skeletons at runtime. In this package, there is a set of predefined animation constraints to manually build a control rig hierarchy for your character, or develop your own custom constraints in C#. This makes it possible to complete powerful actions during gameplay, such as world interactions, skeletal deformation rigging, and physics-based secondary motion.

Other benefits of Animation Rigging include the ability to modify animations that you might not have easy access to, or adapt the animation to new situations that were not considered in the original animation.

The Animation Rigging package in Unity allows you to create rigs that override the animation of certain bones, or set constraints for adding procedural motion to animated objects. You can create dynamic animations, but also modify or create new Animation Clips with the Animation tool, or create cinematic sequences with Timeline.



Gawain_DemoVariant — Static
Tag: Untagged — Layer: Default
Variant: Open | Select | Overrides

Transform
Animator
Bone Renderer
Rig Builder

Rig Layers
ControlRig (Rig)
DeformRig (Rig)

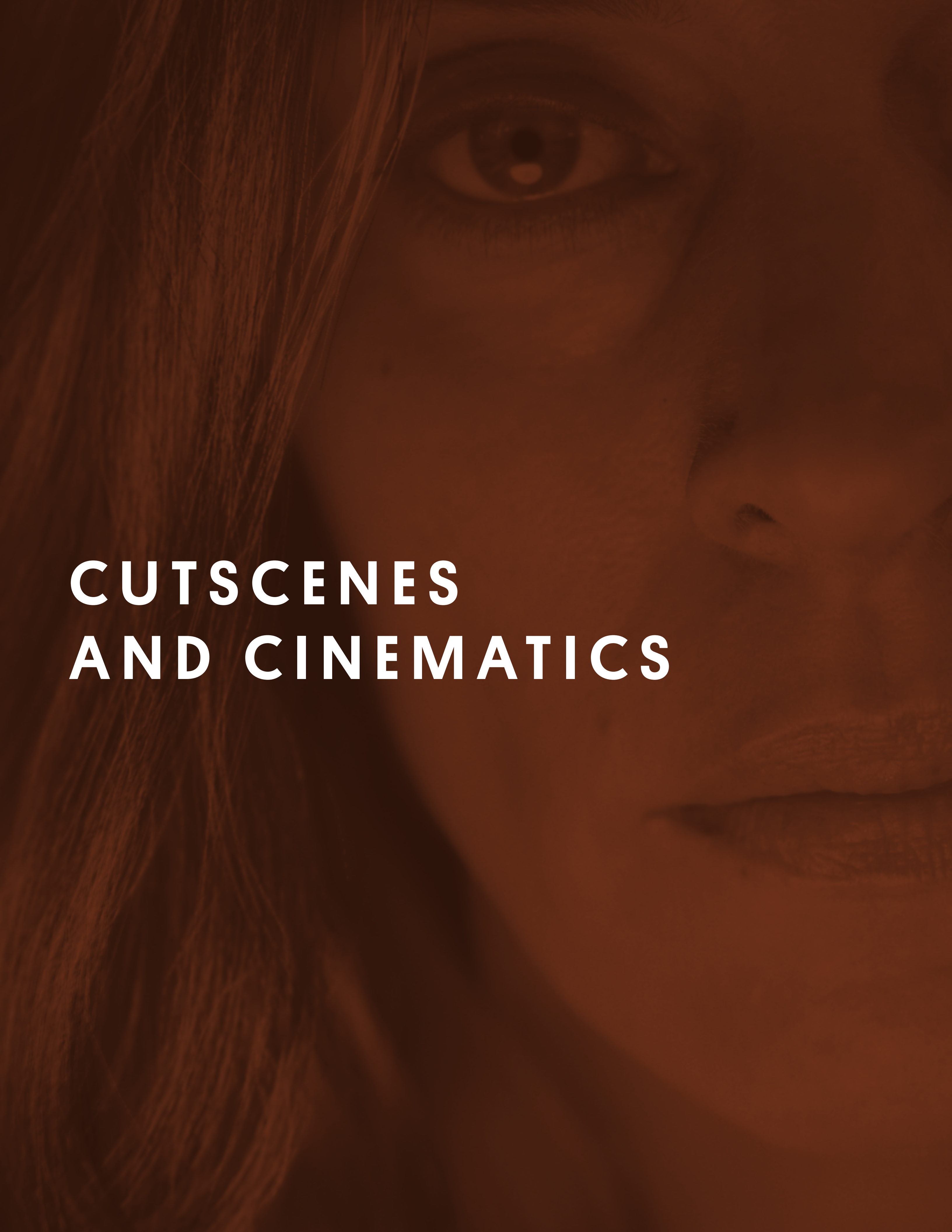
Bone Renderer
Bone Size: 2.06
Shape: Pyramid
Color
Tripod Size: 2.25
Transforms
Size: 38

Element 0	jnt_LeftUpLeg_BendFix (Transform)
Element 1	jnt_LeftUpLeg_BendFixEnd (Transform)
Element 2	jnt_RightUpLeg_BendFix (Transform)
Element 3	jnt_RightUpLeg_BendFixEnd (Transform)
Element 4	jnt_LeftArmpit (Transform)
Element 5	jnt_LeftArmpitEnd (Transform)
Element 6	jnt_RightArmpit (Transform)
Element 7	jnt_RightArmpitEnd (Transform)
Element 8	jnt_LeftArm_twist1 (Transform)
Element 9	jnt_LeftArm_twist2 (Transform)
Element 10	jnt_LeftArm_twist3 (Transform)
Element 11	jnt_LeftForeArm_fan (Transform)
Element 12	jnt_LeftForeArm_fanEnd (Transform)
Element 13	jnt_LeftForeArm_twist1 (Transform)
Element 14	jnt_LeftForeArm_twist2 (Transform)
Element 15	jnt_LeftForeArm_twist3 (Transform)
Element 16	jnt_LeftShoulder_fan (Transform)
Element 17	jnt_LeftShoulder_fanEnd (Transform)
Element 18	jnt_RightArm_twist1 (Transform)
Element 19	jnt_RightArm_twist2 (Transform)
Element 20	jnt_RightArm_twist3 (Transform)
Element 21	jnt_RightForeArm_fan (Transform)
Element 22	jnt_RightForeArm_fanEnd (Transform)
Element 23	jnt_RightForeArm_twist1 (Transform)
Element 24	jnt_RightForeArm_twist2 (Transform)
Element 25	jnt_RightForeArm_twist3 (Transform)
Element 26	jnt_RightShoulder_fan (Transform)
Element 27	jnt_RightShoulder_fanEnd (Transform)
Element 28	jnt_LeftLeg_twist1 (Transform)
Element 29	jnt_LeftLeg_twist2 (Transform)

The Bone Renderer feature in Animation Rigging

More resources

Working with Animation Clips
Improve your workflow with Animation Rigging
Reusing and retargeting animations between rigs



CUTSCENES AND CINEMATICS

This HDRP template includes a tutorial that will walk you through a short cartoon to highlight the professional tools for creating and editing movies, TV shows, and other animated linear content.

Gone are the days of pre-rendered interstitials. With Unity, you can create cutscenes and immersive cinematics equipped with many of the features that once required offline rendering. To get all the features needed for creating your own movies, trailers, and cutscenes in your project, download the **Feature Set: Cinematic Studio** from Package Manager, or start a new project from the Unity Hub with Unity 2021 LTS or newer, and select the **Cinematic Studio Sample**.

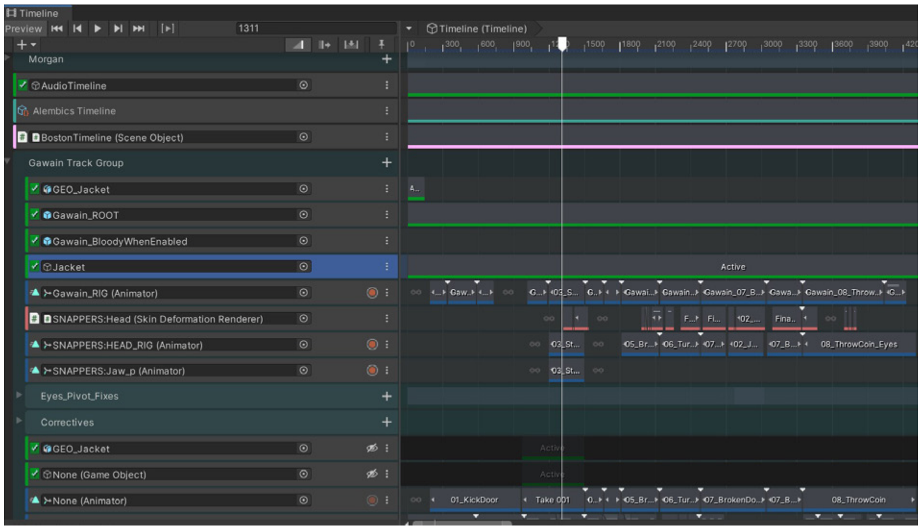
Timeline

Timeline is the backbone of Unity's virtual filmmaking tools. Its interface allows you to control the elements of your scene, including behavior, animations, and enabling or disabling objects. Use it to test your editing skills as both director and editor.

Thanks to the real-time visualization of your composed scene, you don't need to rely as extensively on animatics or storyboards as you would for traditional animation or VFX projects. As in nonlinear video editing, each layer in the Timeline interface is a track.

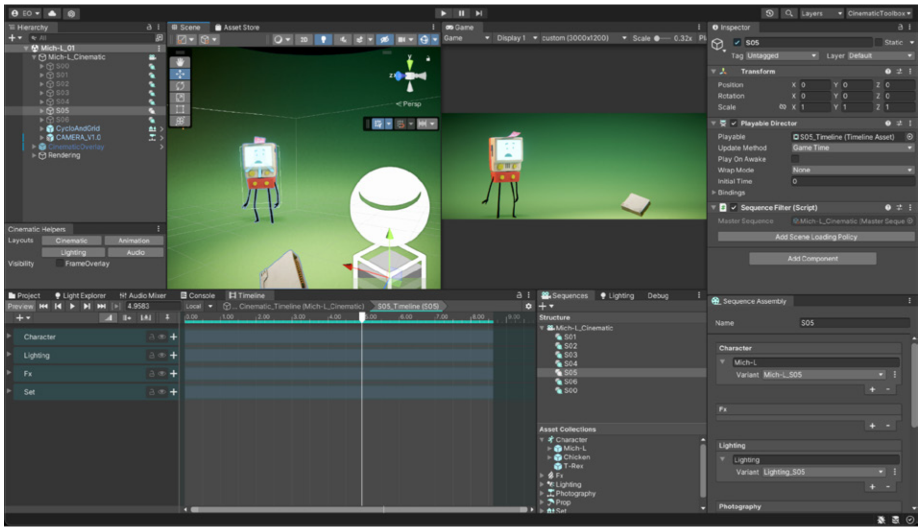
By assembling multiple tracks, you can create a cinematic feature composed of audio, gameplay sequences, particle effects, and more. The machinima of the old days is closer to feature film animation than ever before.

For creative teams, the additional freedom to edit objects ensures that films can undergo significant changes and recuts close to delivery deadlines in a way that is non-destructive – something unthinkable when using most other media.


The demo team customized Timeline with its APIs to edit their film, *The Heretic*, in Unity.

Sequences

Sequences is a new workflow tool in Unity 2021 LTS that helps keep your movie's editorial content well-organized. Use Sequences to strengthen your project's editorial structure and creative content, assemble elements for efficient authoring and collaboration, and test ideas in a non-destructive manner.


The Sequences window brings together the assets from your film to define the ones needed for each sequence, orchestrate the clips from Timeline, and keep your overall project structure organized.

A still from *WiNDUP*, an animated short made with Unity and produced with Unity Recorder

Unity Recorder

The Unity Recorder will render your offline videos, trailers, cutscenes, or films made with any of the render pipelines. It includes several recorder types to either generate videos, stills, arbitrary output variables (AOVs) or capture specific render passes (with HDRP). In Unity 2021 LTS and newer versions, Unity Recorder supports frame interpolation to maintain broadcast-quality motion blur and record path-traced frames.

Alembic support

Alembic is a file format used for the optimized playback of complex animations, such as hair or cloth. You can import and export Alembic files (.abc), plus record and stream animations in the Alembic format right in Unity. The Alembic format is commonly used to transfer animations between applications in the form of baked data.

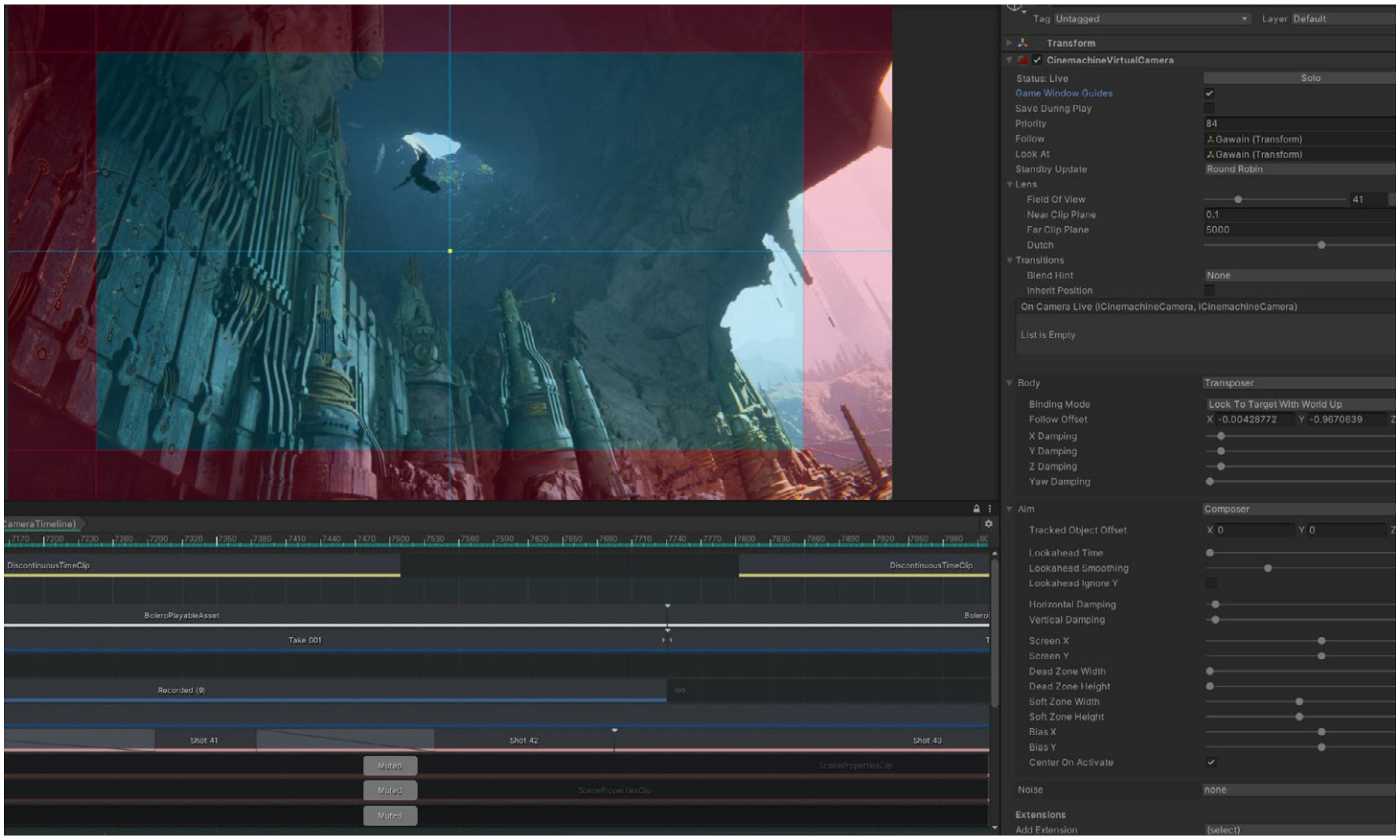
FBX Exporter

The FBX Exporter, covered in a previous section, allows you to roundtrip content to DCCs. VFX and 3D artists can iterate on geometry or animation generated in Unity, and send the final assets back to Unity without a hassle.

Cinemachine

Cinemachine is a virtual camera operator, or an intent-driven camera system, that enables you to dolly, track, and zoom your game camera, similar to how a Hollywood camera operator works on set.

Use Cinemachine to frame and follow your subject without coding – it handles all of the complex logic for you. You can simply plug a set of modules onto a special camera rig, enter a few parameters, and watch Cinemachine do the rest.



Tag	Untagged
Layer	Default
Status: Live	
Game Window Guides	✓
Save During Play	
Priority	84
Follow	Gawain (Transform)
Look At	Gawain (Transform)
Standby Update	Round Robin
Lens	
Field Of View	41
Near Clip Plane	0.1
Far Clip Plane	5000
Dutch	
Transitions	
Blend Hint	None
Inherit Position	
On Camera Live (ICinemachineCamera, ICinemachineCamera)	List is Empty
Body	Transposer
Binding Mode	Lock To Target With World Up
Follow Offset	X -0.00428772 Y -0.9670639
X Damping	
Y Damping	
Z Damping	
Yaw Damping	
Aim	Composer
Tracked Object Offset	X 0 Y 0
Lookahead Time	
Lookahead Smoothing	
Lookahead Ignore Y	
Horizontal Damping	
Vertical Damping	
Screen X	
Screen Y	
Dead Zone Width	
Dead Zone Height	
Soft Zone Width	
Soft Zone Height	
Bias X	
Bias Y	
Center On Activate	✓
Noise	none
Extensions	
Add Extension	(select)

Designing cinematics with Timeline and Cinemachine

If you've already developed your own camera system, Cinemachine can work alongside your custom camera solution.

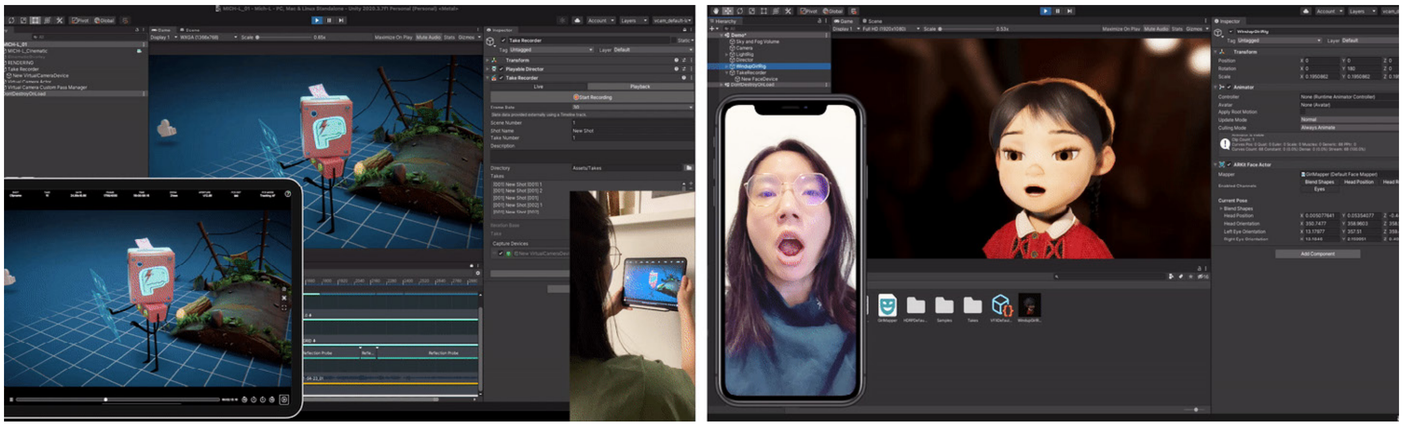
Cinemachine is effective for both in-game and cutscene camera animation. Use it across all game genres: first-person shooters (FPS), third-person, side-scroller, top-down, and real-time strategy (RTS).

Live Capture

The Live Capture package helps you capitalize on the power of augmented reality (AR)-enabled iPads and iPhones to capture real-life performances in Unity. Use the package to connect to the following two companion apps:

— **Unity Virtual Camera:** An app for using your mobile device to drive the Unity camera intuitively and record motion, which is ideal for organic, handheld shots.

— **Unity Face Capture:** An app for animating facial expressions. For an animator, capturing this information and applying it to a character can significantly reduce the time needed to animate a character's face.


Unity Virtual Camera (left) and Unity Face Capture (right) in action

Find out what other filmmakers are creating with Unity on the Film, animation, and cinematics page.

More resources

It's showtime! New tools for cinematic creators
Discover how Unity's virtual production tools can up your game
Blending gameplay and storytelling with Timeline: Cutscenes and game graphics
Empowering storytellers with real-time technology: Part I and Part II
Entertainment posts on the Unity Blog

VISUAL EFFECTS

Unity currently offers two tools for creating particle systems, the **Built-in Particle System** and the **VFX Graph**. In 2021 LTS or newer, both systems are compatible with the Built-in Render Pipeline, URP, and HDRP.

Built-in Particle System

The Built-in Particle System gives you full read/write access to the system and the particles it contains from C# scripts. You can use the **Particle System API** to create custom behaviors for your particle system.

The Particle System component, which can be modified in the Inspector, has a powerful set of properties that are organized into modules for ease of use. In those modules, you can define the emission rate and duration of the particles, the shape of the emissor, as well as the particles' appearance, movement, and behavior during their lifetime.

Additionally, the Built-in Particle System includes a **Renderer module** where you can access the module's settings to determine how a particle's quad or mesh is transformed, shaded, or overdrawn by other particles. You can even render particle trails.

The **Lights module** is a quick way to add real-time lighting to your particle effects. It can be used to make systems cast light onto their surroundings, for example, with fire, fireworks, or lightning. Here, you can have the lights inherit a variety of properties from the particles they are attached to.

In order to pass information about each particle to custom shaders or C# scripts, enable vertex data streams. Some key features include the **Force Field** component to apply force to particles in Particle Systems and the **C# Job System** integration for writing performant C# behaviors.

Benefits of the Built-in Particle System include full multiplatform support, support for the Built-in Render Pipeline, integration with Unity's Physics system, particle light support, access to individual particle data via C# scripts, and robust scalability for projects that have many Prefabs with built-in particles.

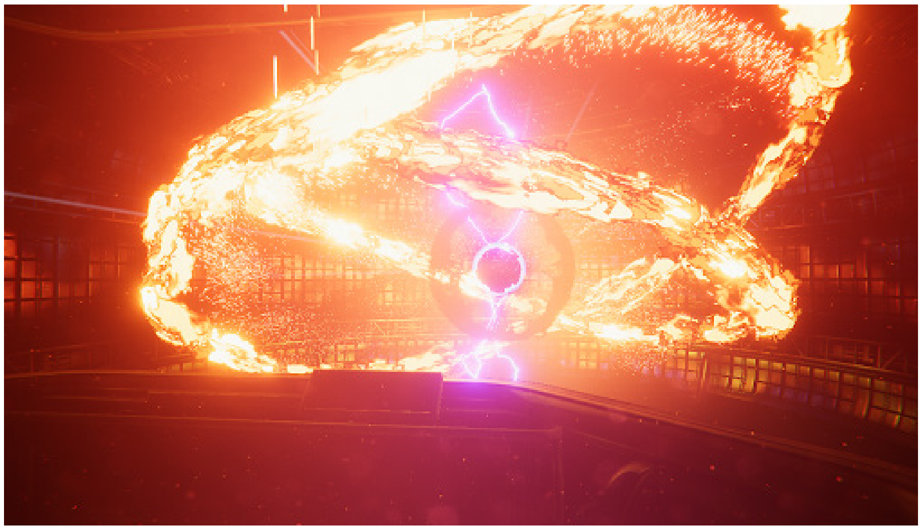
Visual effects made with the Built-in Particle System

VFX Graph

The VFX Graph enables the authoring of effects using node-based visual logic. You can use it for simple effects as well as complex simulations. Unity stores VFX Graphs in **Visual Effect Assets** that you can use on the **Visual Effect Component**. In fact, you can use a Visual Effect Asset multiple times in your scene.

The VFX Graph simulates particle behavior on the GPU, allowing it to simulate more particles than the Built-in Particle System can. But since the VFX Graph is simulated on the GPU, it works exclusively on compute-capable platforms.

Some benefits include the ability to have more particles with faster simulation, customizable behaviors, extensibility (to create subgraphs, templates, etc.), camera buffer access (for HDRP), and native Shader Graph integration. Use any custom shader created in Shader Graph to target VFX Graph. These shaders can use new lighting models like HDRP hair or fabric, or they can modify particles at the vertex level to enable effects like birds with flapping wings, wobbling particles like soap bubbles, and much more.


Visual effects made with VFX Graph

VFX Graph components

The VFX Graph shares a number of similarities with Shader Graph. Here is a brief explanation of VFX Graph's basic components: contexts, blocks, nodes, and properties.

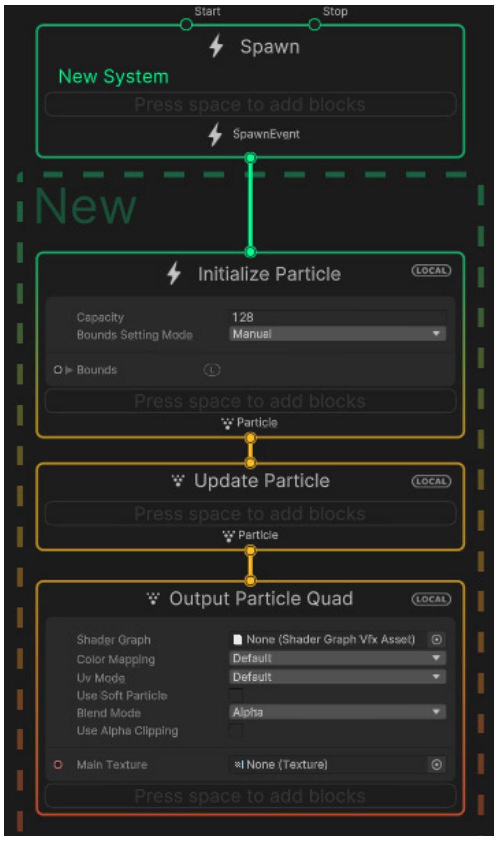
Contexts represent the order of operations for particles being processed:

— **Spawn:** Controls particle spawning

— **Initialize:** Sets the initial particle values

— **Update:** Controls particle behavior over time

— **Output:** Defines how the resulting particle should be rendered to the screen

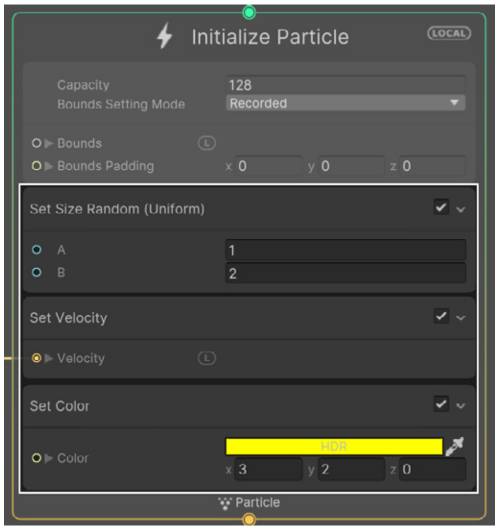


Capacity 128
Bounds Setting Mode Manual

Shader Graph None (Shader Graph Vfx Asset)
Color Mapping Default
Uv Mode Default
Use Soft Particle
Blend Mode Alpha
Use Alpha Clipping
Main Texture None (Texture)

A look at the flow of nodes from VFX initialization to output

Blocks are operations added to each context (similar to calling functions in C#). They can be added by pressing the space bar within a context, or by right-clicking and selecting **Create Block**.



Capacity 128
Bounds Setting Mode Recorded
Bounds
Bounds Padding x 0 y 0 z 0

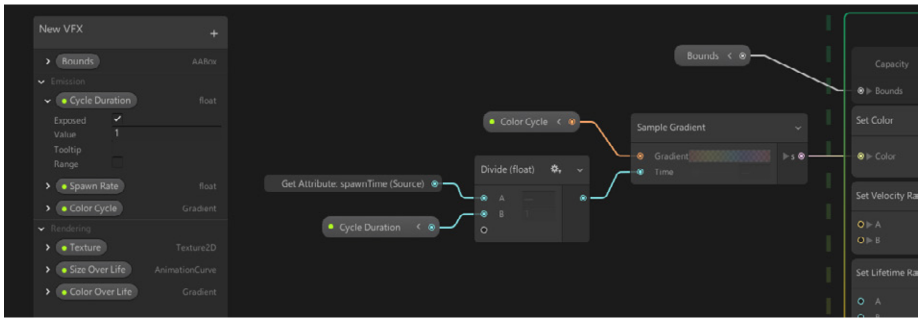
A 1
B 2

Velocity

Color x 3 y 2 z 0

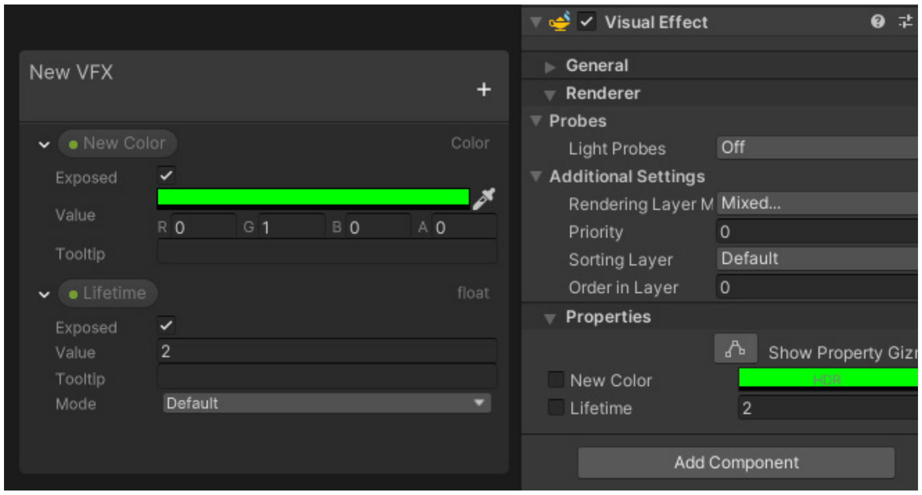
The starting properties of a particle group

Nodes perform individual operations linked together to complete large calculations, similar to how they are leveraged in Shader Graph. They can be used instead of variables to drive the properties of Blocks, as shown below.



The Blackboard shows global variables and allows you to set up parameters to display in the Inspector.

Properties represent data that can be added to the Blackboard and used throughout the graph. When a property is set to be exposed, it can be modified via the Inspector or C# scripts outside of the VFX Graph editor.



New Color		Color
Exposed	✓	
Value	🟩 R 0 G 1 B 0 A 0	
Tooltip		
Lifetime		float
Exposed	✓	
Value	2	
Tooltip		
Mode	Default	

Light Probes	Off
Rendering Layer M	Mixed...
Priority	0
Sorting Layer	Default
Order in Layer	0
New Color	🟩 HDR
Lifetime	2

Blackboard properties exposed in the Inspector

For more information, check out the VFX Graph manual.

Post-processing effects, created with the Built-in Render Pipeline, in the Unity demo *Neon Challenge*

Post-processing

Unity provides a variety of post-processing effects that greatly improve the appearance of your application with little setup time. You can use these effects to simulate physical camera and film properties, or to create stylized visuals.

Popular effects include **Depth of Field**, **Vignette**, **Tonemapping** (including custom LUTs), **Shadows/Midtones/Highlights**, **Split Toning**, and **Chromatic Aberrations**, as well as typical color adjustments for contrast and saturation.

To learn more about some of these post-processing effects, among other HDRP features, take a look at this Unite Now session.

Render pipeline solutions for post-processing

The Built-in Render Pipeline does not include a post-processing solution by default. To use post-processing effects with the Built-in Render Pipeline, download the **Post-Processing package**.

Meanwhile, URP and HDRP each include their own post-processing solution. No additional packages are required. The Scriptable Render Pipeline's implementation uses the **Volume** component. You can also add post-processing effects to your Camera in the same way that you add any other Volume Override. Please note that experienced Unity programmers can skip this section.

UNITY FOR GAMES
E-BOOK
THE DEFINITIVE GUIDE TO CREATING
ADVANCED VISUAL EFFECTS IN UNITY
2021 LTS EDITION
Unity

The definitive guide to creating advanced visual effects in Unity

This e-book prepares artists and technical artists alike to create real-time visual effects with the VFX Graph. Not only does it explain how to achieve common effects through an in-depth walkthrough of the VFX Graph, it guides readers to integrate the package with other Unity and third-party tools, and provides samples to illustrate key workflows.

Get the e-book

More resources

Spaceship Visual Effect Graph demo
Create beautiful and complex effects with the VFX Graph
Making *The Heretic*: The VFX-driven character Morgan
Getting started with the Particle System
Introduction to the post-processing stack

SCRIPTING IN UNITY

Unity's Visual Scripting solution

Note: Experienced Unity programmers can skip this section.

Unity provides many tools to take on the most common use cases in game development. As a technical artist, however, you might encounter situations that call for some custom tooling to help your team work more efficiently. Understanding how coding works in Unity and how to leverage the **Scripting API** gives you the flexibility to streamline any part of the production – from the creation of procedural systems or Visual Scripting tools for designers, to highly customized asset import workflows.

If you lack coding experience and would like to create scripts in Unity, we recommend that you read this short introduction and peruse the list of beginner resources below.

Unity supports C#, an industry-standard language with some similarities to Java or C++. All gameplay and interactivity developed in Unity rests on three fundamental building blocks: GameObjects, Components, and Variables.

Any object in a Unity game is a GameObject, including characters, lights, special effects, props, and so on. GameObjects can't do anything on their own. For them to become animate, you need to give GameObjects their properties by adding components.

Components define and control the behavior of the GameObjects they are attached to. A simple example would be the creation of a light, which involves attaching a Light component to a GameObject – or, for instance, adding a Rigidbody component to an object to make it fall.

Components have any number of editable properties, or variables, that can be tweaked in the Inspector window of the Unity Editor or via C# scripts. In the above example, some properties of a light include range, color, and intensity.

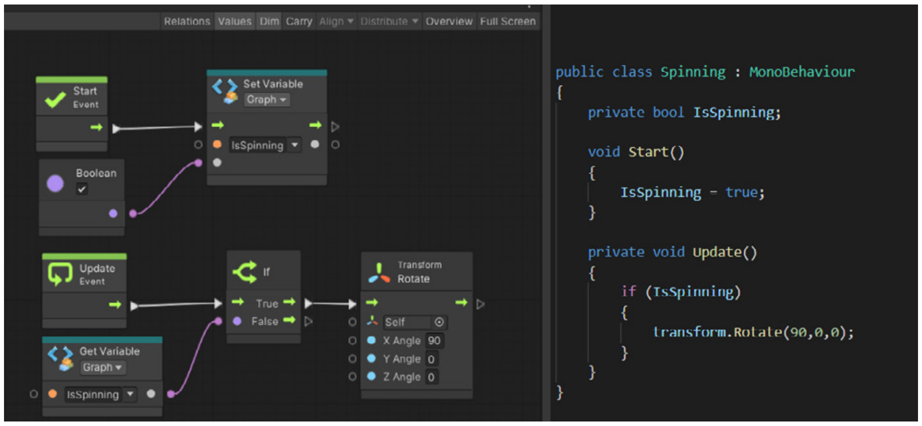
Unity Learn provides excellent free tutorials and courses for learning how to create C# scripts in Unity. Please see the following three recommended learning paths:

— **Short scripting tutorials:** Get started with these beginner and intermediate guiding projects.

— **Creator Kit for coding:** Complete this kit in a few hours to explore the basics of C# code in the context of an action role-playing game (RPG).

— **Create with Code:** Take on a comprehensive course with over 37 hours of instruction.

Visual Scripting in Unity



```
public class Spinning : MonoBehaviour
{
    private bool IsSpinning;

    void Start()
    {
        IsSpinning = true;
    }

    private void Update()
    {
        if (IsSpinning)
        {
            transform.Rotate(90,0,0);
        }
    }
}
```

Unity's Visual Scripting solution

Unity's Visual Scripting system enables you to create game logic for your Unity projects without writing traditional code. In Unity 2021 LTS or newer, Visual Scripting is installed by default as a package from the Editor. For earlier versions, including 2019 LTS and 2020 LTS, you'll need to install it from the Unity Asset Store.

Visual Scripting comprises visual, node-based graphs that both programmers and non-programmers can use to design final logic and create quick prototypes. It offers a more accessible way to collaborate on scripts and can be employed alongside existing C# scripts in your projects. You can even use it to watch how your graph-based logic is executed in real-time.

There are four basic concepts in Visual Scripting that are used when building games:

— **Nodes:** These are the basic building blocks for creating interactions with Visual Scripting. They can do a variety of things, such as listening for events, determining the value of a variable, and modifying a component of a GameObject. Nodes can be connected together to define data or logic for your visual scripts.

— **Graphs:** Graphs visually represent a set of nodes that create logic for your application. There are two types of Graphs, **Script Graphs** and **State Graphs**. A Script Graph is used to define behaviors for execution at a specific time and in a specific order. In comparison, a State Graph contains states (sets of user-defined behaviors) and transitions between them whenever certain user-defined parameters are met.

— **Script Machines and State Machines:** These are components that let you use a Script or State Graph in your application. Script or State Graphs cannot be used until they're attached to a GameObject in your scene using one of these components.

— **Variables:** These are containers that store values and data, which can then be referenced by nodes within your graphs. The value inside of a variable can change during runtime.

Visual Scripting provides programmers and TAs with a solution for creating extensions, templates, and tools to better collaborate with artists and designers. It ensures that everyone on a project can work in a streamlined process, regardless of whether or not they know C#.

The Unity game designer playbook

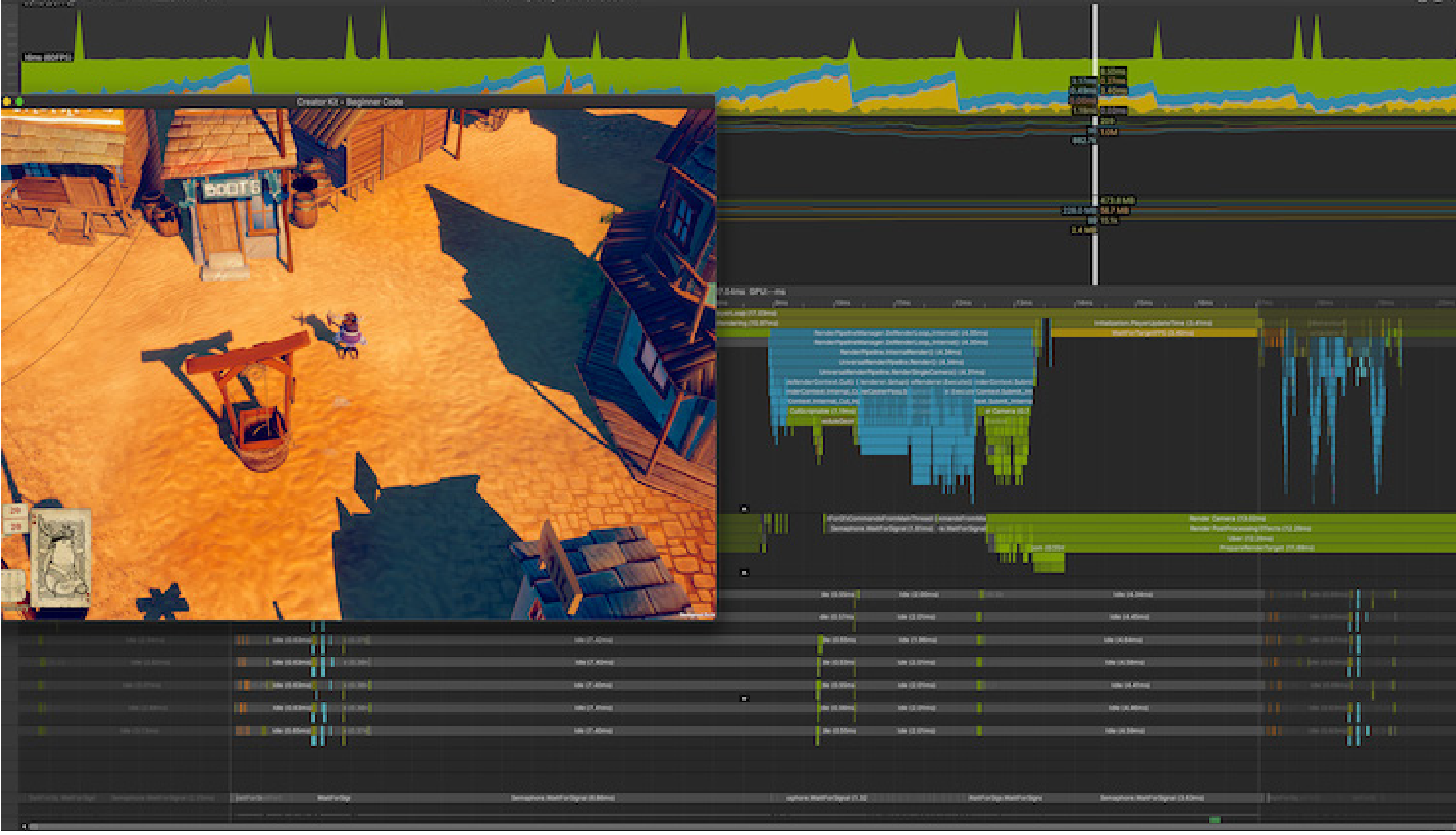
This guide presents the essential tools for designers to prototype, create, and refine gameplay in Unity. Get an in-depth introduction to scripting in C#, Unity Visual Scripting, and learn how to accomplish game design tasks with minimal coding through input control, character controllers, grey-boxing, and so much more.

Download the e-book

More resources

More about Unity Visual Scripting
Introduction to Visual Scripting tutorial

PROFILING AND DEBUGGING


Overview of the Profiler displaying stats of a game running in-Editor

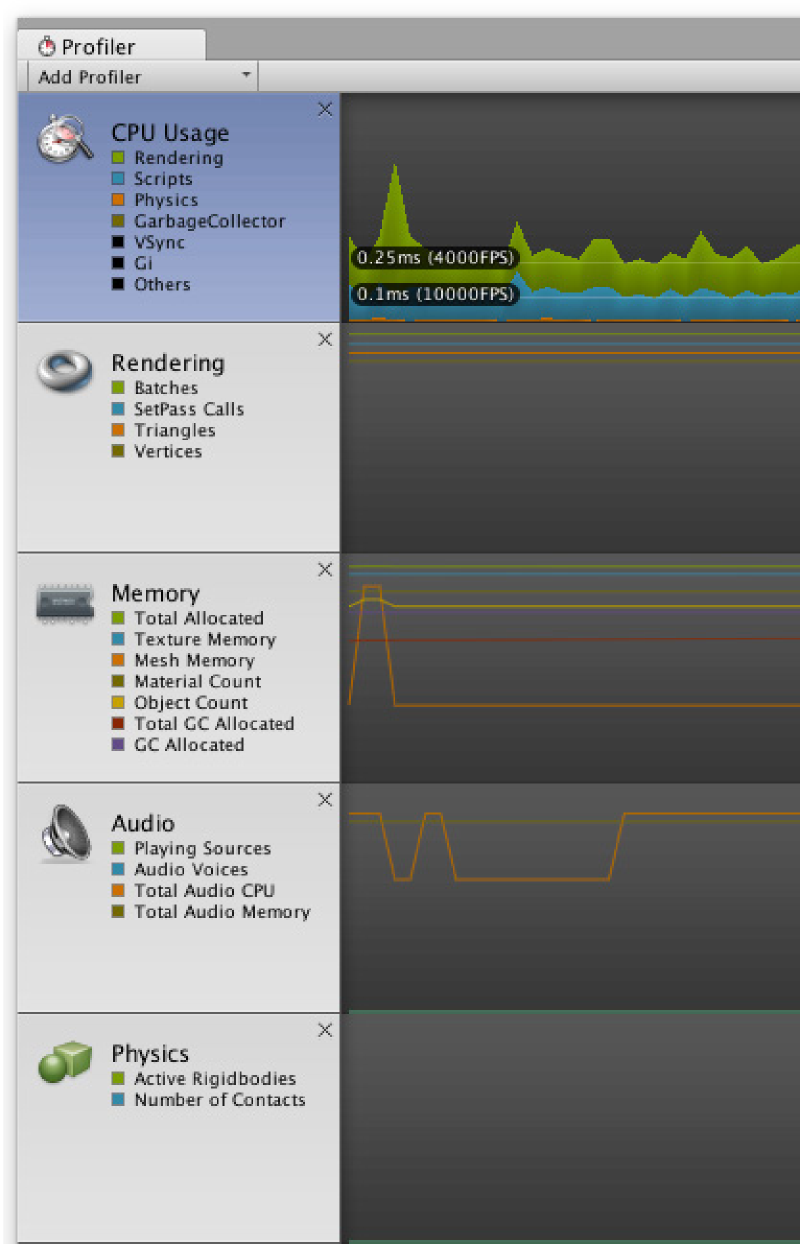
Profiler

The Unity Profiler is a tool you can use to get performance information about your application. You can connect it to devices on your network to see how your application runs on the intended release platform. You can also run it in the Editor to get an overview of resource allocation while developing your application.

The Profiler gathers and displays data on the performance of your application in areas such as the CPU, memory, rendering, and audio. It's a useful tool to identify areas for performance improvement in your application. You can pinpoint how your code, assets, scene, settings, camera, rendering, and build affect your application's overall performance. The Profiler displays the results in a series of charts, so you can clearly see where spikes in your application's performance occur.

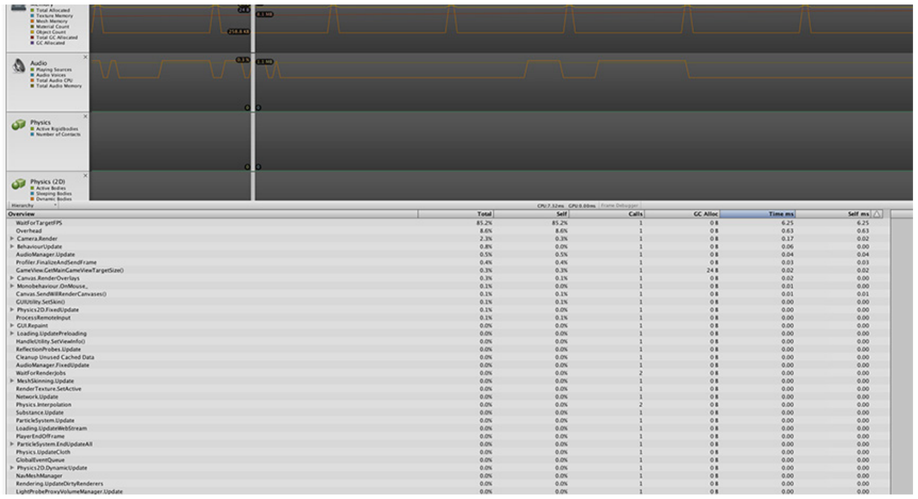
In addition to using the built-in Profiler, you can use the low-level native plug-in **Profiler API** to export profiling data to third-party profiling tools, and the **Profiling Core package** to customize your profiling analysis. You can also add powerful profiling tools, such as the **Memory Profiler** and the **Profile Analyzer**, to your project to analyze performance data in further detail.

To access the Profiler window, go to **Window > Analysis > Profiler**. On the left of the Profiler window, you'll see a column of **Profiler modules**. Each module displays information about a specific aspect of your content. There are separate modules for CPU usage, GPU usage, rendering, memory usage, audio, physics, and networking.



Navigating the different systems to pin down areas of optimization

The bottom half of the Profiler window displays detailed information from the selected module, on the selected frame of data.



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
WaitForTargetFPS	85.2%	85.2%	1	0 B	6.25	6.25
Overhead	8.6%	8.6%	1	0 B	0.63	0.63
▸ Camera.Render	2.3%	0.3%	1	0 B	0.17	0.02
▸ BehaviourUpdate	0.8%	0.0%	1	0 B	0.06	0.00
AudioManager.Update	0.5%	0.5%	1	0 B	0.04	0.04
Profiler.FinalizeAndSendFrame	0.4%	0.4%	1	0 B	0.03	0.03
GameView.GetMainGameViewTargetSize()	0.3%	0.3%	1	24 B	0.02	0.02
▸ Canvas.RenderOverlays	0.3%	0.1%	1	0 B	0.02	0.00
▸ Monobehaviour.OnMouse_	0.1%	0.0%	1	0 B	0.01	0.00
Canvas.SendWillRenderCanvases()	0.1%	0.1%	1	0 B	0.01	0.01
GUIUtility.SetSkin()	0.1%	0.1%	1	0 B	0.00	0.00
▸ Physics2D.FixedUpdate	0.1%	0.0%	1	0 B	0.00	0.00
ProcessRemoteInput	0.1%	0.1%	1	0 B	0.00	0.00
▸ GUI.Repaint	0.0%	0.0%	1	0 B	0.00	0.00
▸ Loading.UpdatePreloading	0.0%	0.0%	1	0 B	0.00	0.00
HandleUtility.SetViewInfo()	0.0%	0.0%	1	0 B	0.00	0.00
ReflectionProbes.Update	0.0%	0.0%	1	0 B	0.00	0.00
Cleanup Unused Cached Data	0.0%	0.0%	1	0 B	0.00	0.00
AudioManager.FixedUpdate	0.0%	0.0%	1	0 B	0.00	0.00
WaitForRenderJobs	0.0%	0.0%	2	0 B	0.00	0.00
▸ MeshSkinning.Update	0.0%	0.0%	1	0 B	0.00	0.00
RenderTexture.SetActive	0.0%	0.0%	1	0 B	0.00	0.00
Network.Update	0.0%	0.0%	1	0 B	0.00	0.00
Physics.Interpolation	0.0%	0.0%	2	0 B	0.00	0.00
Substance.Update	0.0%	0.0%	1	0 B	0.00	0.00
ParticleSystem.Update	0.0%	0.0%	1	0 B	0.00	0.00
Loading.UpdateWebStream	0.0%	0.0%	1	0 B	0.00	0.00
PlayerEndOfFrame	0.0%	0.0%	1	0 B	0.00	0.00
▸ ParticleSystem.EndUpdateAll	0.0%	0.0%	1	0 B	0.00	0.00
Physics.UpdateCloth	0.0%	0.0%	1	0 B	0.00	0.00
GlobalEventQueue	0.0%	0.0%	1	0 B	0.00	0.00
▸ Physics2D.DynamicUpdate	0.0%	0.0%	1	0 B	0.00	0.00
NavMeshManager	0.0%	0.0%	1	0 B	0.00	0.00
Rendering.UpdateDirtyRenderers	0.0%	0.0%	1	0 B	0.00	0.00
LightProbeProxyVolumeManager.Update	0.0%	0.0%	1	0 B	0.00	0.00

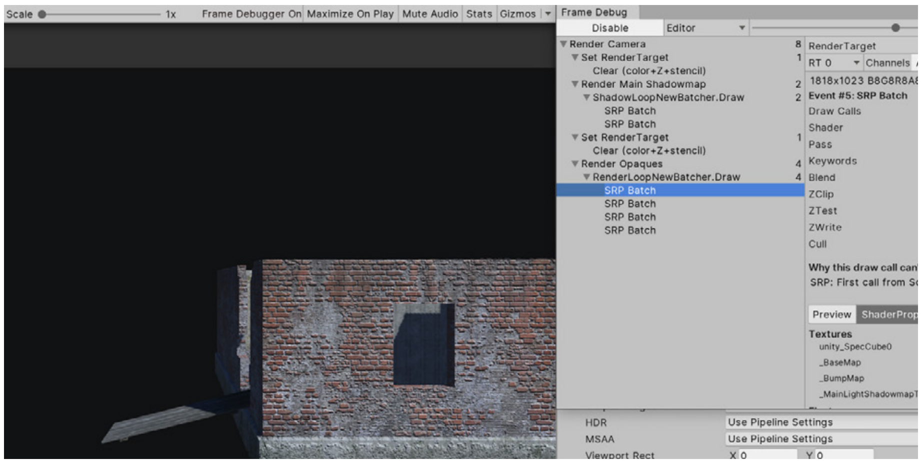
The list of tasks and the time they take to process in the frame

The type of data shown in the above image depends on the profiler that is currently selected. For example, if you select the **Memory Usage Profiler**, you will see information regarding the assets that use the most memory and the total amount of memory used. Meanwhile, selecting the **Rendering Profiler** will show statistics on the number of objects being rendered or the number of rendering operations being performed.

Frame Debugger

The Frame Debugger is a handy tool that allows you to freeze playback for a game running on a particular frame. This way, you can view the individual draw calls used to render that frame. The Debugger also lets you go through frames one by one, so you can see how the scene is constructed in greater detail. This helps you debug your projects whenever specific scenes cause frame rate issues.

The Debugger can be found under the **Windows > Analysis** tools. Once you enable it, the Game view will freeze and you will be able to observe the different draw calls taking place onscreen, in the paused frame. For example, when you click on any of the draw mesh calls on the left side of the window, it will update the Game window with what the draw mesh has actually rendered.



```
▼ Render Camera                                8
  ▼ Set RenderTarget                           1
      Clear (color+Z+stencil)
  ▼ Render Main Shadowmap                      2
    ▼ ShadowLoopNewBatcher.Draw                2
        SRP Batch
        SRP Batch
  ▼ Set RenderTarget                           1
      Clear (color+Z+stencil)
  ▼ Render Opaques                             4
    ▼ RenderLoopNewBatcher.Draw                4
        SRP Batch
        SRP Batch
        SRP Batch
        SRP Batch
```

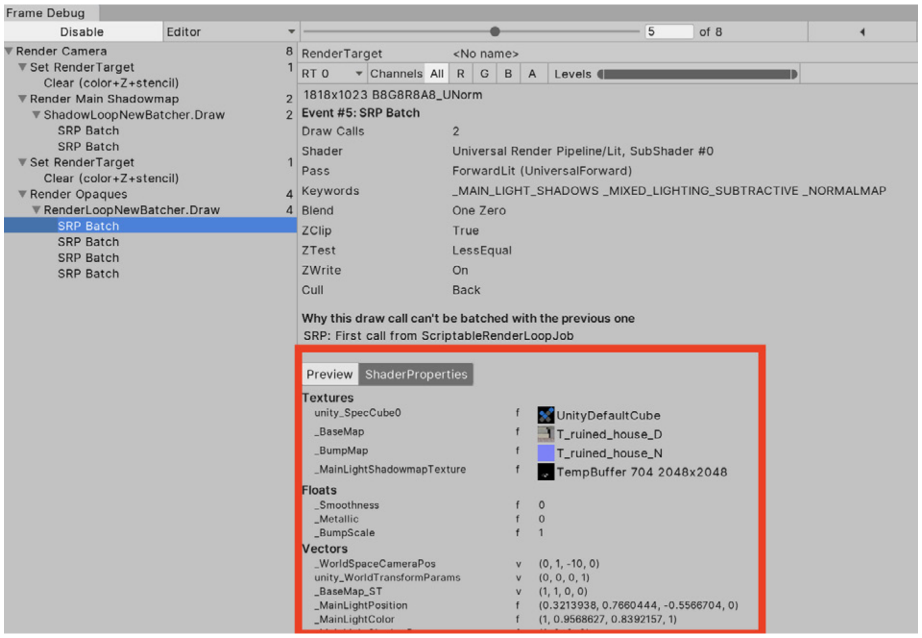
```
RenderTarget
RT 0        Channels
1818x1023 B8G8R8A8
Event #5: SRP Batch
Draw Calls
Shader
Pass
Keywords
Blend
ZClip
ZTest
ZWrite
Cull
Why this draw call can
SRP: First call from Sc
Textures
  unity_SpecCube0
  _BaseMap
  _BumpMap
  _MainLightShadowmapT
```

```
HDR            Use Pipeline Settings
MSAA           Use Pipeline Settings
Viewport Rect  X 0     Y 0
```

A render pass in the Frame Debugger

The **Frame Debug** window is segmented into sections so you can conveniently find the information that you're looking for. On the left side is the sequence of draw calls and other events such as post-processing effects. The right side of the window shows more information on the selected draw call, such as the geometry detail and the shader used for rendering.

ShaderProperties also display the shader stages that were used. It's helpful to know the current state of the selected shader and properties used, so you can confirm whether your shaders work properly during the draw process.


Frame Debug
Disable | Editor | 5 of 8

```
▼ Render Camera                          8
  ▼ Set RenderTarget                     1
      Clear (color+Z+stencil)
  ▼ Render Main Shadowmap                2
    ▼ ShadowLoopNewBatcher.Draw          2
        SRP Batch
        SRP Batch
  ▼ Set RenderTarget                     1
      Clear (color+Z+stencil)
  ▼ Render Opaques                       4
    ▼ RenderLoopNewBatcher.Draw          4
        SRP Batch
        SRP Batch
        SRP Batch
        SRP Batch
```

RenderTarget	<No name>
RT 0	Channels All R G B A Levels

1818x1023 B8G8R8A8_UNorm

Event #5: SRP Batch

Draw Calls	2
Shader	Universal Render Pipeline/Lit, SubShader #0
Pass	ForwardLit (UniversalForward)
Keywords	_MAIN_LIGHT_SHADOWS _MIXED_LIGHTING_SUBTRACTIVE _NORMALMAP
Blend	One Zero
ZClip	True
ZTest	LessEqual
ZWrite	On
Cull	Back

Why this draw call can't be batched with the previous one
SRP: First call from ScriptableRenderLoopJob

Preview | ShaderProperties

Textures

unity_SpecCube0	f	UnityDefaultCube
_BaseMap	f	T_ruined_house_D
_BumpMap	f	T_ruined_house_N
_MainLightShadowmapTexture	f	TempBuffer 704 2048x2048

Floats

_Smoothness	f	0
_Metallic	f	0
_BumpScale	f	1

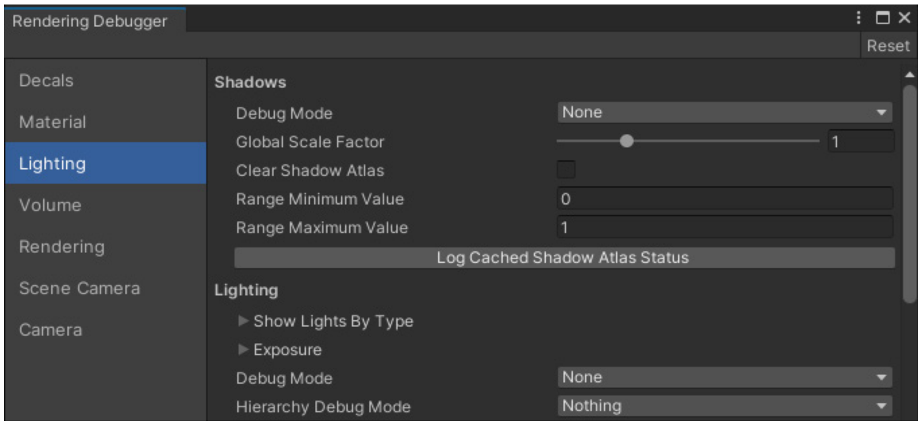
Vectors

_WorldSpaceCameraPos	v	(0, 1, -10, 0)
unity_WorldTransformParams	v	(0, 0, 0, 1)
_BaseMap_ST	v	(1, 1, 0, 0)
_MainLightPosition	f	(0.3213938, 0.7660444, -0.5566704, 0)
_MainLightColor	f	(1, 0.9568627, 0.8392157, 1)

The ShaderProperties view in the Frame Debugger

Rendering Debugger

You can use the Rendering Debugger for both URP and HDRP projects in Unity 2021 LTS or newer. The Rendering Debugger lets you further visualize lighting, rendering, and material properties. With this level of visualization, you can identify rendering issues and optimize scenes and rendering configurations. Find more details on the different Debug views in the URP and HDRP documentation.



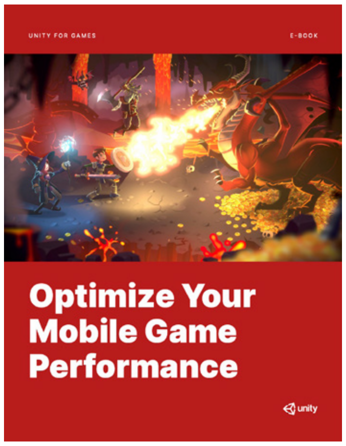
Shadows

Debug Mode	None
Global Scale Factor	1
Clear Shadow Atlas	
Range Minimum Value	0
Range Maximum Value	1

Lighting

Debug Mode	None
Hierarchy Debug Mode	Nothing

In a development build, the Rendering Debugger is available in the Editor during Play mode and at runtime in the Unity Player (on any device).

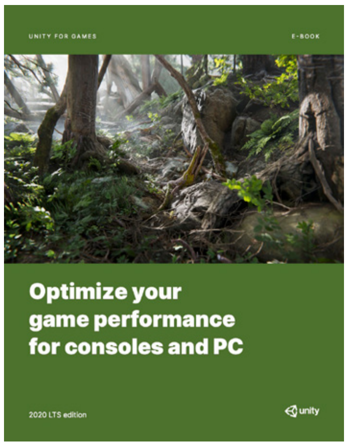


Optimize your game performance

One of our most comprehensive guides collects over 75 actionable tips and best practices from Unity's expert team of support engineers. Learn how to optimize your mobile game to run smoothly on as many devices as possible, while providing players with a refined experience.

The in-depth advice compiled here derives from actual projects involving top game studios. It will help you make the most of Unity and boost your mobile game's performance.

Download the e-book



There are also plenty of great optimization tips for developers targeting console and PC. Check out this e-book, which consists of 80 actionable best practices centered on optimization for PC and console, from Unity's team of expert engineers.

Download the e-book

More resources

How to profile and optimize a game
Introduction to profiling in Unity

2D GAME DEVELOPMENT

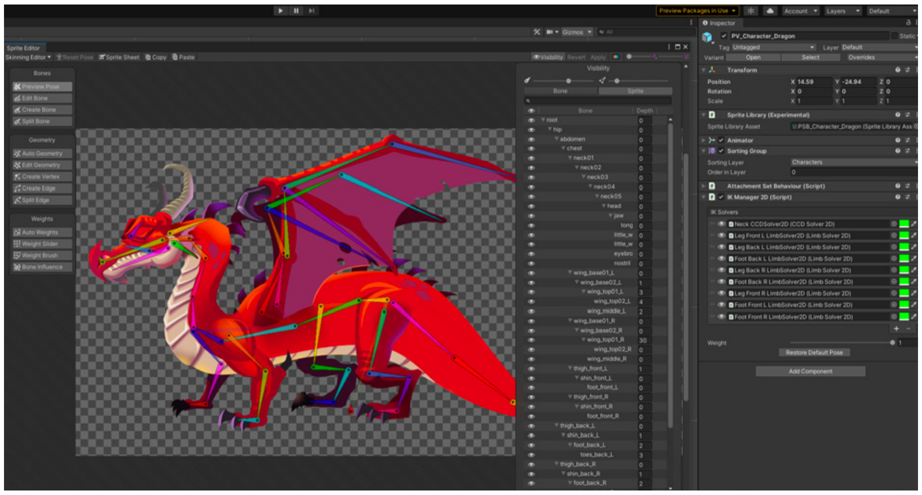
Unity's demo *Dragon Crashers* is available on the Asset Store.

From RPGs to Match-3s, some of today's most successful and top-grossing games are 2D games. The workflows for 2D and 3D development are similar in Unity, which is especially advantageous for mobile studios that often alternate between 2D and 3D production. Unity's comprehensive suite of 2D tools provides teams with the flexibility and scalability to create any kind of 2D game or experience across multiple platforms.

Unity's native 2D tools

Beyond stellar graphics, Unity offers the features you need for 2D game development: Sprite support, 2D skeletal animation with Inverse Kinematics (IK), worldbuilding with Tilemap-based grids or organic shapes, 2D Physics, Sprite Atlasing tools for packing sprites into textures, and more.

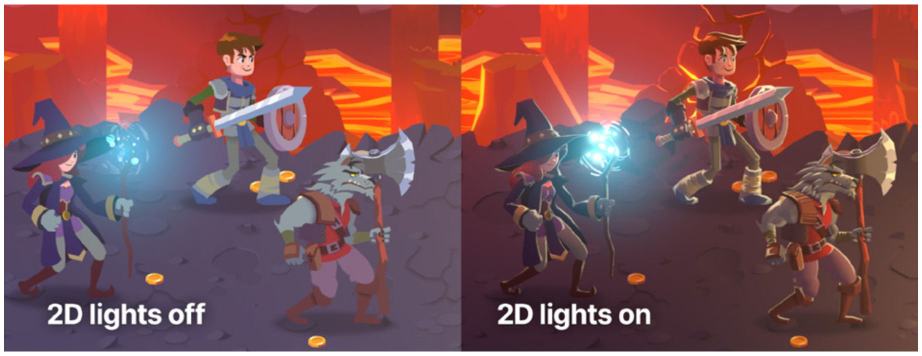
These 2D tools are compatible with both the **Built-in Renderer** and **2D Renderer** included in URP; the latter of which enables visual effects such as 2D Lights, post-processing, and visual shader authoring via Shader Graph.



The skeleton and parts of the Dragon Boss enemy from *Dragon Crashers*

2D Animation

Rig sprites and set up bones to create smooth, skeletal animation with 2D Animation tools. Use the animation features together with the PSD Importer to easily import your character artwork from Photoshop into Unity. This way, you can enable animation and directly incorporate layered artwork into your project. Even more, these tools come with swapping functionality to create characters that reuse the same rigs and animations.



Moving the light information to Secondary Textures (normal maps and mask maps) serves to create more dynamic and immersive lighting effects in 2D.

2D graphics: Lights and shaders

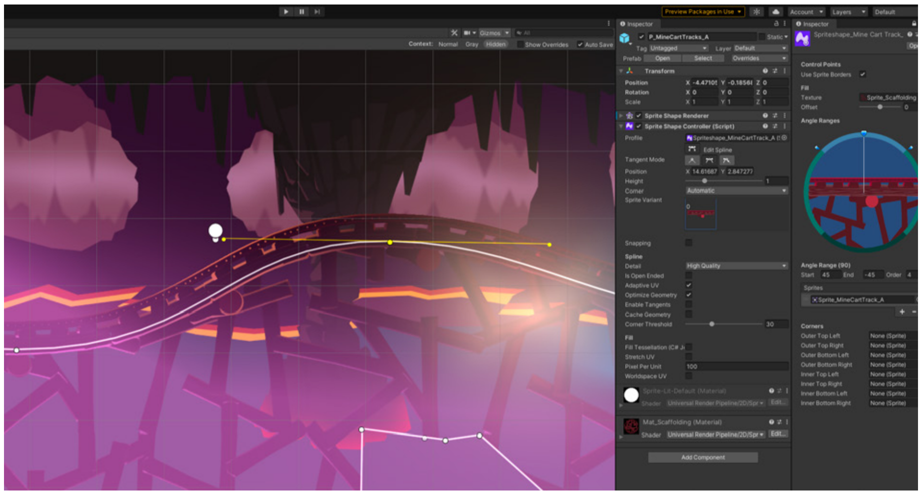
You can make your 2D visuals or gameplay more immersive with the dynamic **2D Lighting system**. The 2D Lighting system included with URP consists of artist-friendly tools and runtime components that help you quickly create lit 2D scenes. It operates through core Unity components like the **Sprite Renderer** and **2D Light**, which act as 2D counterparts to familiar **3D Light** components.

In the Inspector, you can conveniently apply parameters, such as light colors, intensity, falloff, and blending effects. Additionally, by including normal maps in your sprite, you can add an extra layer of possibilities with 2D Lights, and readily author shaders by building them visually with Shader Graph.

Worldbuilding

Tilemaps

Develop large, grid-based **Hexagonal** and **Isometric** worlds that are optimized for size and performance with the Tilemap system. Tilemaps have less overhead than individual sprites, plus the **Tilemap API** and additional features offer an array of creative possibilities. Create custom brushes, add GameObjects to your **Tile Palette**, and apply different sprite **Sorting Layer** logic based on your needs.



P_MineCartTracks_A
Tag Untagged Layer Default
Prefab Open Select Overrides
Transform
Position X -4.47101 Y -0.18561 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sprite Shape Renderer
Sprite Shape Controller (Script)
Profile Spriteshape_MineCartTrack_A
Edit Spline
Tangent Mode
Position X 14.61687 Y 2.847277
Height 1
Corner Automatic
Sprite Variant 0
Snapping
Spline
Detail High Quality
Is Open Ended
Adaptive UV ✓
Optimize Geometry ✓
Enable Tangents
Cache Geometry
Corner Threshold 30
Fill
Fill Tessellation (C# J
Stretch UV
Pixel Per Unit 100
Worldspace UV
Sprite-Lit-Default (Material)
Shader Universal Render Pipeline/2D/Spr
Mat_Scaffolding (Material)
Shader Universal Render Pipeline/2D/Spr

Spriteshape_Mine Cart Track_
Control Points
Use Sprite Borders ✓
Fill
Texture Sprite_Scaffolding
Offset 0
Angle Ranges
Angle Range (90)
Start 45 End -45 Order 4
Sprites
Sprite_MineCartTrack_A
Corners
Outer Top Left None (Sprite)
Outer Top Right None (Sprite)
Outer Bottom Left None (Sprite)
Outer Bottom Right None (Sprite)
Inner Top Left None (Sprite)
Inner Top Right None (Sprite)
Inner Bottom Left None (Sprite)
Inner Bottom Right None (Sprite)

Modifying a track's Bezier curves with Sprite Shape

Sprite Shape

Create organic 2D environments with a visual and intuitive workflow. Similar to vector drawing software, 2D Sprite Shape features **Sprite tiling** along a shape's outline, and automatically deforms and swaps sprites based on the outline's angle. You can attach a **Collider 2D** component to your Sprite Shape to enable Collider properties, and modify spline anchor points through the **Sprite Shape API** to make moving shapes at runtime that can interact with the player in your game. For example, you can create or deform terrain during gameplay.

Sprite technology

The **Sprite Editor** facilitates the setup of your art assets for 2D projects. Use the Sprite Editor to configure the **Pixels per Unit** (PPU) of sprites for precise roundtripping, make sprites tileable, slice them, define their collision, pivot points, and more.

Mixing 2D and 3D visuals

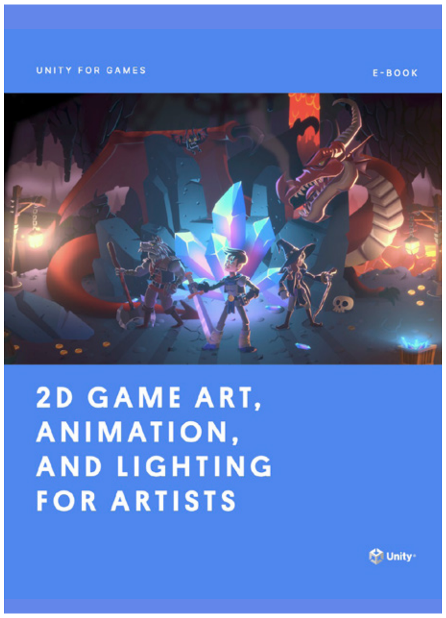
If your project uses the Built-in Render Pipeline or URP, mixing 2D and 3D elements in the same scene is relatively straightforward. 2D rendering uses the notion of **Sorting Layers** and **Sorting Groups** to define the order for rendering game elements.

Adding the Sorting Group component to a 3D GameObject allows for the integration of 3D and 2D objects in the same game. You can make them interact together by using a common **Physics system** (either 2D or 3D Physics, depending on the approach for your game), and also mix 2D and 3D Lighting systems through the **Camera Stacking** feature in URP.

Pixel art graphics

Pixel art games never go out of style. The **Pixel Perfect Camera** component allows you to set up any desired low resolution and low-fi settings to achieve your ideal aesthetic – whether that's an old school or pixelated look.

Establish a consistent pixel resolution through the Pixel Perfect component property. You can include features like upscaling, which keeps the pixel art crisp without interpolation, even if the sprites rotate or scale. This provides options for scaling and camera movement that follow the pixel grid indicated in the settings. You can either replicate retro aesthetic visuals with the tool or combine it with modern graphics like 2D Lights, shaders, or post-processing to achieve modern pixel art graphics.



2D game art, animation, and lighting for artists

2D games are making their mark. The evolution of hardware, graphics, and game development software makes it possible to create 2D games with real-time lights, high-resolution textures, and an almost unlimited sprite count.

Get Unity's most comprehensive 2D development guide, created for developers and artists who want to make a commercial 2D game.

Download the e-book

More resources

How to speed up 2D art workflows
Great tips for making 2D games
How to make retro 8-bit and 16-bit games
Setting the mood with 2D lights

APPENDIX 1: DIGITAL HUMANS IN UNITY

This digital character from *Enemies* features a new Strand-based Hair Solution and real-time hair rendering in Unity.

The evolution of cinematic storytelling in games is driving the demand for tools that can create lifelike human characters. Players expect modern AAA titles to include multifaceted characters that look realistic from any angle or distance. That's why Unity's award-winning demo team has been pushing the limits of what's possible with tools for creating digital humans in works such as *Enemies* and *The Heretic*.

The latest demo, *Enemies*, is built with advanced lighting and VFX on top of Unity's SRP rendering architecture. By using the latest installment of HDRP with its integrated post-processing stack, the team achieved a cinematic look that closely emulates how physical cameras work, all rendered in real-time. The human characters in *Enemies* and *The Heretic* required a high level of detail for closeups, a major challenge overcome by the demo team.

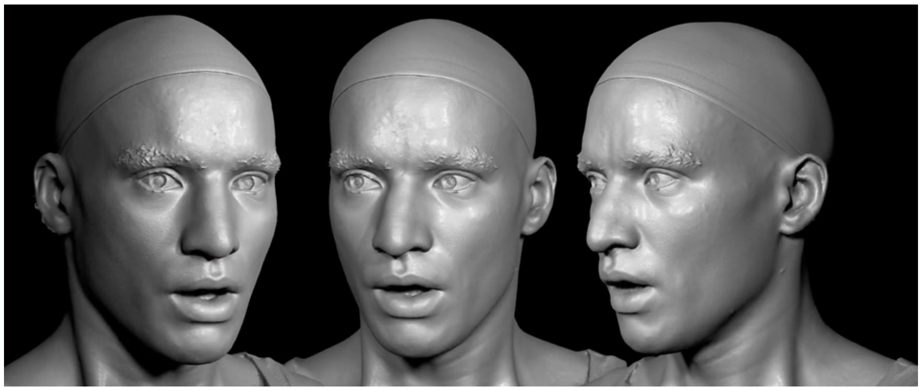
Gawain, the digital human from *The Heretic*, is available to use for educational and non-commercial projects via the Unity Asset Store.

Preparation

The Unity demo team needed to recreate all of the details of a real human being, including subtle facial movements. To achieve this, they opted for a hybrid of performance capture with details added in post-capture.

Data acquisition

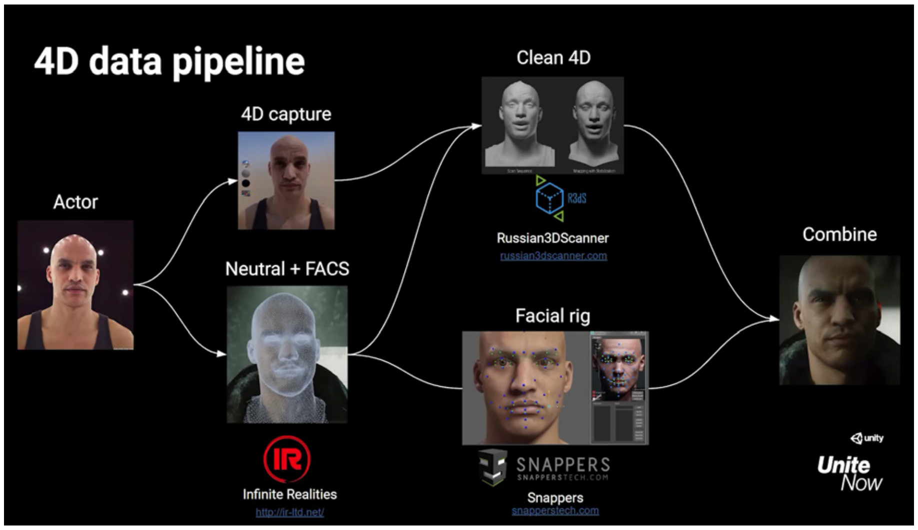
For the base facial motion capture in *Enemies* and *The Heretic*, Infinite-Realities studio reconstructed the actor's face many times per second using photogrammetry. This resulted in a high-resolution 3D scan corresponding to every frame of animation; "4D data" that helped reproduce the subtle nuances of the actor's facial movements through meshes, references, and textures.



The actor's 4D scan made for *The Heretic*

Data processing

A 4D scan captures nuances of the actor's performance, but this approach can be noisy and miss certain data points. To mitigate this, you will need to ensure proper data cleaning, mesh stabilization, and alignment with solutions like Wrap3D or a service provider.



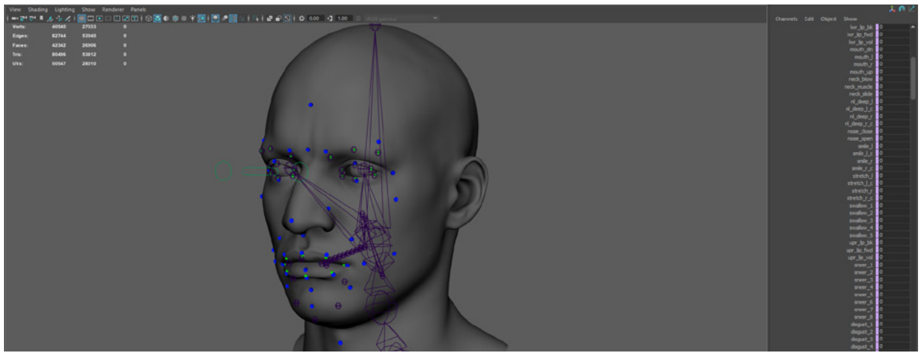
A breakdown of the 4D data facial capture pipeline used to create *The Heretic*

For *The Heretic*, SnappersTech studio created a traditional facial animation rig with over 300 blend shapes. The poses could be correlated to the cleaned-up 4D data. Once combined, additional details were layered onto the skin, resulting in a lifelike performance.

Unity setup

Since *The Heretic* and *Enemies* are real-time demos, the character rigs needed to be constructed in a way that is closer to a game project rather than a film production. The number of joints had to be optimal.

In *Enemies*, more specifically, blood flow simulation, wrinkle maps, and the new hair technology, alongside the lighting features of HDRP, made a new level of realism possible. Note that the new Hair Solution will be made publicly available soon in a GitHub repository.

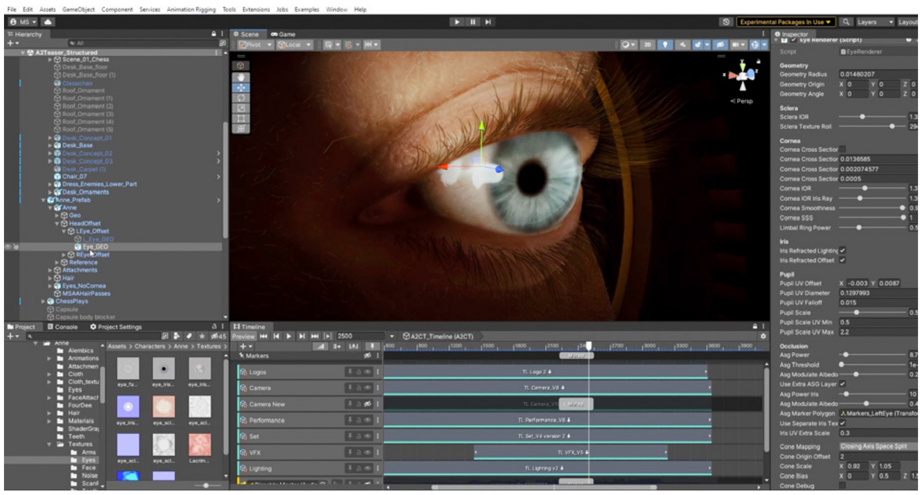


The facial animation rig used over 300 blend shapes to create Gawain in *The Heretic*.

Eyes

To recreate believable eyes in *The Heretic*, the team used HDRP's Eye Shader and added features such as the ability to reflect light and additional controls for each component of the eye.

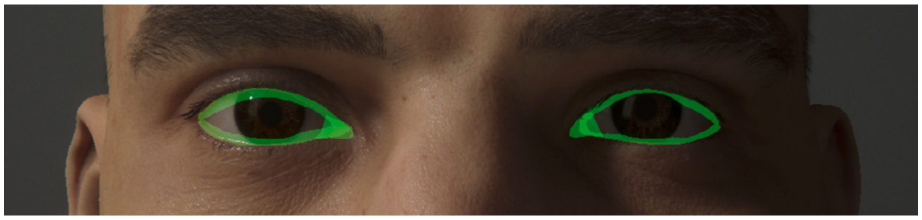
For *Enemies*, the team improved upon this shader with caustics that form with the curvature of the cornea and an iris lit realistically. The solution is already available in this GitHub repository.



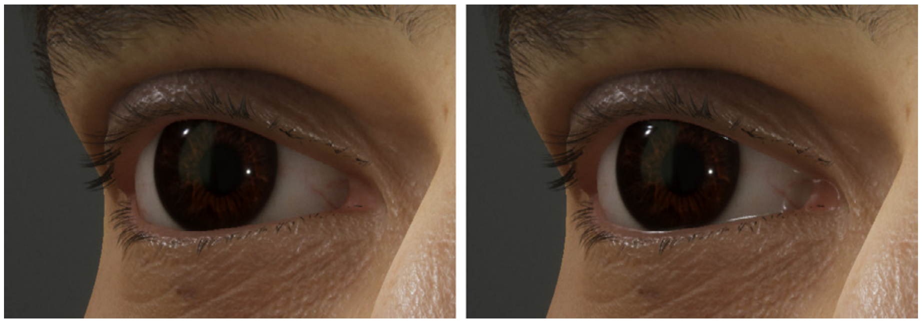
```
Script                      EyeRenderer
Geometry
Geometry Radius             0.01460207
Geometry Origin             X 0    Y 0    Z 0
Geometry Angle              X 0    Y 0    Z 0
Sclera
Sclera IOR                  1.3
Sclera Texture Roll         29
Cornea
Cornea Cross Sectior        ☐
Cornea Cross Sectior        0.0136585
Cornea Cross Sectior        0.002074577
Cornea Cross Sectior        0.0005
Cornea IOR                  1.3
Cornea IOR Iris Ray         1.3
Cornea Smoothness           0.9
Cornea SSS                  1
Limbal Ring Power           0.5
Iris
Iris Refracted Lighting     ✓
Iris Refracted Offset       ✓
Pupil
Pupil UV Offset             X -0.003    Y 0.0087
Pupil UV Diameter           0.1297993
Pupil UV Falloff            0.015
Pupil Scale                 0.5
Pupil Scale UV Min          0.5
Pupil Scale UV Max          2.2
Occlusion
Asg Power                   8.7
Asg Threshold               1e-
Asg Modulate Albedo         0.2
Use Extra ASG Layer         ✓
Asg Power Iris              10
Asg Modulate Albedo         0.4
Asg Marker Polygon          Markers_LeftEye (Transfo
Use Separate Iris Tex       ✓
Iris UV Extra Scale         0.3
Cone Mapping                Closing Axis Space Split
Cone Origin Offset          2
Cone Scale                  X 0.92    Y 1.05
Cone Bias                   X 0    Y 0.5    Z 1.5
Cone Debug                  ☐
```

Enemies required a custom version of HDRP's Eye Shader.

The eyes received special treatment so that the cornea's shine could render separately from the diffuse color of the eyeball (the iris and sclera). This process also allowed the iris to reflect light more accurately.



In *The Heretic*, a separate tearline mesh controls the wetness of the character's eyes, where the eyeballs meet the eyelids.

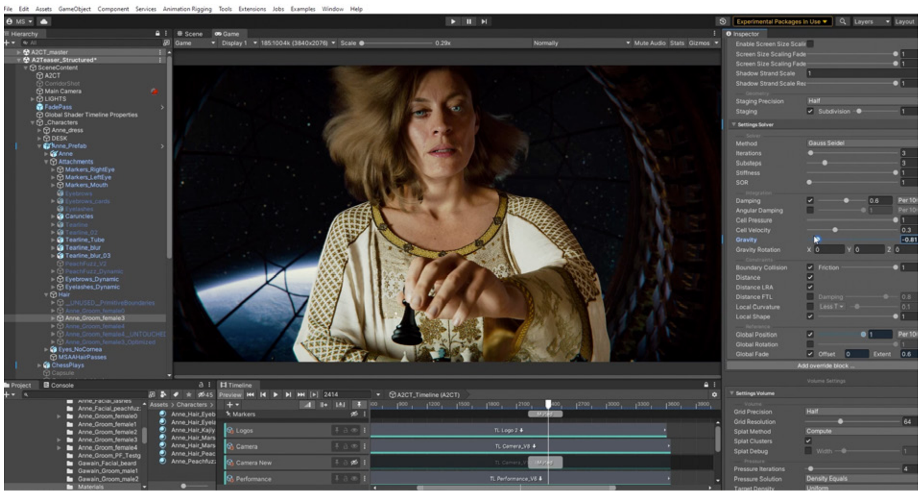


Notice how the tearline (right) captures the wetness between the lid and eye.

Hair

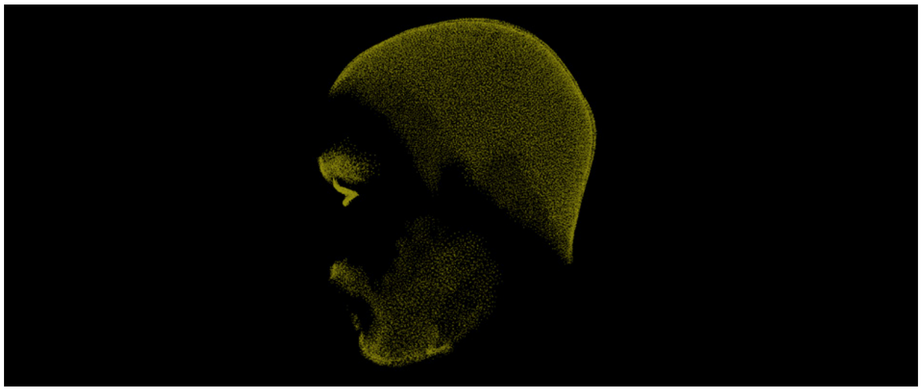
One of the biggest challenges for the team was to produce detailed hair that could be performant in real-time. In *Enemies*, the main character's hair is long and flowing. Thanks to built-in physics, it appears as real hair does, including the shine from realistic lighting.

Of course, there are many different types of hair, so the team designed features to reproduce straight, curly, and short hairstyles. They created many different grooms with XGen in Maya, and then imported those designs back into Unity. The tech was also applied to other facial hair, such as eyelashes, eyebrows, and skin. This technology, which will soon be released to all users, aims to be the best-in-class solution for hair today.



The Hair Solution offers a complete set of features for realistic hair simulation. This screenshot shows the hair of the main character in *Enemies* reacting to gravity.

Meanwhile, Gawain's character design purposely omitted long hair – but you might be surprised to learn just how much short hair is still present. The demo team created a skin attachment system to hold the brows, stubble, and lashes in place. No matter how the face was animated, tens of thousands of little hairs stuck to the 4D capture.



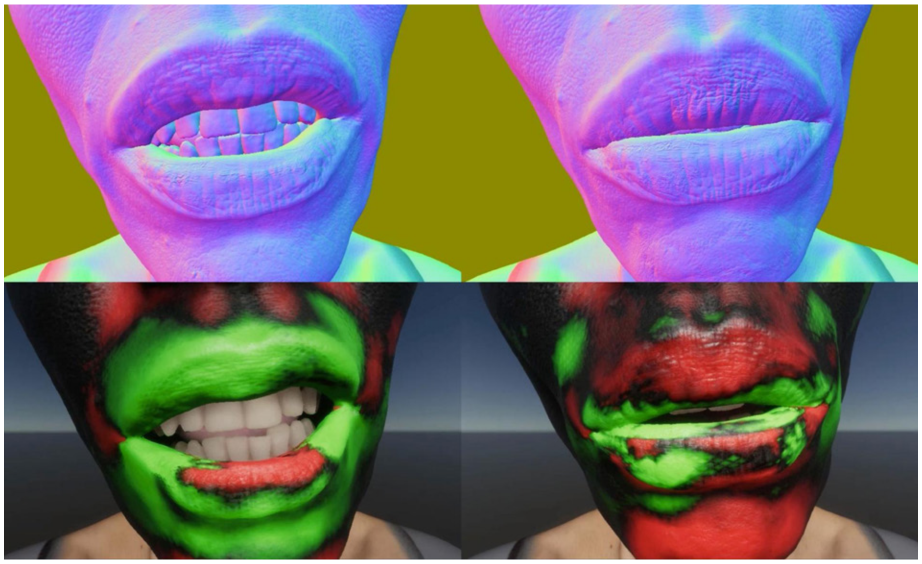
The skin attachment system held tens of thousands of small hairs in place for Gawain in *The Heretic*.

Skin

HDRP's existing Lit and Layered Lit shaders provide a solid foundation for digital human shading features. In the case of the demos, HDRP's existing support for Subsurface Scattering was harnessed to simulate the way that light penetrates and moves within an area under organic surfaces for both the skin and teeth.

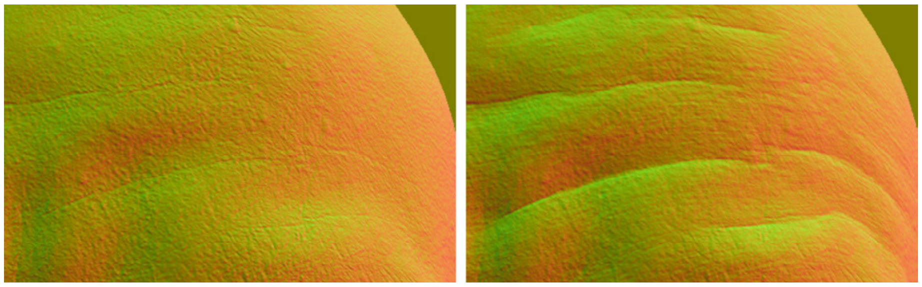
At the same time, the *Enemies* demo features the Skin Tension solver, new technology that facilitates blending between several texture maps based on where the skin stretches or compresses. Depending on the tension of the skin, the material used by the solution employs a version of the textures with more or fewer wrinkles. It works with the whole head skin or any model, whether it's been rigged or animated with volumetric capture. The Tension Scale value then determines how much the compressed or stretched areas of skin will react to pose changes.

The Tension shader tool was created along with Shader Graph, making it accessible to creators with minimal coding experience. Both the hair and skin technology from *Enemies* will soon be available to all users. See the *Enemies* page for updates.



Taken from *Enemies*: The two images on top show the different normal map textures used in the lip area. The two bottom images, taken in Debugging mode, show the stretched skin areas in green.

In *The Heretic*, the team used pose-driven attributes from the Snappers' rig to add greater detail to the skin. For instance, some of Gawain's facial expressions, like squinting or frowning, produced wrinkle maps that defined extra creases on his face and added a degree of expression on top of the 4D capture.



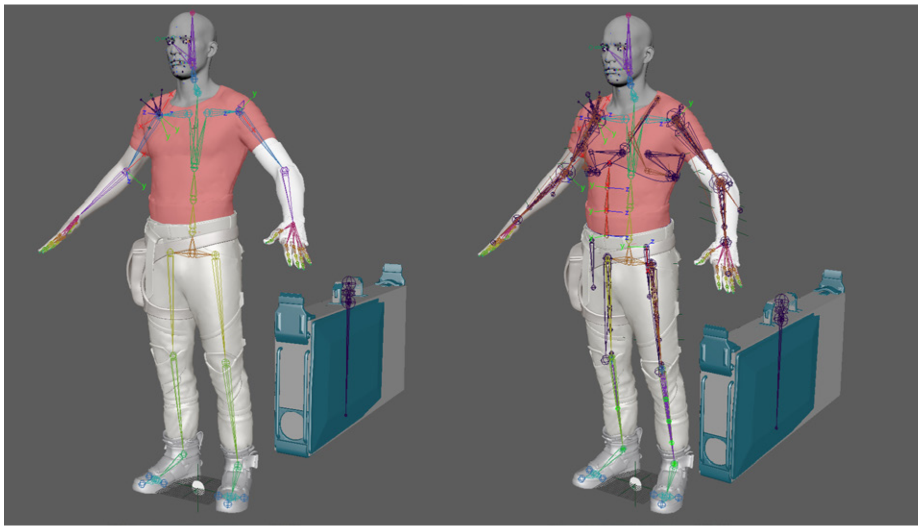
Pose-driven wrinkles added on top of the underlying 4D captured skin for Gawain in *The Heretic*

The base layer consisted of the main skeleton that was responsible for transferring the body motion between Autodesk MotionBuilder® and Maya. There was a low-fidelity version of Gawain built specifically for MotionBuilder and used for all motion capture cleanup.

The second layer served as the deformation layer. This more detailed version of the character drove the actual geometry and included joints for better deformation. In particular, this layer featured joints for twisting the arm along its length, fan-joints to soften the deformation around the shoulder area, and a double-knee setup that solved compression artifacts around the knees. The Snappers' facial rig was referenced in the scene before exporting to Unity.

Body animation

Most of the animations for Gawain were produced at a motion capture facility. Despite this fairly standardized process, there was one big challenge: syncing the character's body motions with the prerecorded 4D facial performance.



Compare the MotionBuilder version of the rig (left) with the Maya deformation rig (right).

This sync required many takes to get the body motions to fit the facial performance. After processing the animation data, applying slight adjustments, and retiming the movement in MotionBuilder, the finished animations resulted in an almost seamless body-face performance.

Shading the human face is challenging.

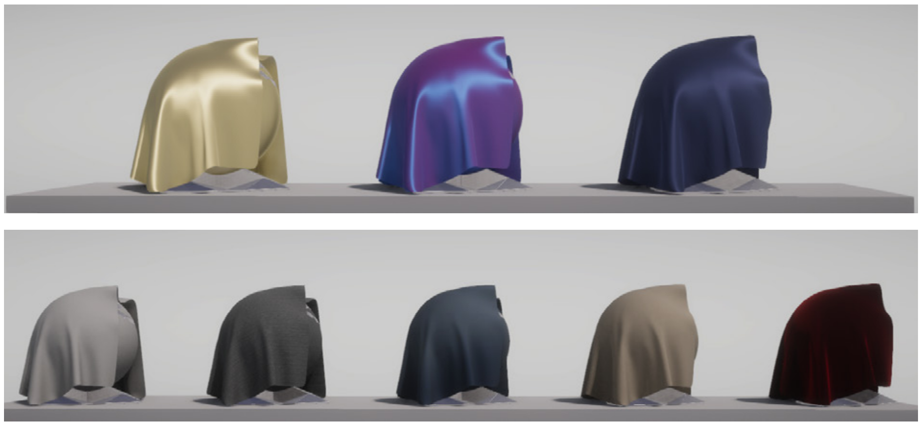
Read this behind-the-scenes blog post for more information on the 4D and motion capture process, and watch the Unite Now session, Meet the Devs: *The Heretic* short film, for further details regarding the pipeline.

Here are just a few numbers that reflect the massive effort involved in creating the main character for *The Heretic*:

- Vertices: 28k for the head, 57k for the body, and 43k for the jacket
- Maya skeleton rig: Approximately 360 joints
- Base texture data: 4k maps for albedo, normal, cavity, thickness, etc.
- Pose-driven texture data: 48 activation masks, plus 16 × 3 additional 4k maps for albedo, normal, and cavity
- Facial blend shapes: 318

Props and clothing

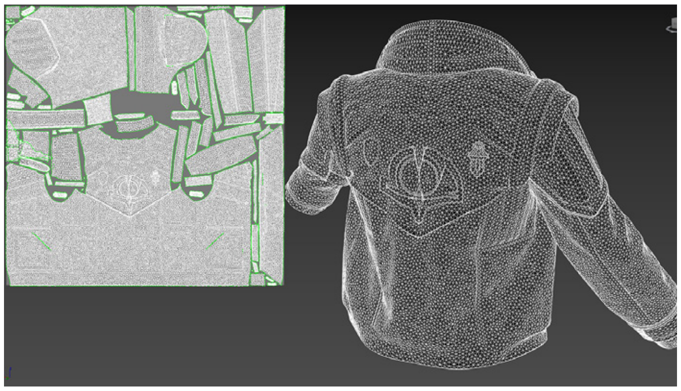
Unity can help you achieve photorealistic results through physically based rendering of your characters' props and clothing. HDRP includes shaders for most of your material and surfacing needs. Materials such as stone, metal, and fabric use values aligned with their real-world equivalents.


Cotton and silk shaders can meet your needs for clothing or fabric.

Though your characters won't be defined by their clothes, it never hurts to make them look fresh. Just as much of *The Heretic* was produced outside of Unity as it was in-engine. For example, artists simulated Gawain's jacket in Marvelous Designer.

Then, they imported the Alembic file for real-time playback later. Just as Boston rips apart the double doors in the basement corridor, the animated geometry comes from a baked simulation.

Render test of fabric materials for Gawain's jacket

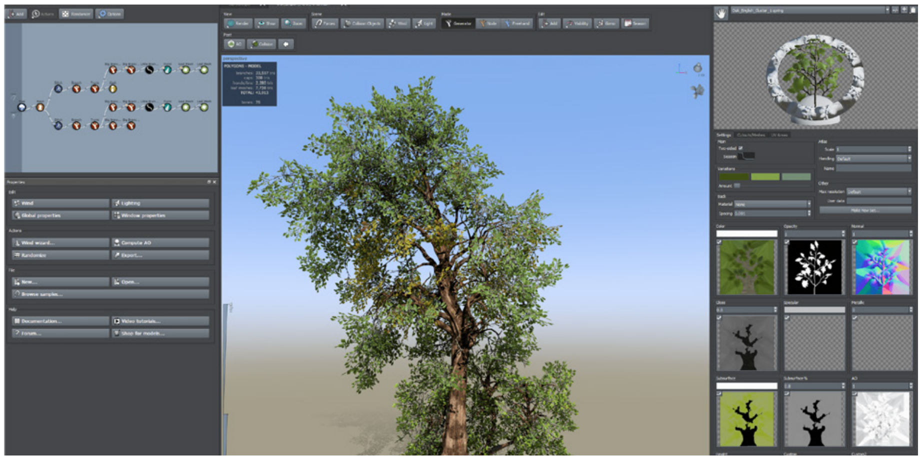

Gawain's shirt and jacket were both assembled and simulated in Marvelous Designer, then textured in Substance Designer.

APPENDIX 2: MORE ART AND UPCOMING TOOLS

Unity aims to empower artists everywhere by bringing their creative visions to life. The following technology combines professional artistry tooling and efficient team workflows in diverse and demanding productions. While these tools are engine-agnostic, they might require their own licensing agreements or the installation of standalone apps.

SpeedTree

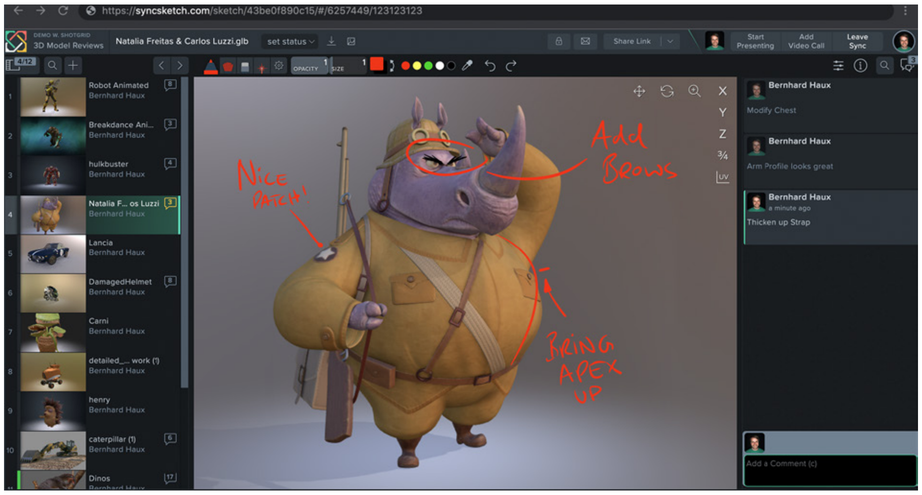
Trees and foliage are often essential parts of large, complex, and lifelike landscapes. SpeedTree provides visual creators like you with powerful procedural and hand-modeling tools, in addition to the SpeedTree Library's comprehensive array of trees and foliage. This way, you can start adding natural-looking assets to your innovative experiences.



SpeedTree is a solution used for many commercial games and films: In Unity, use it as part of the Terrain or SRP workflows.

SyncSketch

Designed for smooth collaboration across and among art teams, SyncSketch helps artists, animators, VFX professionals, students, and other creators to communicate in real-time.



Bernhard Haux
Modify Chest

Bernhard Haux
Arm Profile looks great

Bernhard Haux
a minute ago
Thicken up Strap

SyncSketch enables rapid visual communication with just a link.

An example of what's possible with the tools from Ziva

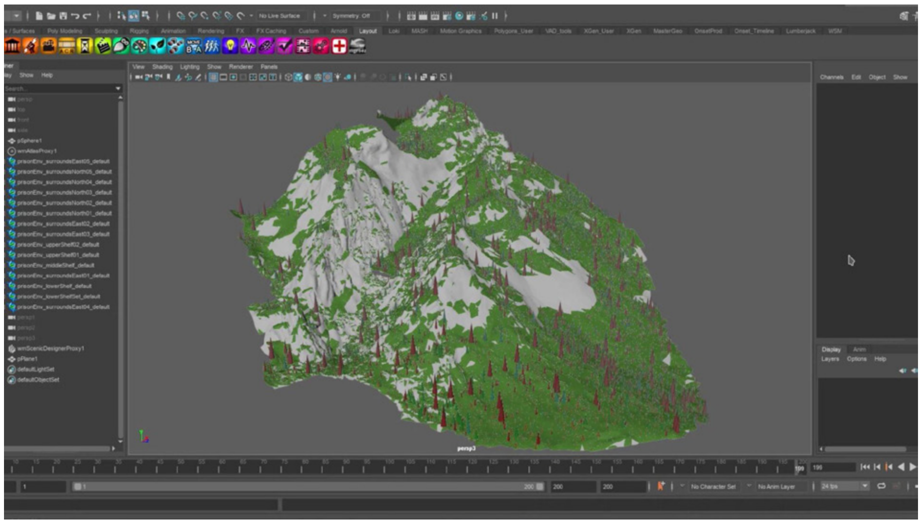
SyncSketch supports 2D imagery, video, and 3D models. Users can conveniently rotate static or animated 3D assets, change the lighting, add notes, and share feedback in real-time or asynchronously with just a link. You can review projects from your phone or tablet, create summaries in PDF, and easily share them. There's even a whiteboard feature to brainstorm on the fly. Read more about it here.

Upcoming solutions

Coming soon, the tools from Ziva Dynamics will set a new standard for high-quality human and creature characters, whether realistic or stylized, in games or on film. Ziva technology makes it possible to digitally replicate and couple the physics and materiality of soft tissue, such as muscle, fat, and skin, for artists to create the most lifelike CGI characters to date.

In 2021, Unity acquired Weta Digital, along with its artist tools, core pipeline, intellectual property, and engineering talent. The Academy Award-winning VFX service teams from Weta will continue as a standalone entity known as Weta FX – Unity's soon-to-be largest customer in the media and entertainment space.

By combining Weta Digital's industry-leading VFX tools and technical talent with Unity's development and real-time platform, we aim to deliver solutions that will unlock the full potential of the metaverse. Get a glimpse of what's to come in this blog.

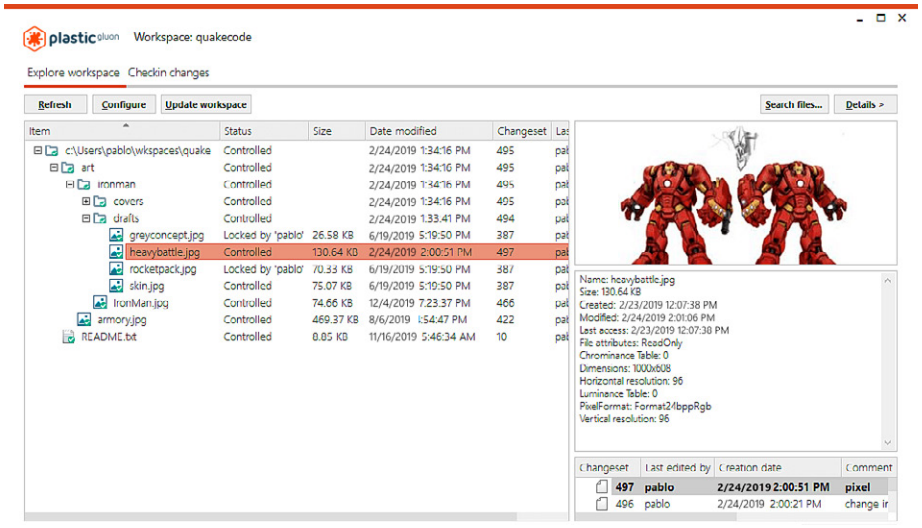


Totara from Weta, a procedural growth and simulation system for vegetation and biomes

Source control

Unity offers a number of version control solutions, such as Perforce, Plastic SCM, and Git, including Git LFS (Large File Support). When Plastic SCM first joined Unity in 2020, its version control tools were integrated into the Editor.

Plastic is designed for use case-driven game development. It excels at handling large repos and binary files, and lets you download only the files that you're working on, rather than the entire project build. Plastic SCM also offers Gluon, a version control solution tailored to programmers and artists alike, along with other version control tools, hosting, and a GUI client.



Workspace: quakecode

Item	Status	Size	Date modified	Changeset	Las
c:\Users\pablo\wkspaces\quake	Controlled		2/24/2019 1:34:16 PM	495	pal
art	Controlled		2/24/2019 1:34:16 PM	495	pal
ironman	Controlled		2/24/2019 1:34:16 PM	495	pal
covers	Controlled		2/24/2019 1:34:16 PM	495	pal
drafts	Controlled		2/24/2019 1:33:41 PM	494	pal
greyconcept.jpg	Locked by 'pablo'	26.58 KB	6/19/2019 5:19:50 PM	387	pal
heavybattle.jpg	Controlled	130.64 KB	2/24/2019 2:00:51 PM	497	pal
rocketpack.jpg	Locked by 'pablo'	70.33 KB	6/19/2019 5:19:50 PM	387	pal
skin.jpg	Controlled	75.07 KB	6/19/2019 5:19:50 PM	387	pal
IronMan.jpg	Controlled	74.66 KB	12/4/2019 7:23:37 PM	466	pal
armory.jpg	Controlled	469.37 KB	8/6/2019 1:54:47 PM	422	pal
README.txt	Controlled	8.85 KB	11/16/2019 5:46:34 AM	10	pal

Name: heavybattle.jpg
Size: 130.64 KB
Created: 2/23/2019 12:07:38 PM
Modified: 2/24/2019 2:01:06 PM
Last access: 2/23/2019 12:07:38 PM
File attributes: ReadOnly
Chrominance Table: 0
Dimensions: 1000x608
Horizontal resolution: 96
Luminance Table: 0
PixelFormat: Format24bppRgb
Vertical resolution: 96

Changeset	Last edited by	Creation date	Comment
497	**pablo**	**2/24/2019 2:00:51 PM**	**pixel**
496	pablo	2/24/2019 2:00:21 PM	change ir

Plastic's workflow is designed for both technical and non-technical creators to preview files, check in changes, and review their history.

UNITY FOR GAMES
E-BOOK
VERSION CONTROL AND PROJECT ORGANIZATION BEST PRACTICE GUIDE
2020 LTS EDITION
Unity

Version control and project organization best practices

This e-book, made for both technical and non-technical creators, explains how to set up version control and plan for productive teamwork. The guide covers comparisons between centralized and distributed systems, solutions like Git, Perforce, and Plastic SCM, and also demonstrates how to effectively organize your Unity project.

Download the e-book

More resources

SpeedTree YouTube channel
Ziva Dynamics website
Weta Digital website
Plastic SCM

Unity for artists

Find additional art and technical tips on the **Unity blog**, or start by searching the #unitytips hashtag on **Unity Learn** and our **community forums**. The **Unity Developer Tools** microsite also provides some key resources for technical artists – from documentation to our latest roadmap and release details.

Looking for even more support? Unity Professional Training gives technical artists and creative teams the skills they need to achieve their project goals. We offer an extensive training catalog, designed for professionals in any industry, at any skill level, working in Unity. All materials are created by our experienced Instructional Designers, in partnership with engineering and product teams. This means that you always receive the most up-to-date training on the latest Unity tech.

Learn more about what kind of training is available for you and your team.

Thank you to all the experts who contributed to this guide. And to our readers: Good luck on your creative projects ahead.

Unity®

unity.com