UNITY QUICK START GUIDE

E-BOOK

Unity for technical artists: Key workflows and toolsets

unity

Contents

Unity for technical artists

Read this guide for an introduction to the authoring tools and APIs available in Unity, and discover how they can help you set up an efficient content creation pipeline that supports everyone on your team to deliver their best result.

Introduction

This guide provides 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.

An HDRP urban environment

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.

That's why this guide was assembled with input from some of Unity's most experienced TAs from different R&D teams and Unity's Demo team – the creators of *The Heretic*, *Book of the Dead*, and *Adam*, among other high-end demos. Our TAs have years of experience working in large, team-based game productions, with deep knowledge of lighting, animation, cinematics, visual effects, and shader and graphics programming.

Read this guide for an introduction to the authoring tools and APIs available in Unity, and discover how they can help you set up an efficient content creation pipeline that supports everyone on your team to deliver their best result.

Scripting in Unity

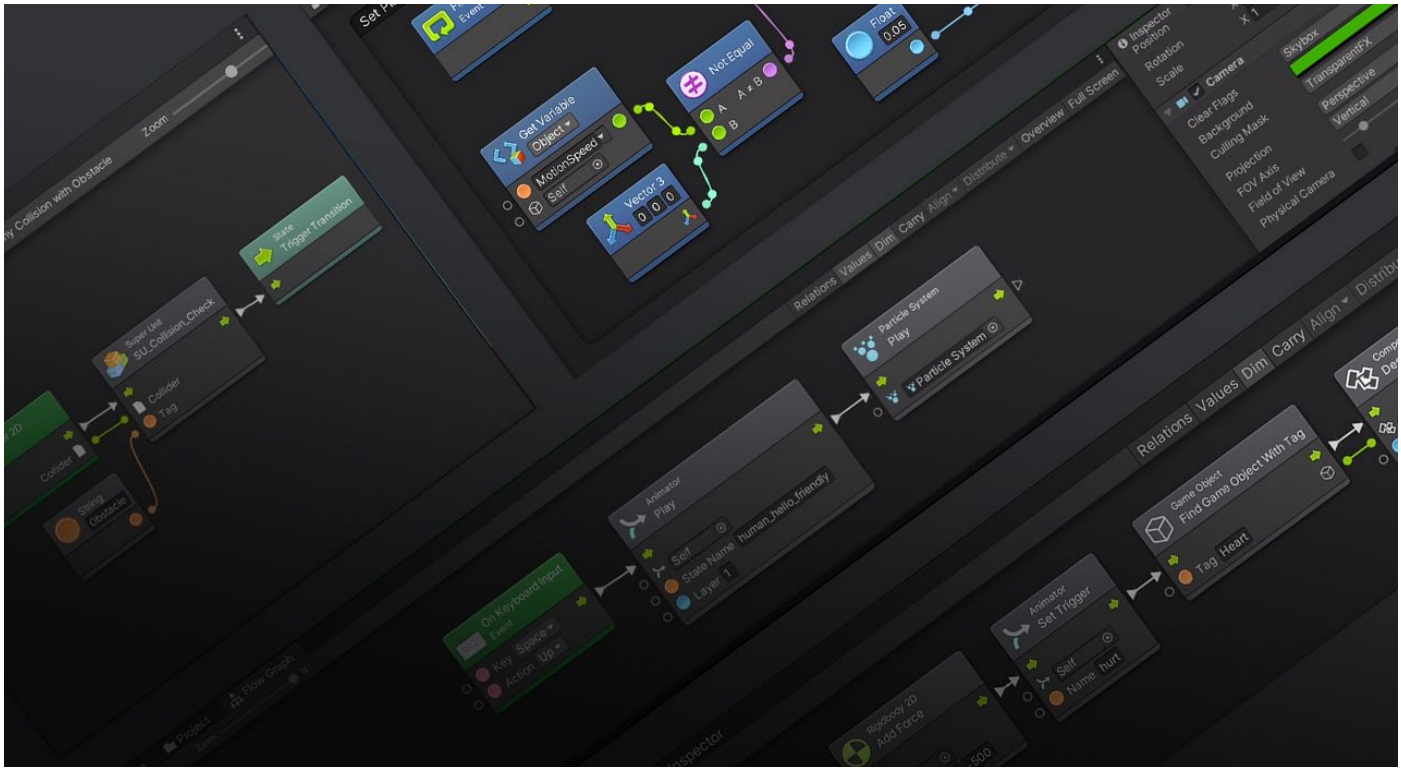
Experienced Unity programmers can skip this section. If you have no coding experience and would like to create scripts in Unity, we recommend that you read this short introduction and peruse the list of beginner resources below.

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: characters, lights, special effects, props, and so on. GameObjects can't do anything on their own. To become animate, you need to give a GameObject its properties by adding components.

Components define and control the behavior of GameObjects that 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.



Unity's Visual Scripting solution

Components have any number of editable properties, or variables, that are tweaked via the Inspector window in the Unity Editor and/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 three recommended learning paths:

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

— Creator Kit for coding: Complete this kit in a few hours to explore the basics of C# code for Unity in the context of an action RPG video game.

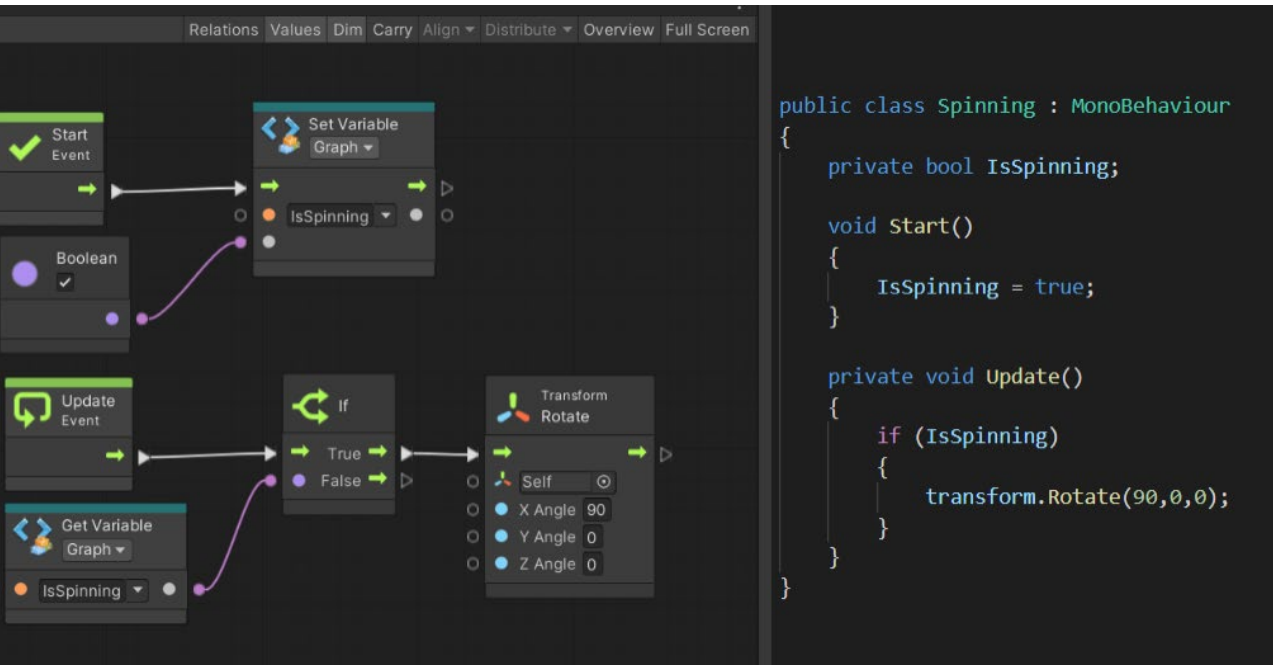
— Create with Code: Take on a comprehensive course that provides over 37 hours of instruction.

Visual scripting in Unity

Available in Unity 2021.1 and later, Unity's visual scripting system lets you create the logic for your Unity projects without writing actual code. Visual scripting has visual, unit-based graphs that both programmers and non-programmers use to design final logic or quickly create prototypes. It makes it easier to learn and understand scripting concepts, and even watch scripts run in real-time.

There are four basic concepts in visual scripting that are used when building games:

Type is an attribute of data that tells the compiler how to use the data. In scripting, everything is an object: numbers, pieces of text, vectors, and Unity components.



Unity's visual scripting solution

Variables are containers that store values and data. Each variable has a name, type, and value. The value inside a variable can change during runtime.

Graphs are visual representations of logic at the core of visual scripting.

Groups are boxes that surround units, created to organize the graphs.

Visual scripting provides programmers and TAs with a solution to better collaborate with artists and designers. They can use visual scripting to create extensions, templates, and tools for other Unity users who do not create via scripting, so that everyone on a project can work and contribute in a streamlined process, regardless of whether they know C#.

More resources

Introduction to visual scripting

Visual scripting for programmers

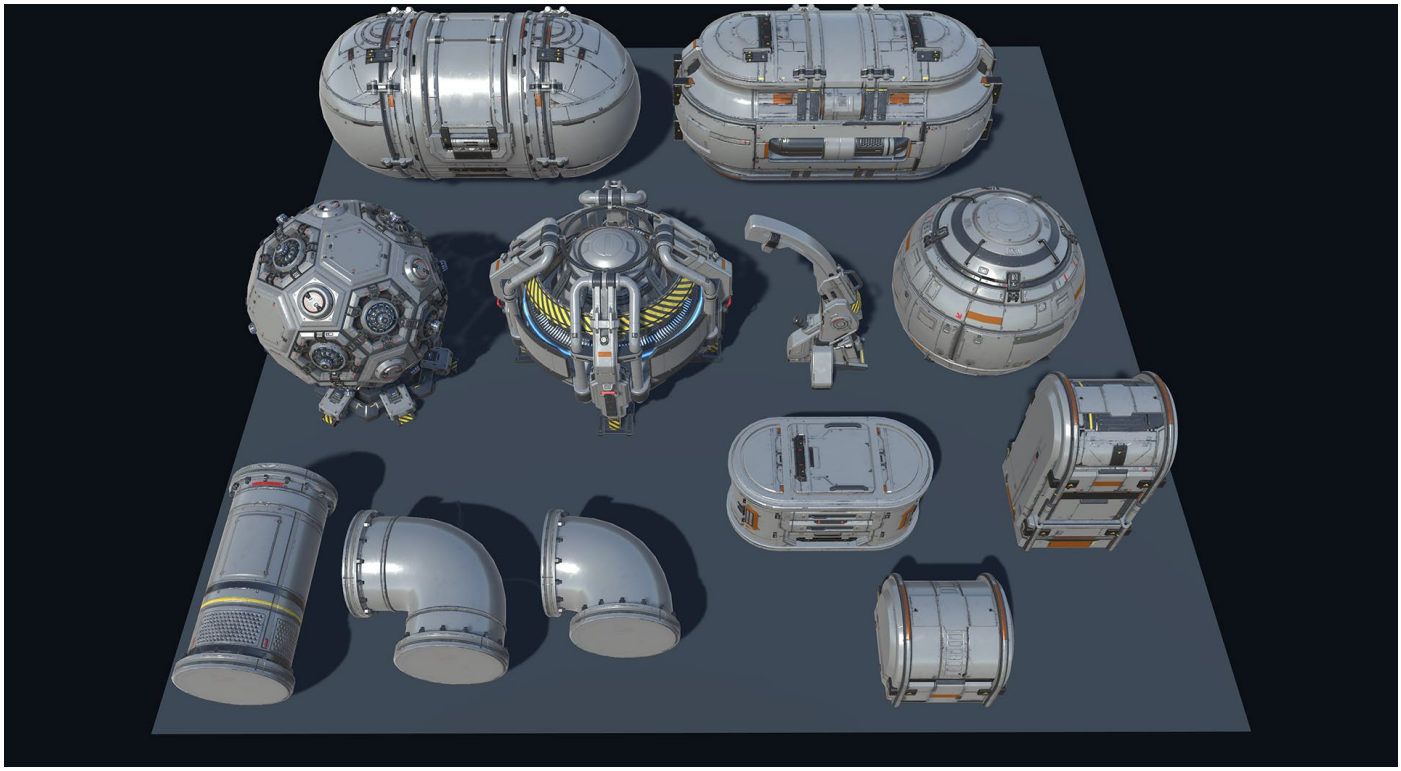
Prefabs

Unity's Prefab system allows you to create, configure, and store a GameObject, complete with all its components, property values, and child GameObjects, as a reusable Asset. The Prefab Asset acts as a template from which you can create new Prefab instances in the Scene. 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.

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.

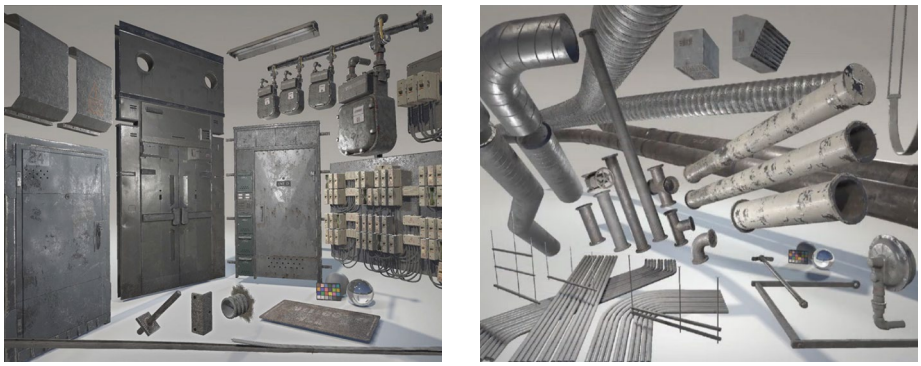


Examples of Prefabs

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

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 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 transformers 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

Roundtripping

The FBX Exporter package is one of the products developed from the ongoing collaboration between Unity and Autodesk. It enables a smooth roundtrip workflow for the easy transfer of 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 your work in Unity.

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

Additional tools that make the roundtrip workflow more efficient include Presets to automatically apply preferred settings when you import assets. This is useful when you need to import hundreds of assets for your project. In fact, you can create your own importer in C# with Scripted Importers to work with files that are not natively supported in Unity.

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

Working with other DCC tools

If you work with DCC software such as Blender or SketchUp, the simplest way to get started with your 3D assets in Unity is to save your .Blend files inside the Unity project's Asset folder. We recommend that everyone on your team uses the same version of your selected DCC software. Unity will import the Mesh with all nodes in their saved position, rotation, and scale. Pivot points and names are also imported, as are vertices, polygons, triangles, UVs, and normals, in addition to 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. Despite the easy round-tripping with Blender and other DCC software, we recommend that you use .fbx files to maximize compatibility with Unity features like Cloud Build.

If you prototype or greybox 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

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 on a screen. At a high level, these operations are culling, rendering, and post-processing.

Unity provides three render pipelines for various purposes: the Built-in Render Pipeline, and two Scriptable Render Pipelines (SRPs) – the Universal Render Pipeline (URP) 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 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 Build-in Render Pipeline and URP. It is a general-purpose rendering path that makes it expensive to render real-time lights. There is also a limit to the number of additional lights that you can add per object. If your project does not use a large amount of real-time lights, or if lighting fidelity is not crucial to its success, then this rendering path might be a good choice for your project.

Deferred shading

Deferred shading is the default path in the HDRP – and the rendering path with the most lighting and shadow fidelity. Deferred shading requires GPU support, and has some limitations. If your project demands both a large number of real-time lights and a high level of lighting fidelity, this rendering path might be the right choice for your project.

Universal Render Pipeline

The URP provides optimized graphics performance on a broad range of platforms including VR. This is made possible due to some trade-offs around lighting and shading. Certain restrictions are applied, and features that aren't supported on lower-end devices are disabled. This allows developers to worry less about how to optimize their work and focus more on developing projects that will reach a larger audience. URP will soon be the default render pipeline in Unity.

The URP renders in a light loop, which makes 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 the URP will result in fewer draw calls. It's also supported by the Shader Graph tool, which provides additional benefits for shader authoring workflow.

To start using the URP, open a new URP project from the Unity Hub or install the URP package.

An image from a URP demo with post-processing effects

High Definition Render Pipeline

The HDRP is a high-fidelity SRP built by Unity to target modern platforms that are compatible with compute shaders. It aims to provide developers with tools to achieve high-definition visuals with ease. Use HDRP for AAA quality games, automotive demos, architectural applications, and anything that requires high-fidelity graphics.

An HDRP environment

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 the lighting within the Scene. It uses a hybrid of deferred and forward rendering paths, along with tile and cluster renderers, so that the lighting scales better than it would with Unity's Built-in Render Pipeline.

HDRP is designed to be accessible to smaller development teams that aim to achieve AAA graphics on high-end PCs and consoles. If you want the highest-quality visual fidelity for your project, but performance isn't your first consideration, then HDRP might be the ideal choice for you.

To start using the HDRP, open a new HDRP project from the Unity Hub or install the HDRP package.

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 the SRP Core package, which is 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, and 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 tell Unity to execute them before the low-level graphics architecture sends instructions to the graphics API.

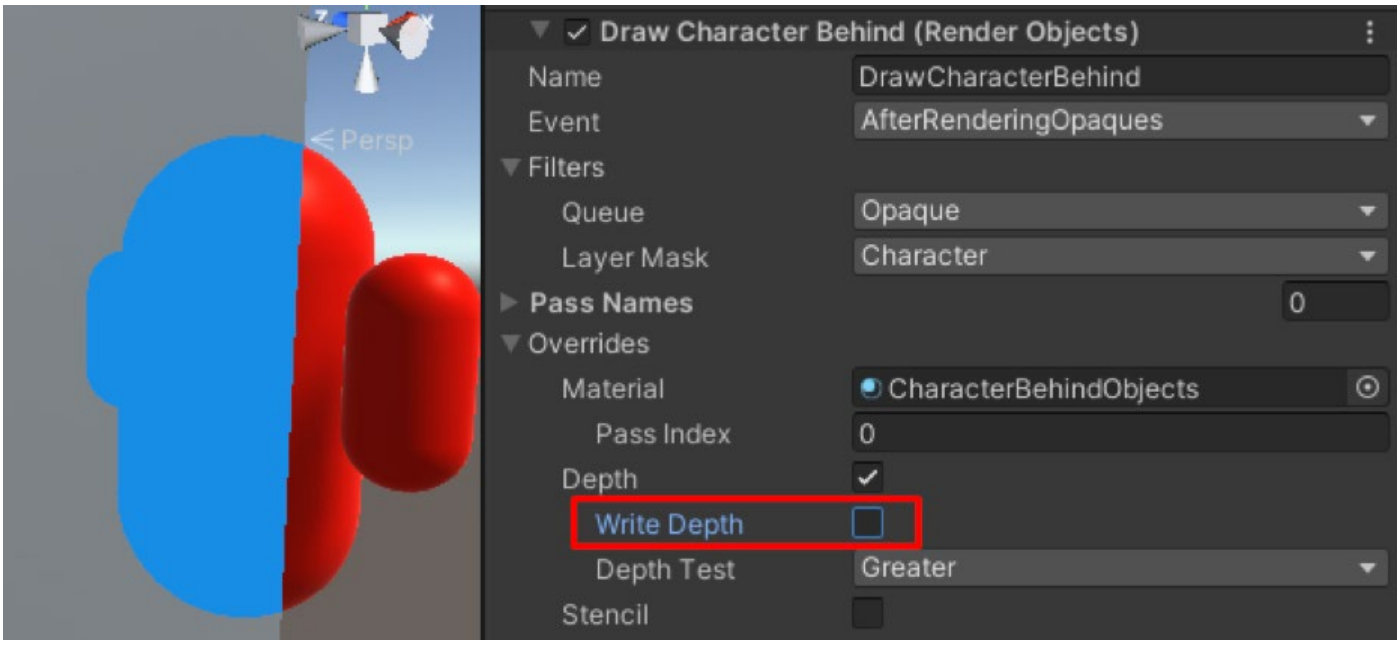
Unity documentation provides instructions for creating custom rendering, including custom versions of URP and HDRP (see next section). SRP source code is also available on GitHub.

An HDRP environment

Custom render passes in URP

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

In Unity 2020 LTS and newer versions, the RenderObjects Renderer for the URP provides you access to custom render passes based on layers that filter the GameObjects to render. This allows TAs to create advanced visual effects or gameplay mechanics like rendering the silhouette of a character hidden from the camera behind a wall.



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

In HDRP, custom passes are done in a slightly different way; you can find more information about it in the documentation and these sample projects.

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 are the most common type. They perform calculations to determine the color of pixels on the screen. See surface shaders, for example.

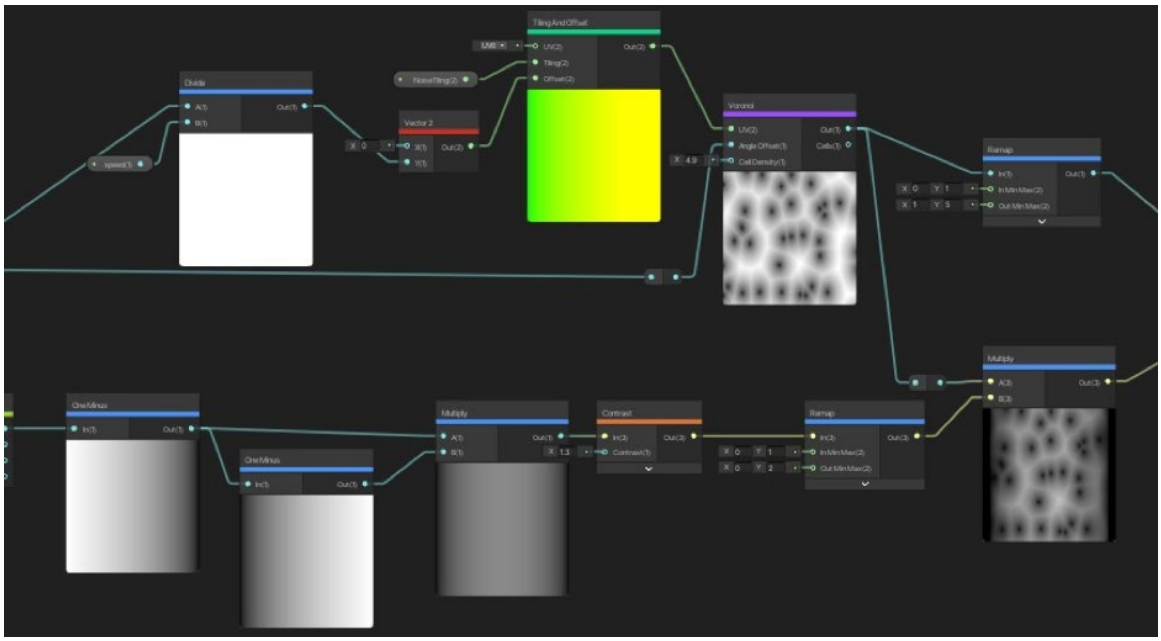
— Compute shaders perform calculations on the GPU, outside of the regular graphics pipeline.

— Ray-tracing shaders are mainly used to generate lighting.

Shader visual authoring: Shader Graph

Unity shaders are written in a Unity-specific language called ShaderLab, but it is also possible to create shaders visually with Shader Graph. Instead of writing code, saving, compiling, and testing in the 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; this includes their mathematical functions, procedural patterns, and more. In addition, the Shader Graph system is extensible, which allows programmers to develop custom Shader Graph nodes.



Node-based shader creation in Shader Graph

In Unity 2020 LTS and newer versions, you can create shaders that work for both URP and HDRP.

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.

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 efficiently requires an 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 used 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 allows artists more freedom to define the look of their surfaces.

More resources

Shader Graph Master Stack

Experimenting with Shader Graph: Doing more with less

Normal map compositing using the surface gradient framework in Shader Graph

Lighting in Unity

Global Illumination

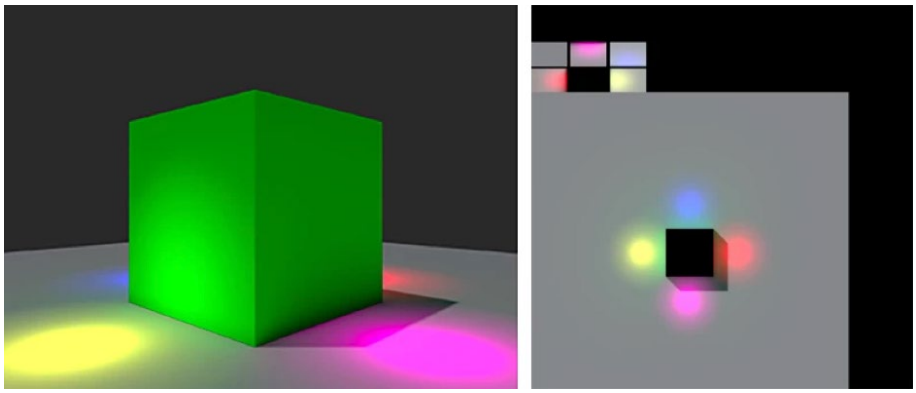
Lighting in modern games makes great use of Global Illumination, or GI. GI covers a range of techniques and mathematical models that attempt to simulate the complex behavior of light as it bounces and interacts with the world. Simulating Global Illumination accurately is challenging and can be computationally 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.

Global Illumination in Unity

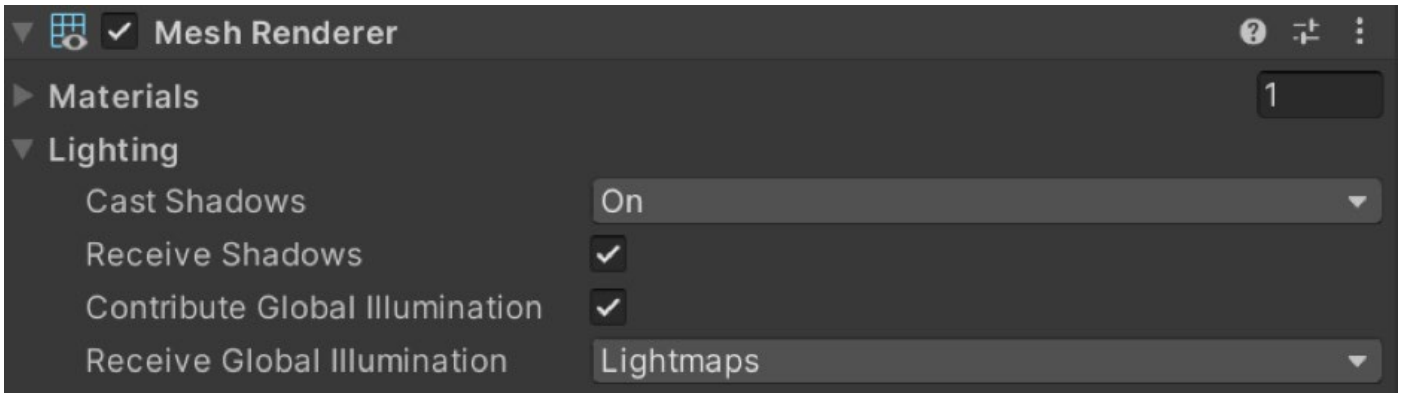
Baked Global Illumination

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



In Unity, precomputed lighting is generated automatically in the background or initiated manually. Either way, it's possible to continue working in the Editor while these processes run behind the scenes.

The GI system will only consider objects that have the property Contribute Global Illumination selected in their Mesh Renderer component. And to be lit by a Lightmap, the Lightmaps options must be selected for the property Receive Global Illumination to be in effect. Otherwise, the object will be lit by Light Probes.



If a GameObject is set to receive GI from Light Probes, it needs to have a Light Probe Group component attached. 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 fast 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. This makes it robust and allows you to bake selected portions of lightmaps, for faster testing and iteration on your Scene.

The Progressive Lightmnapper 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, and high-end mobile devices when used with small Scenes and 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, yet it allows for 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 supports Enlighten. HDRP and URP will get support for Enlighten with Unity 2021.2 and beyond.

Ray-traced Global Illumination

Another form of real-time GI can be attained through ray tracing. Currently only HDRP offers this capability, which also 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 high-end PCs and consoles that support hardware-accelerated ray tracing.

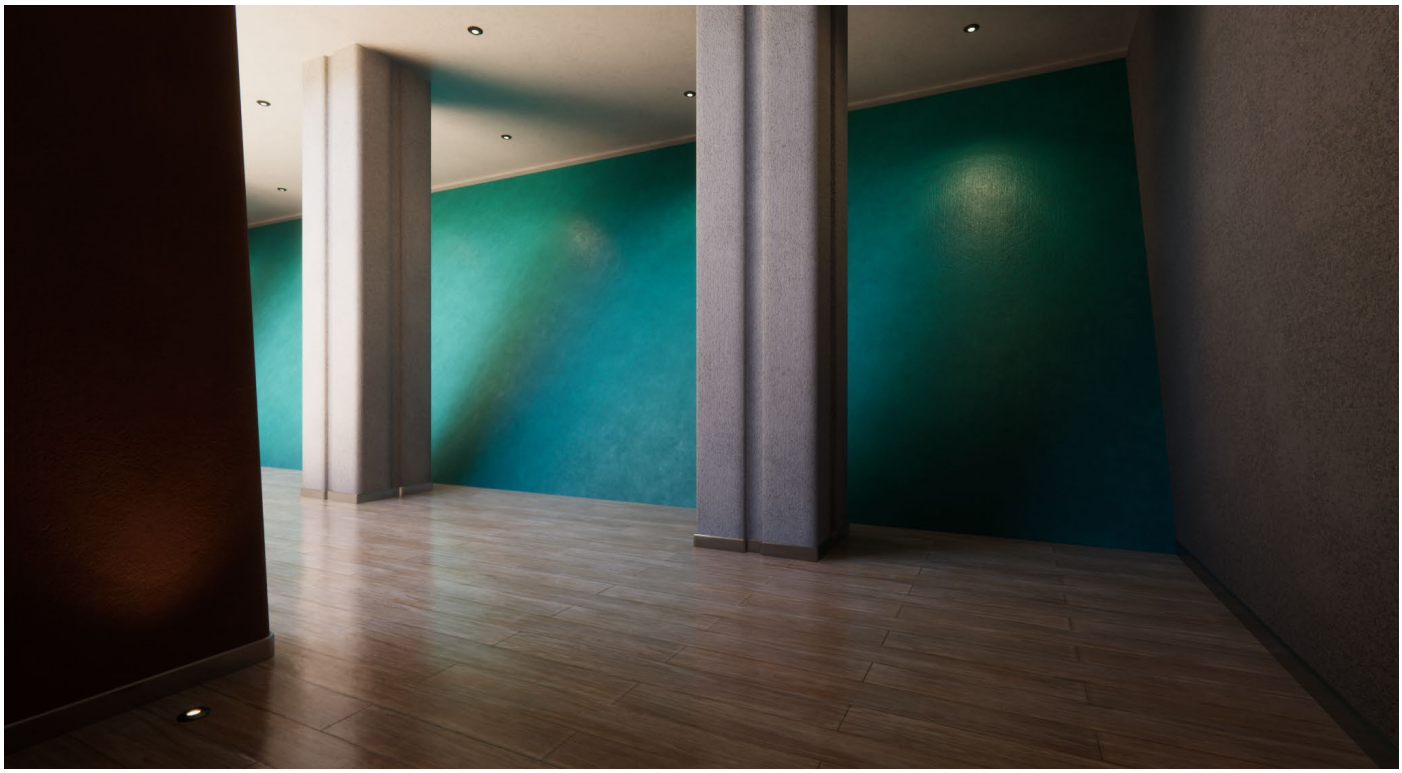
To learn more about all of the ray tracing effects available in Unity, such as ray-traced GI and reflections, check out this video.

More resources

Configuring Global Environment Lighting

Introduction to Lighting and Rendering

Configuring Lightmaps



An HDRP ray-traced scene in Unity

Worldbuilding

A terrain created with Unity Terrain Editor

Terrain sculpting tools

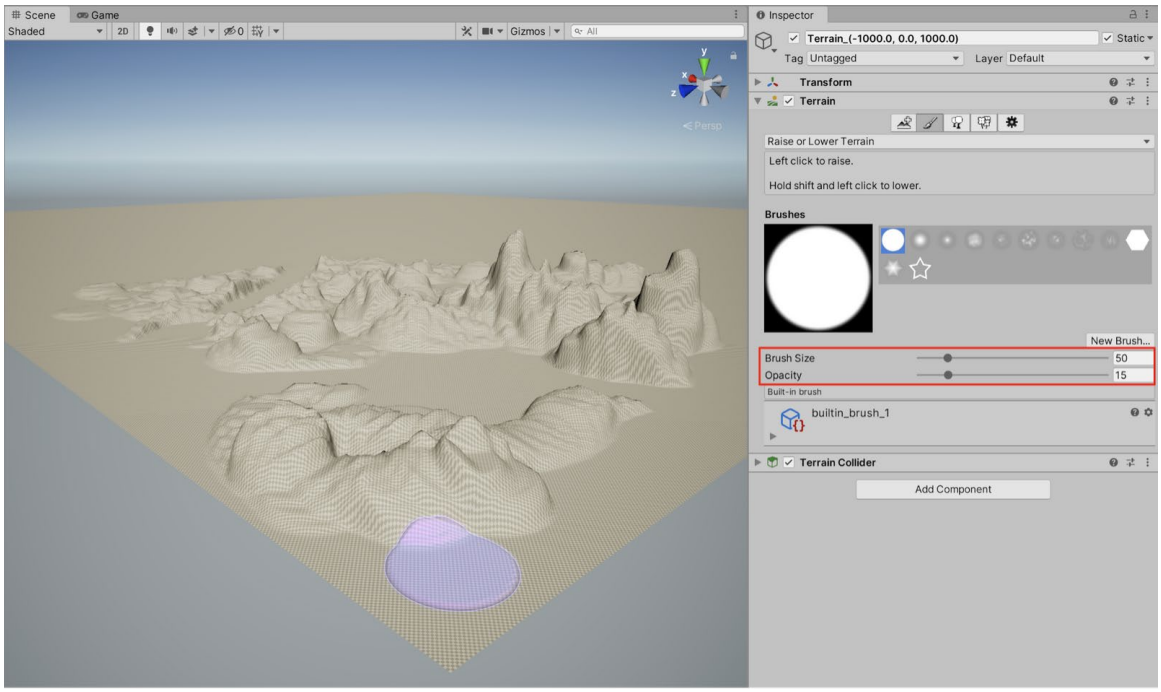
The Unity Terrain Editor enables you to create detailed, realistic, and optimized terrains with ease. The Terrain tools are available when you create or select a Terrain object in the Hierarchy window.

The Terrain component provides brushes to raise or lower terrain wherever you paint the heightmap of the terrain with a paintbrush tool. You can hide portions of the Terrain, use a stamp brush on top of the current heightmap, or refine the Terrain.

Additionally, you can paint textures that apply surface textures to the geometry of the Terrain.

To extend its functionality, the Terrain Tools package adds additional terrain sculpting brushes and tools to your project to help create stunning assets and ease workflows.

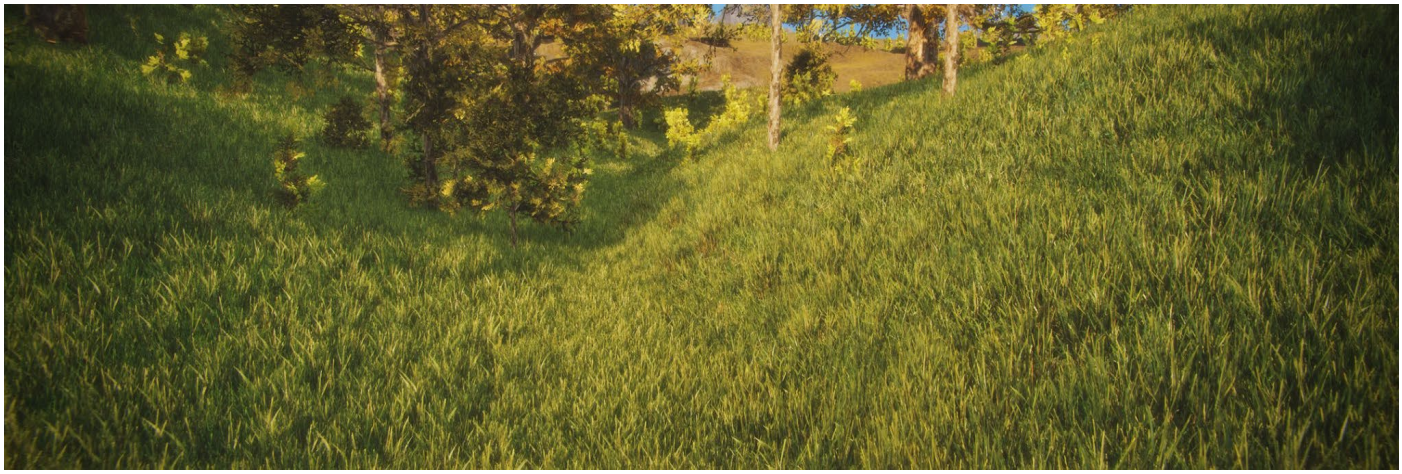
Learn how to create and customize a Terrain using specific tools and techniques with this series of tutorials from Unity Learn.



Starting to create a new Terrain with the default brush

Trees and vegetation

Unity provides brushes as part of the Terrain component. The brushes let you design tree areas and terrain details within the Editor, which is useful for creating large and detailed forests and jungles. The addition of color and size variations make the environment look organic, as the detail brush works in a similar way to the tree painting tool. It's mainly used for adding details; 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 resave your .spm files using the Unity version of the Modeler. If you're using SpeedTree Modeler 8, save your .st files directly into the Unity Project folder.

To create the effect of wind on your Terrain, you can add one or more GameObjects with Wind Zone components. Trees 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 vegetation with many options available to find the right balance between the performance and aesthetic that suits your target platform and art direction. Additionally, you can install the Terrain Tools package, which adds the Terrain Toolbox to your project with extra terrain sculpting brushes and tools for more advanced use cases.

As with other Unity tools, the Terrain API allows you to make custom Editor tooling to use the feature in the way that best suits your team of artists or designers.

SpeedTree Modeler 8 leaves working with Terrain

Prototyping

ProBuilder

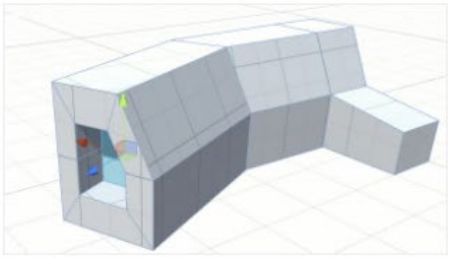
ProBuilder is a unique hybrid of 3D modeling and level-design tools, optimized for building simple geometry, and with the capacity for detailed editing and UV unwrapping.

Available via the Package Manager, ProBuilder enables you to quickly prototype structures, complex terrain features, vehicles, and weapons, as well as to make custom collision geometry, trigger zones, or nav meshes.

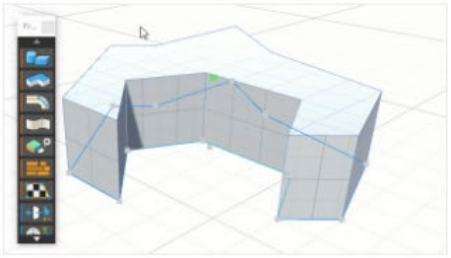
ProBuilder takes advantage of Unity's seamless roundtripping capabilities with digital content-creation tools, so you can further detail and polish models with your favorite tools and preferred features.

You can create predefined shapes with the Shape tool, which includes a library of both standard and complex geometric shapes that correspond to common objects in level-building. You can also create a 2D shape and extrude it or use the Bezier Shape for creating curved meshes that wrap around the line of the spline.

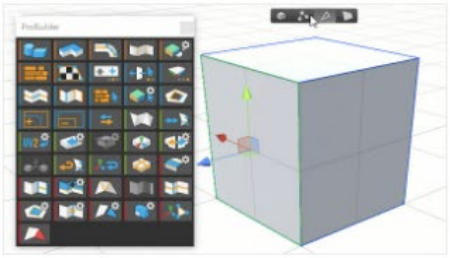
Create new Meshes with ProBuilder or modify existing ones. You can apply a Material to the entire Mesh, or on selected faces. This allows you to provide more realistic-looking surfaces during gameplay or while grey-boxing. For instance, you might decide to use tiles on the floor, brick on some walls, and stone on others. The UV Editor inside the tool will also allow you to unwrap the UVs or edit them.



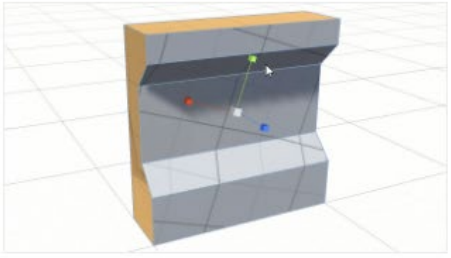
Extrude and inset



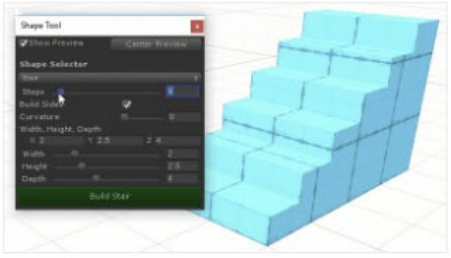
Versatile Poly Shapes



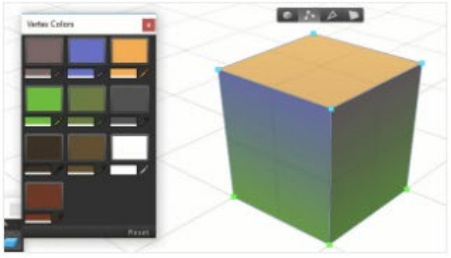
Dynamic user interface



In-scene UV controls



Procedural shapes



Vertex coloring

Some of the functionalities inside ProBuilder for creating 3D meshes

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 gives you a complete in-Editor level design solution.

Using the FBX Exporter for refining

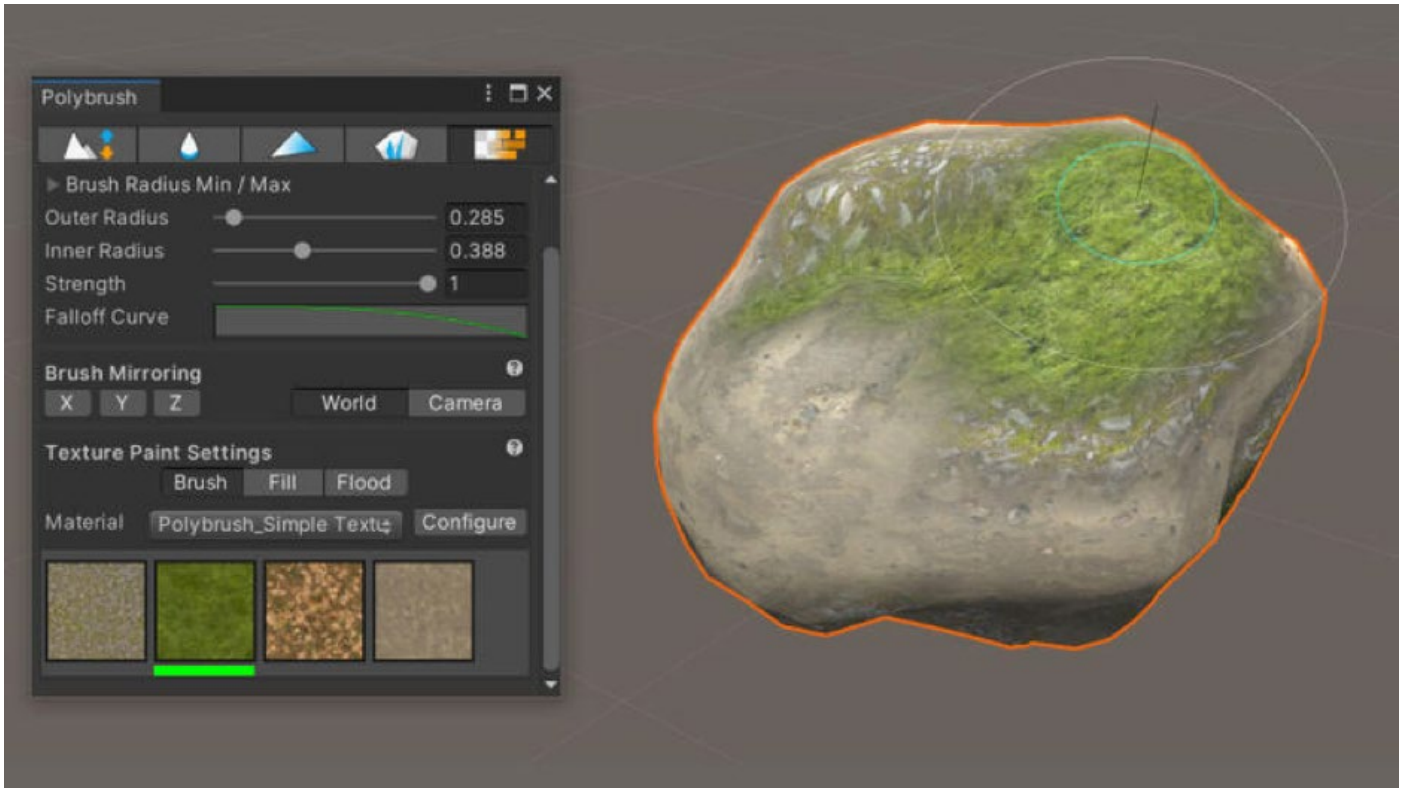
Combine ProBuilder, Polybrush, and FBX Exporter in your workflow to quickly grey-box levels and models for rapid prototyping and functionality testing. The FBX Exporter allows you to tailor assets to the correct dimensions before exporting them to a DCC to polish and refine them.

More resources

Introducing the Terrain Editor

Build beautiful landscapes

Faster level design with ProBuilder and Polybrush
Asset management with FBX Exporter, ProBuilder, and Polybrush



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

Animation

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

Animation System

Animations for Unity projects are typically created from motion capture, or by using software such as Maya, 3ds Max, or Blender.

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-like system called an Animator Controller.

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

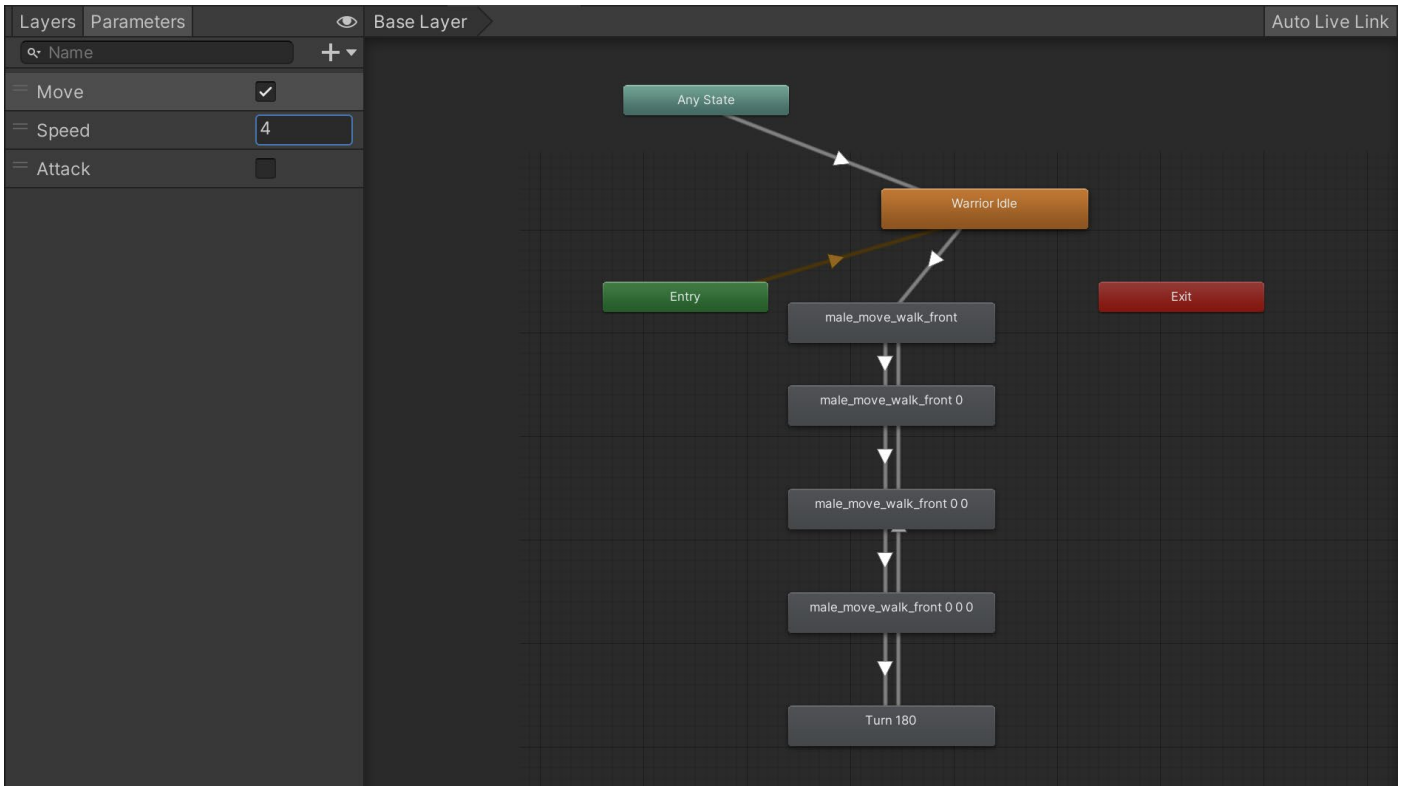
Animation State Machines

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 the Transitions between them using a 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.

Blend trees are good for hiding complexity. A blend tree doesn't have state; it doesn't 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 and prevent bugs down the road, since you can't tie behavior to most of the animations in a blend tree.

Unity uses Animation Layers for managing complex state machines. For instance, you can 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.



The flow of Animation Clips and parameters in Animation Controller

Animation Window

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

Besides animating movement, the Editor also allows you to animate variables of materials and components (almost any GameObject's properties) and augment your Animation Clips with Animation Events, which are functions called at specified points along the Timeline.

Unity's Animation Window also allows you to animate:

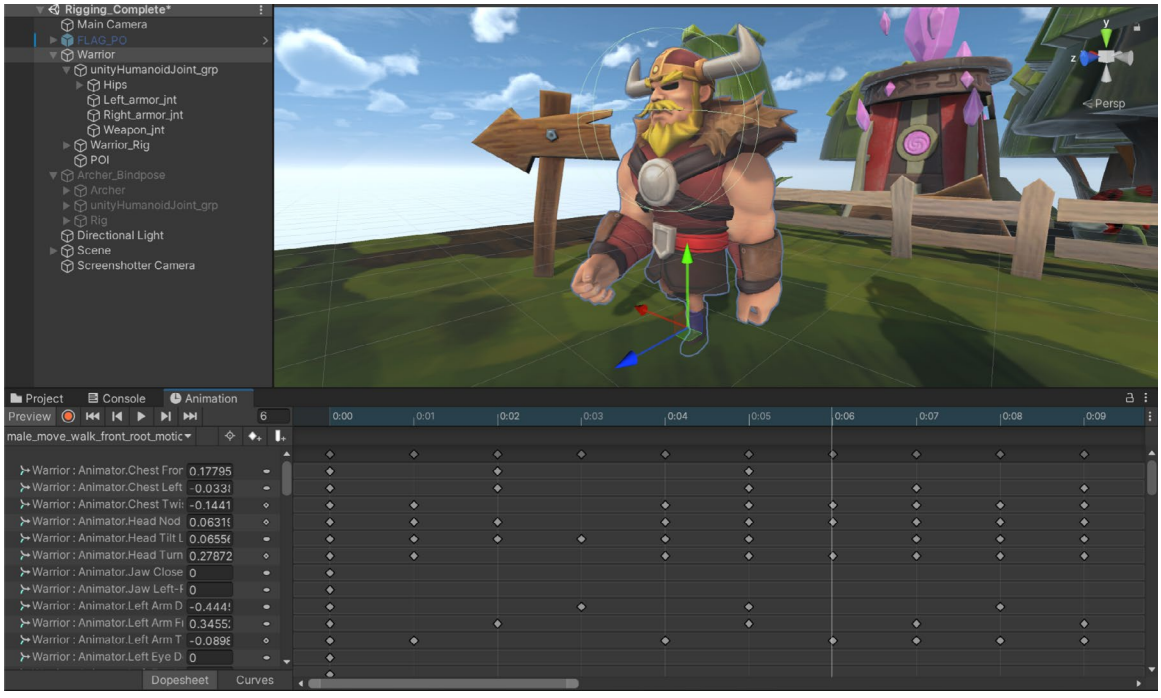
— The position, rotation and scale of GameObjects

— Component properties such as material color, intensity of light, and sound volume

— Properties within your own scripts, including float, integer, enum, vector, and Boolean variables

— The timing of calling functions within your own scripts

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



With the Animation tool, you can animate with Keyframes and Curves.

Animation Rigging

The Animation Rigging package enables you to set up procedural motion on animated skeletons at runtime. You can use a set of predefined animation constraints to manually build a control rig hierarchy for a 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.

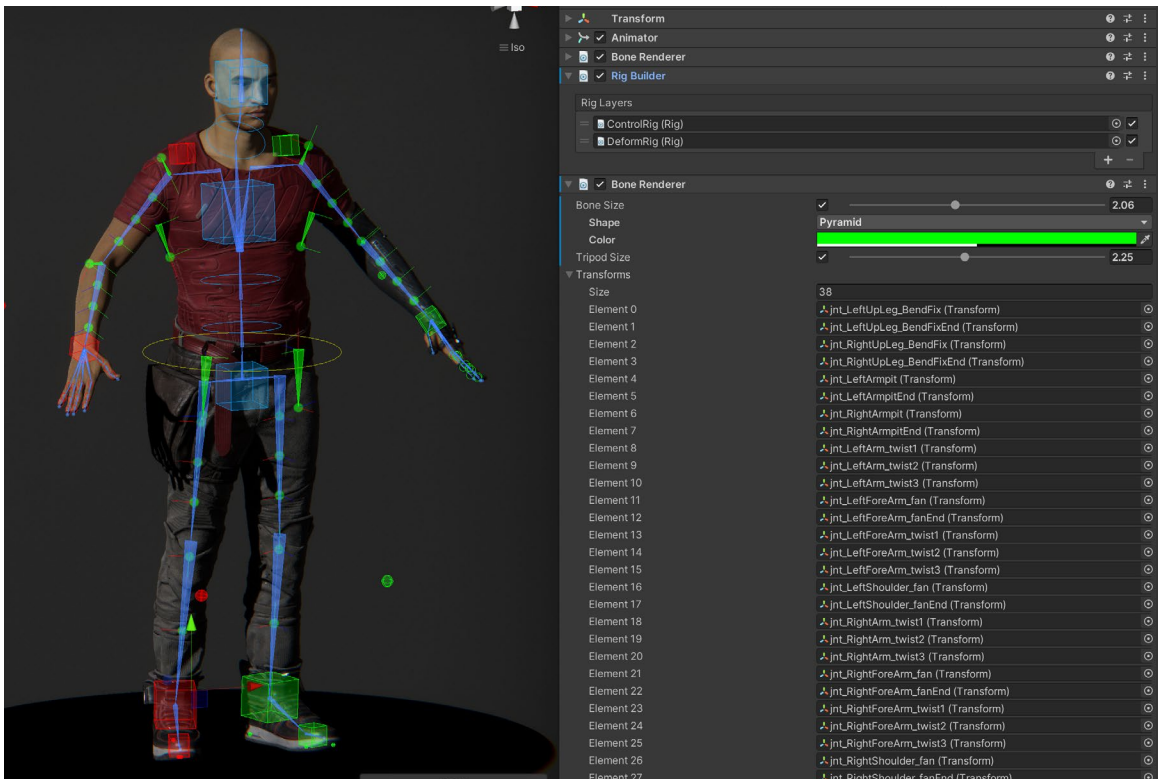
You can create dynamic animations, but also modify or create new Animation clips with the Animation tool, or create cinematic sequences with Timeline.

More resources

Working with Animation Clips

Improve workflow with Animation Rigging

Reusing and retargeting animations between rigs

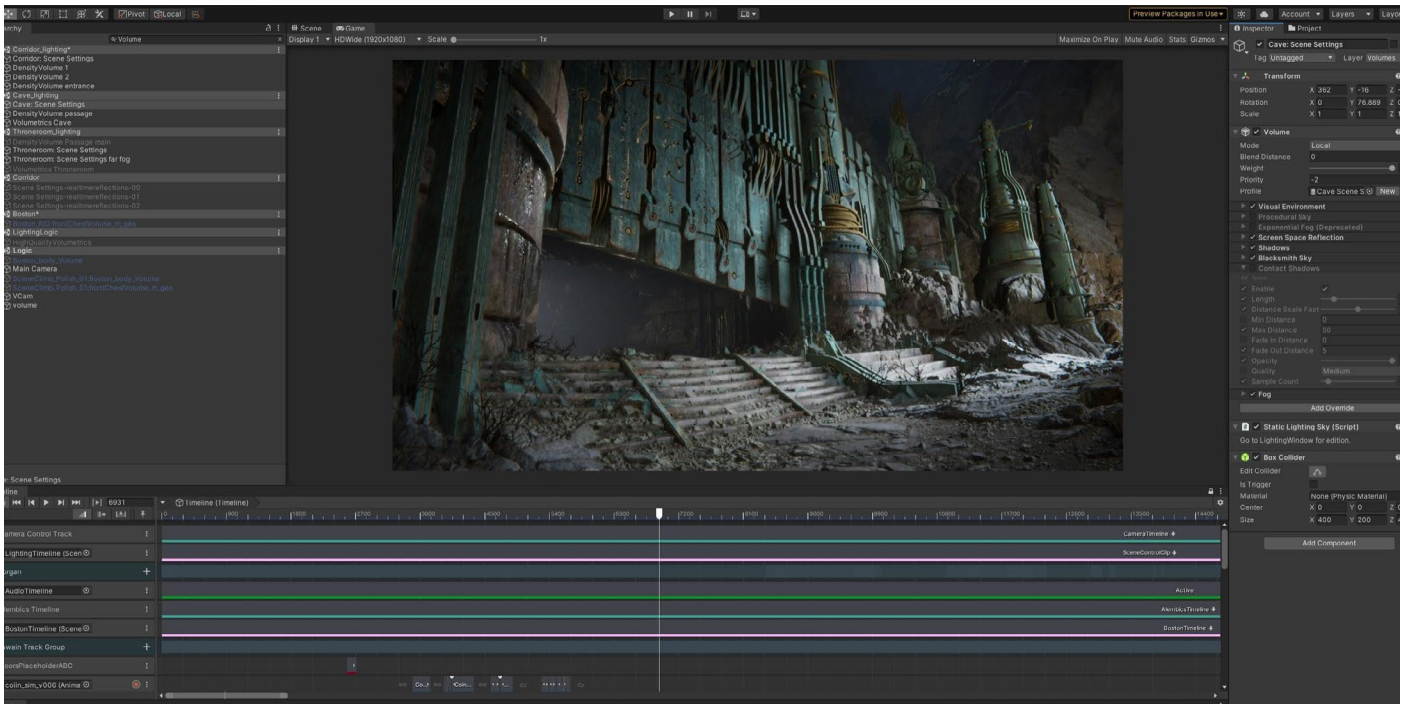


The Bone Renderer feature in Animation Rigging

Cutscenes and cinematics

Gone are the days of pre-rendered interstitials. Unity allows you to create cutscenes and immersive cinematics completely in-engine, and it's equipped with many of the features that once required offline rendering.

The Heretic, a short film created by Unity's Demo team, uses Unity's cinematic features, Cinemachine and Timeline. In the short film, the camera work is "shot" differently in each of its two halves. When Gawain, the main character, and Boston, his sidekick robotic bird, first enter the scene at the beginning, the camera has a handheld feel. To produce this effect, the Demo team created a custom Timeline track and imported motion capture data from real cameras.



Everything in the Unity demo *The Heretic* plays back at 30 frames per second.

Gawain, the main character in *The Heretic*, enters a cave at the start of the film. The tunnel later becomes a portal to another world.

In the second part, when the pair exits the cave, the camera sweeps across the vista of Morgan's Citadel. From that point on, the team opted for smoother movements. Cinemachine partially automated some of the cameras in the throne room. This is just one example of how the cinematic tools in Unity can be used to produce compelling storytelling.

Morgan's Citadel in *The Heretic*.

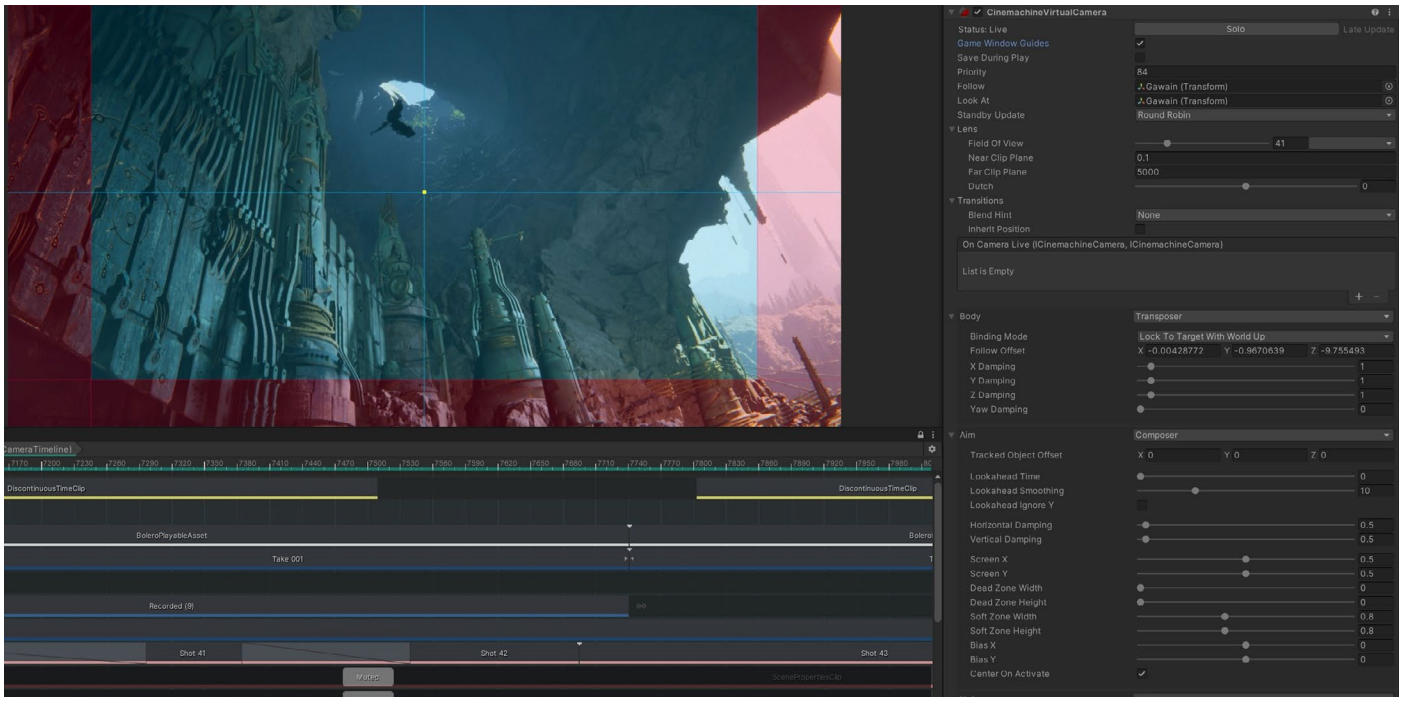
Cinemachine

Cinemachine is a suite of tools that enable you to dolly, track, and zoom your game camera, similar to how a Hollywood camera operator would work on set.

Cinemachine helps you frame and follow your subject without coding, and 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.

If you've already developed your own camera system, Cinemachine can even work side-by-side with your custom camera solution.

Cinemachine works for both in-game and cutscene camera animation. Use it across all genres: FPS, third-person, side-scroller, top-down, and RTS.



Designing cinematics with Timeline and Cinemachine

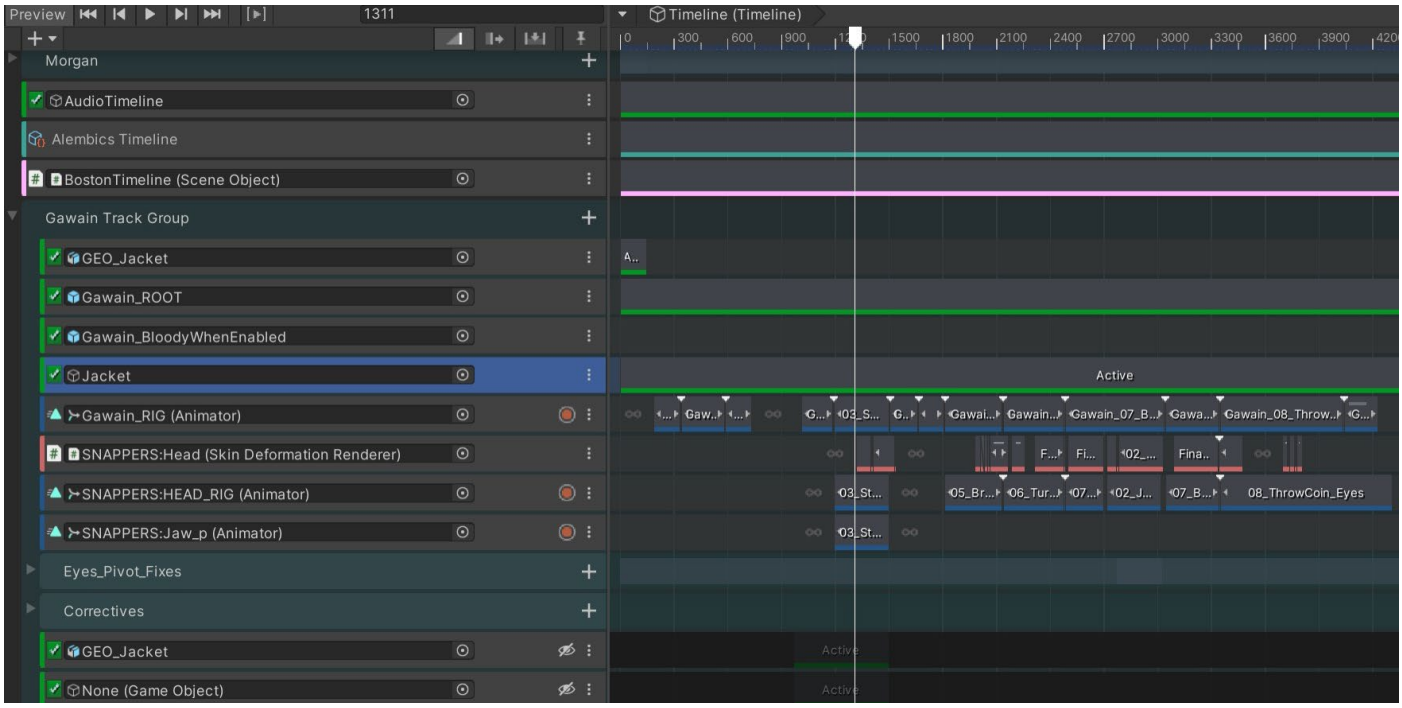
Timeline

Timeline rounds out Unity's virtual filmmaking tools. This interface allows you to control behavior, play animations, and enable/disable objects. Here, you can test your editing skills as both director and editor.

Just 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, or particle effects. The machinima of yesterday is now much closer to feature film animation than ever before.

Because the Demo team used Timeline, *The Heretic* could be edited in real-time. The team did not need to rely as extensively on animatics or storyboards as they would have had to while working in traditional animation or visual effects.

To facilitate this process, the Demo team created a custom SceneMaster tool to link various properties to the Timeline clips. They could set parameters for a light or object, save a "snapshot clip," and then add that to the Timeline. This allowed them to easily embed shot-specific settings that would update automatically as they scrubbed through the interface.



The Unity Demo team customized Timeline to edit their film in-engine.

For the creative team, this meant additional freedom to edit non-destructively. The film could undergo significant changes and recuts very close to delivery – something unthinkable when using most other media.

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

More resources

Making of *The Heretic*: Environment Art

Blending gameplay and storytelling with Timeline: Cutscenes and game graphics

Empowering storytellers with real-time technology, Part I and Part II

Visual effects

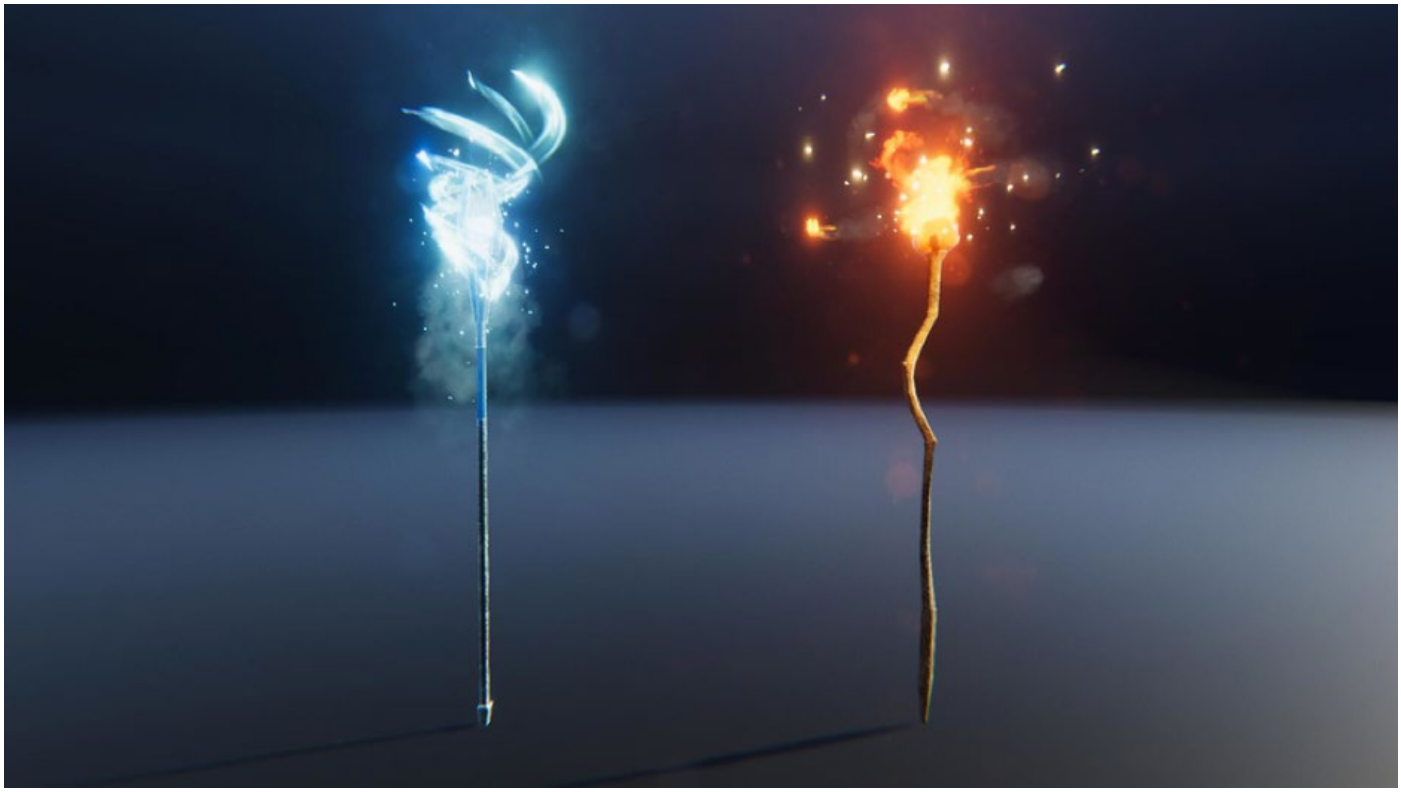
Unity currently offers two main particle systems for achieving visual effects, the Built-in Particle System and the Visual Effect Graph (or VFX Graph).

Built-in Particle System

The Built-in Particle System gives you full read/write access to the system, and the particles that 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 easily define the emission rate of particles, shape of the emissor, duration of particles, as well as their appearance, movement, and behavior during their lifetime.

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, and overdrawn by other particles, as well as render particle trails.



Visual effects made with the Built-in Particle System

The Lights module is a fast 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.

You can enable Vertex data streams for passing information about each particle to custom shaders or C# scripts. Other features include the Force Field component, which applies forces to particles in Particle Systems, and C# Job System integration for writing performant C# behaviors.

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

Visual Effect Graph for the Scriptable Render Pipelines

The VFX Graph enables 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.

Effects created using the Visual Effect Graph

The VFX Graph simulates particle behavior on the GPU, which allows it to simulate many more particles than the Built-in Particle System. Since the VFX Graph is simulated on the GPU, it only supports compute-capable platforms.

Some key benefits include the ability to have more particles with faster simulation, customizable behaviors, extensibility (to create subgraphs, templates, etc.), camera buffer access, and native Shader Graph integration.

VFX Graph components

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

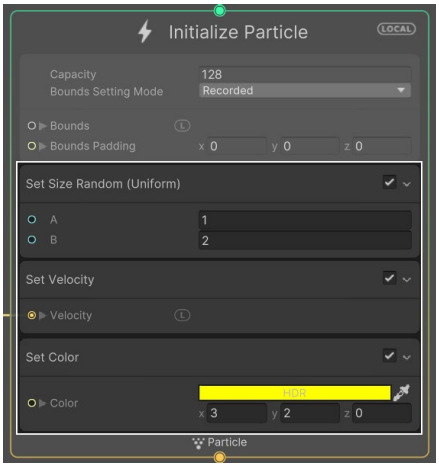
Contexts represent the order of operation in which particles are processed:

— Spawn: Controls particle spawning

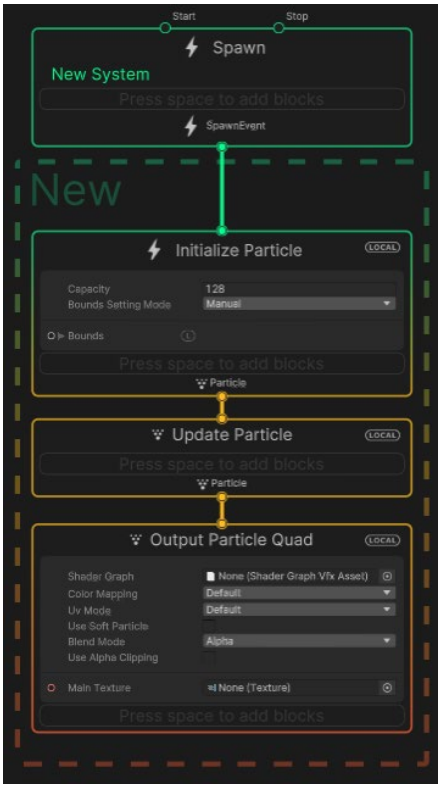
— Initialize: Sets the initial particle values

— Update: Controls particle behavior over time

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



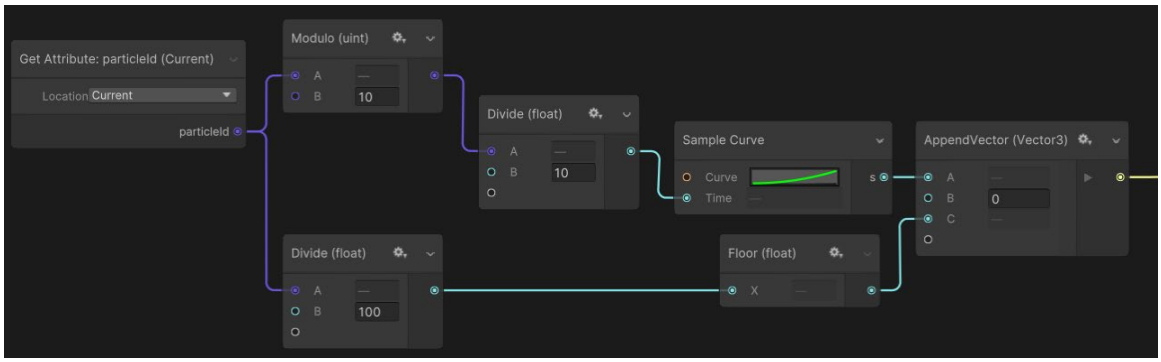
A Block in a context in the VFX Graph



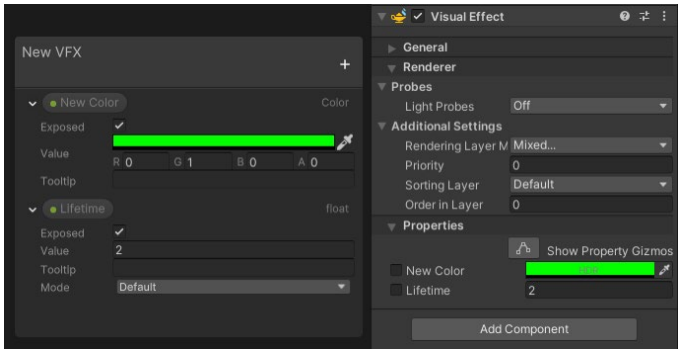
Contexts in the VFX Graph

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 right-clicking within a context and selecting Create Block.

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



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



Blackboard properties exposed and editable in the Inspector

For more information, visit the Visual Effect Graph manual.

Post-processing

Unity provides a large number 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, as well as many 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.

URP and HDRP include their own post-processing solution, which Unity installs when you create a project using a URP or HDRP Template. 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.

More resources

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

Post-processing effects in the Unity demo made for the Neon Challenge

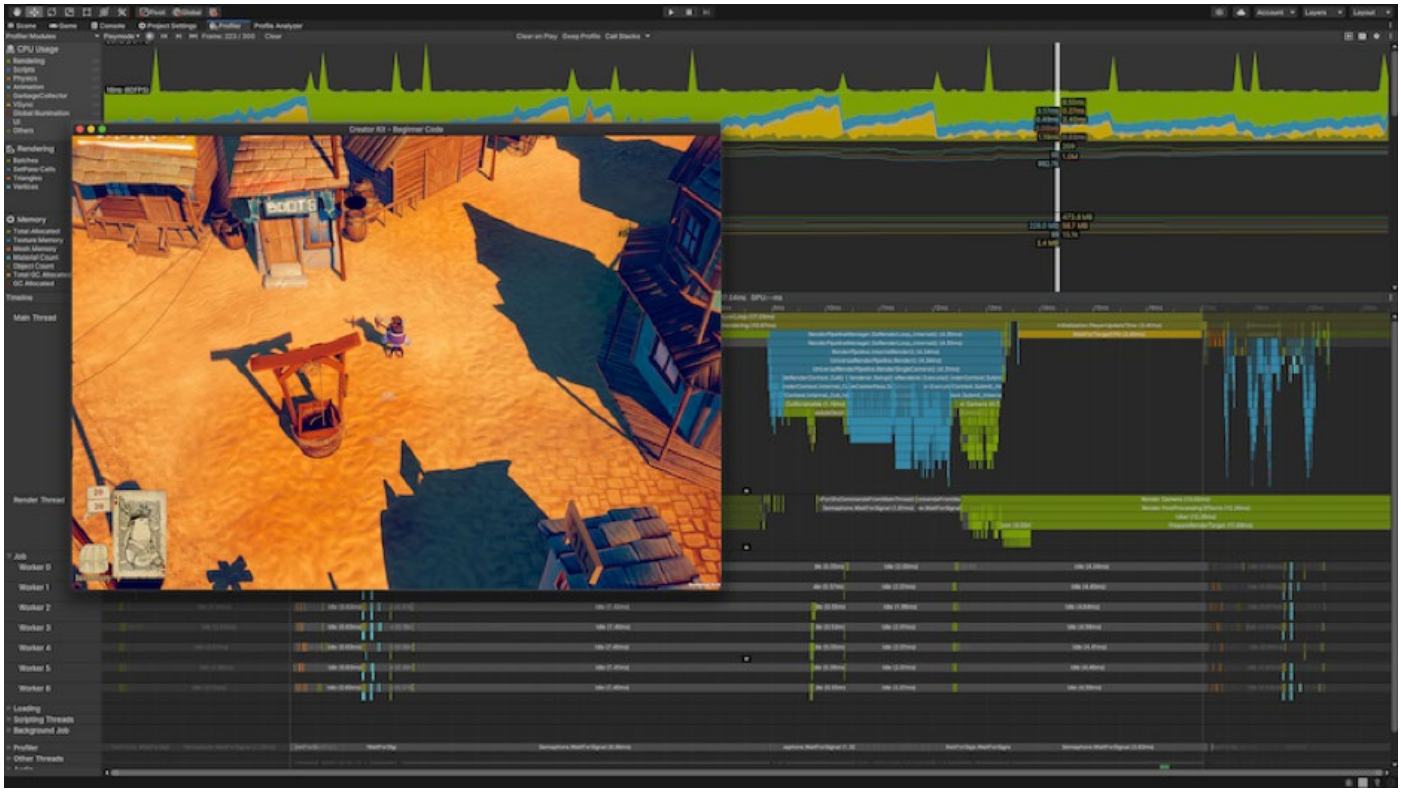
Profiling and debugging

Profiler

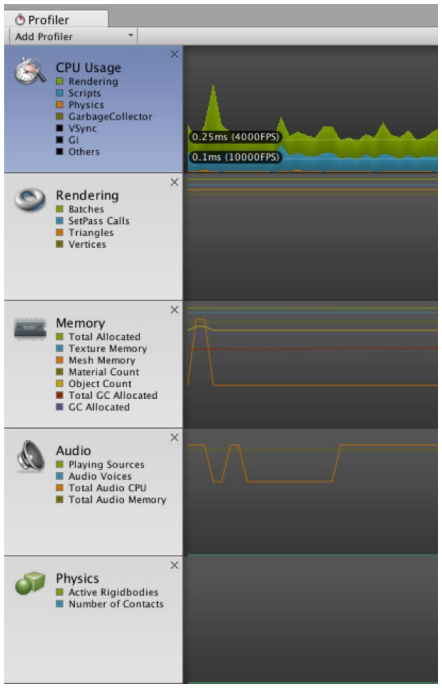
The Unity Profiler is a tool you can use to get performance information about your application. You can connect it to devices on your network to see how your application runs on the intended release platform. You can also run it in the Editor to get an overview of resource allocation while developing 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 settings affect your application's overall performance. It displays the results in a series of charts, so it's easy to 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.

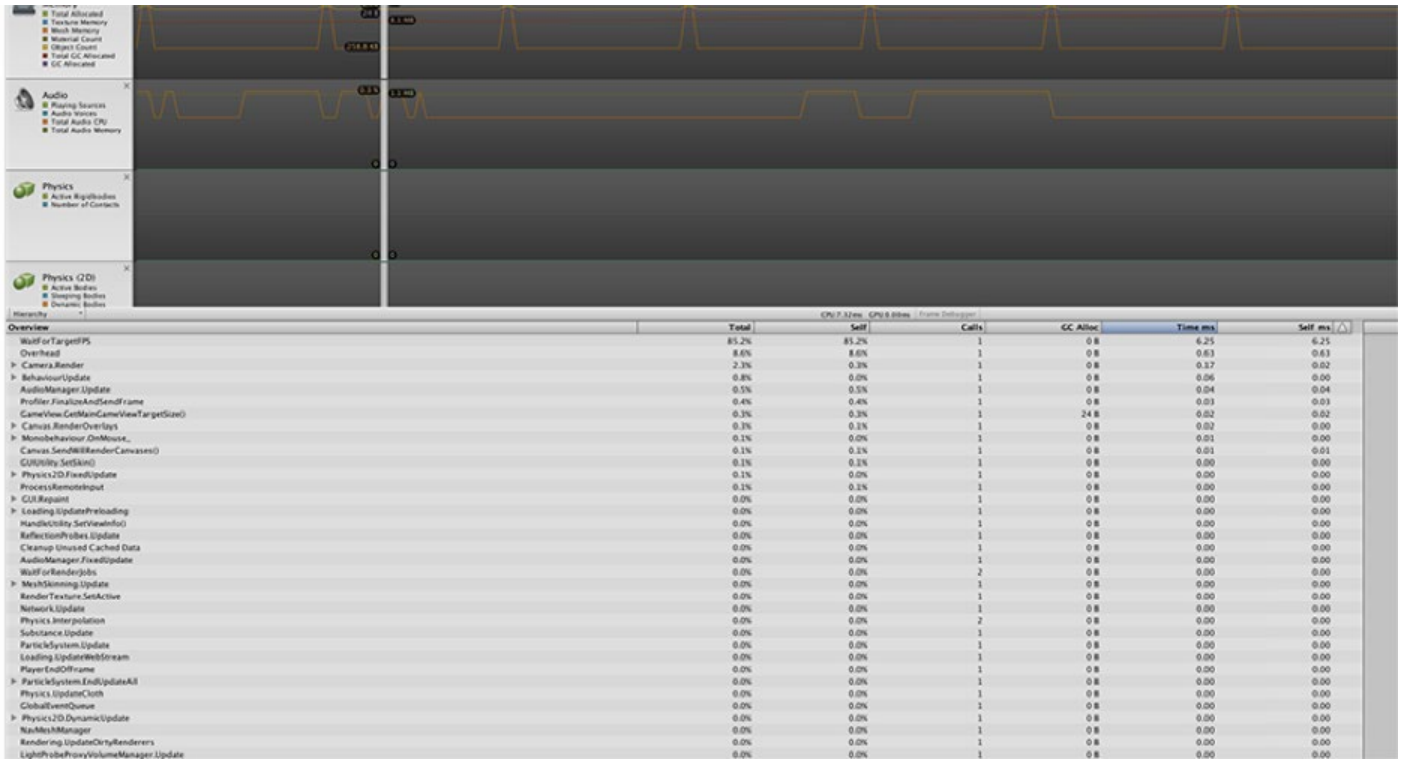


Overview of the Profiler displaying stats of a game running in-Editor



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.

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

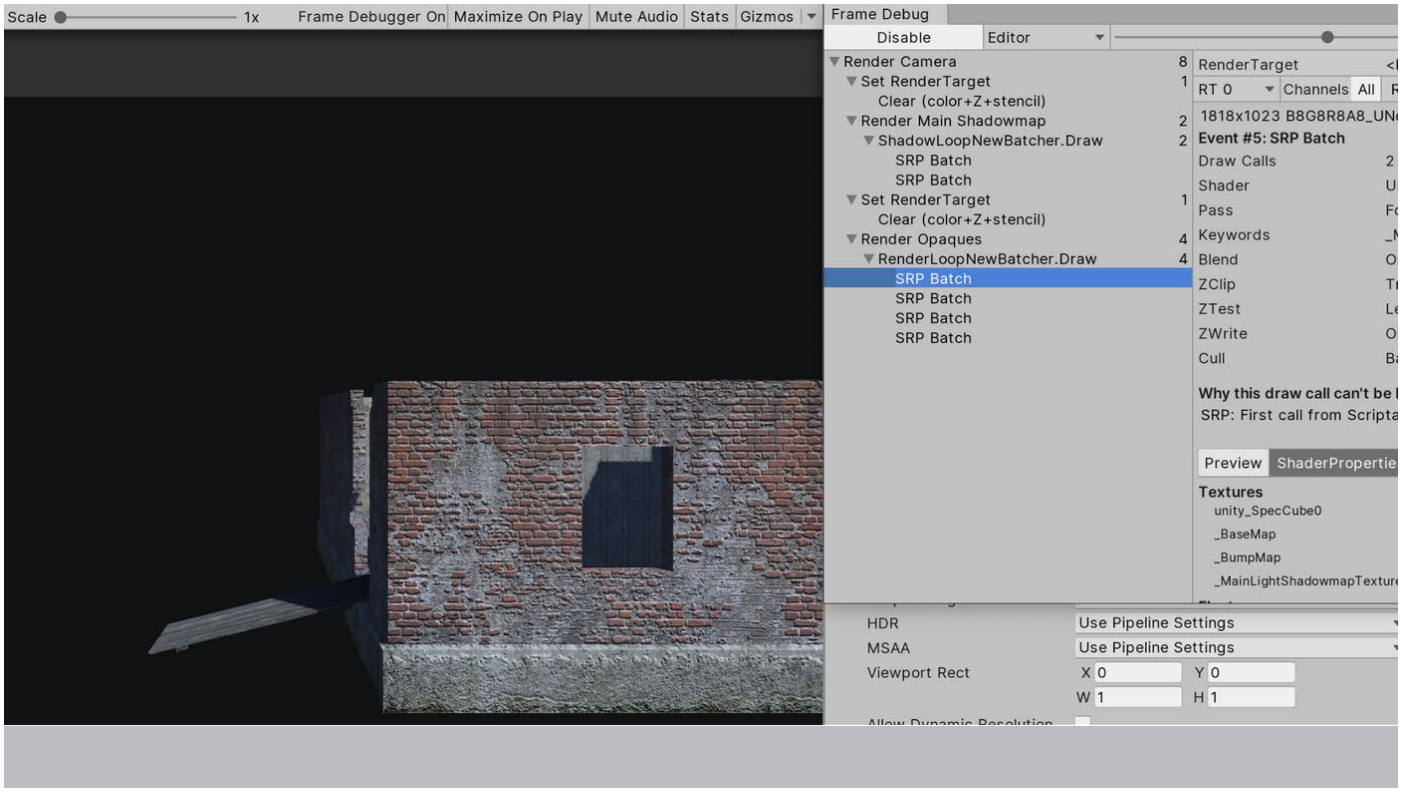


Frame Debugger

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 will help you debug your projects whenever particular Scenes cause frame rate issues.

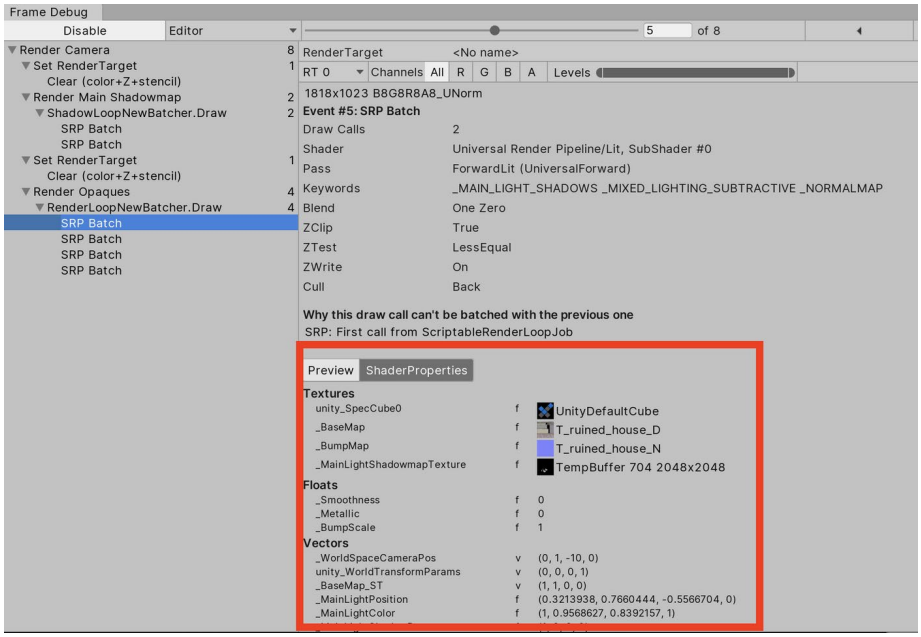
The Debugger can be found under the **Windows > Analysis** tools. Once you enable it, the Game view will freeze and you will be able to observe the different draw calls taking place on the screen in the paused frame. For example, when you click on any of the Draw Mesh calls on the left side of the window,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

The Frame Debug window is segmented into sections so you can efficiently 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 further information on the selected draw call, such as the geometry detail and the shader used for rendering.



A render pass in the Frame Debugger

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



The ShaderProperties view in the Frame Debugger

More resources

[How to profile and optimize a game]

[Introduction to profiling in Unity]

2D game development

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 flexibility and scalability to create any kind of 2D game or experience across many platforms.

Unity's native 2D tools

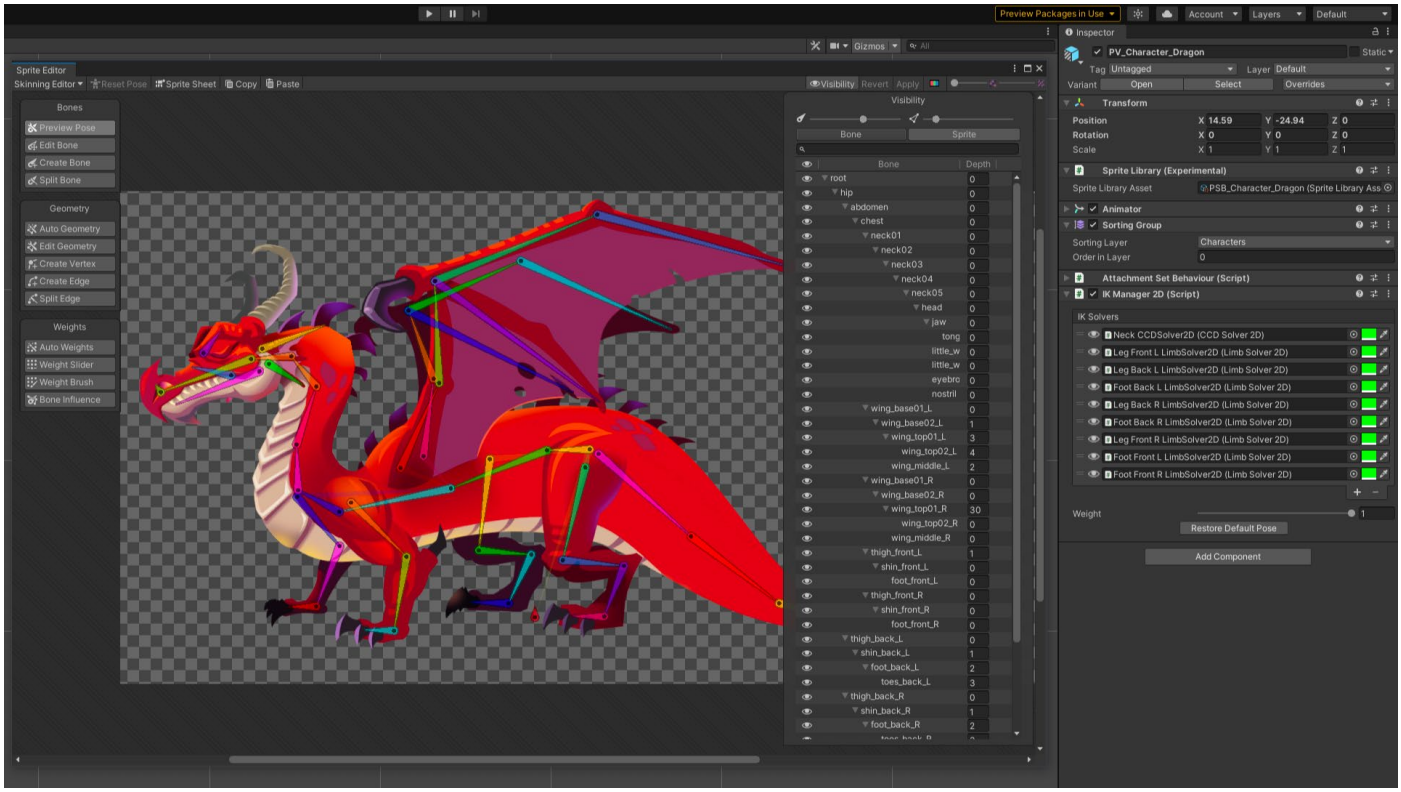
Beyond cutting-edge graphics, Unity has all the features you need for 2D game development: Sprite support, 2D skeletal animation with Inverse Kinematics (IK), worldbuilding with tilemap-based grids or organic shapes, 2D physics, Sprite Atlasing tools for packing sprites into textures, and more.

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

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

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.



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

2D graphics: Lights and shaders

You can make your 2D visuals or gameplay more immersive with dynamic 2D lightning. 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 components that act as 2D counterparts to familiar 3D Light components.

In the Inspector, you can easily apply parameters, such as light colors, intensity, fall-off, 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.

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

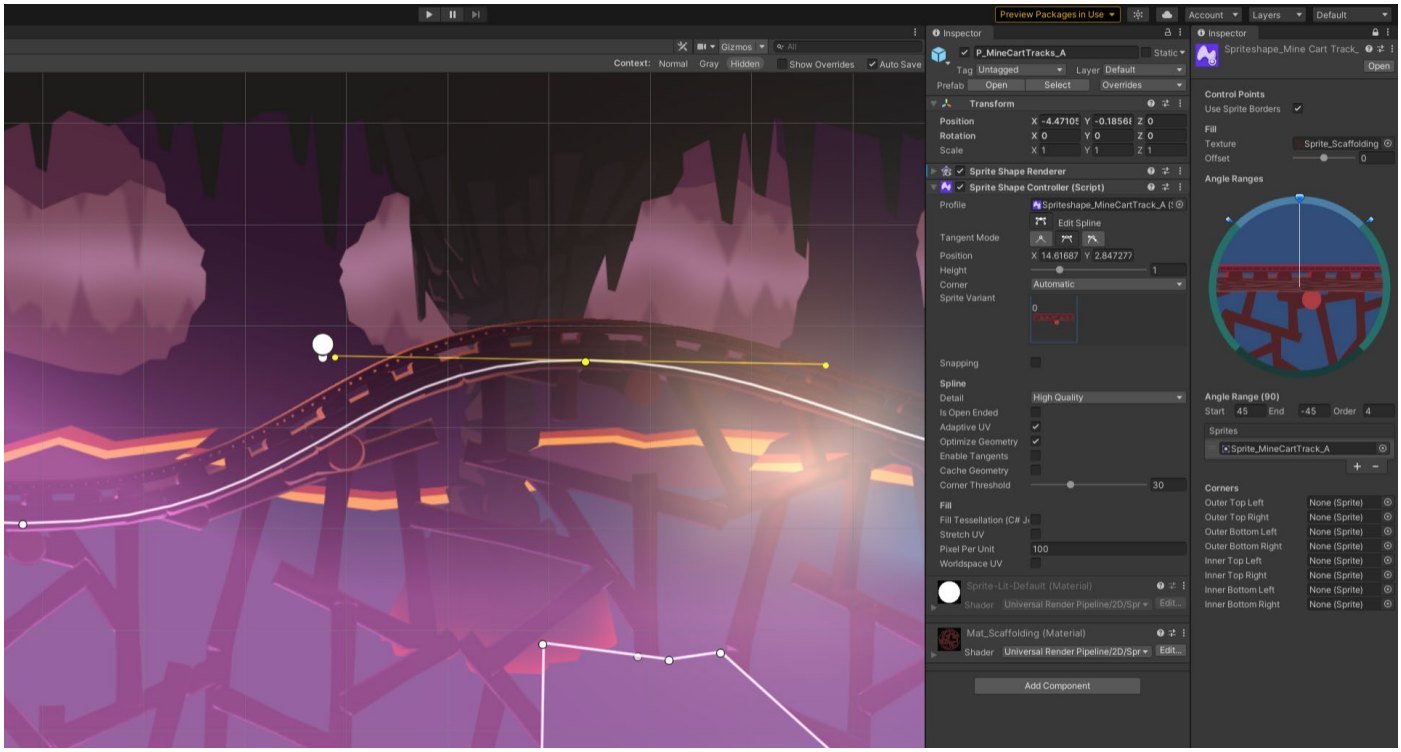
Worldbuilding

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, and the Tilemap API and additional features offer a slew of creative possibilities. Create custom brushes, include GameObjects on your Tile Palette, and apply different Sprite sorting layer logic based on your needs.

Sprite Shape

Create organic 2D environments with a visual and intuitive workflow thanks to 2D Sprite Shape. Similar to vector drawing software, Sprite Shape features Sprite tiling along a shape's outline, and automatically deforms and swaps Sprites based on the angle of the outline. 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.



Modifying a track's Bezier curves with Sprite Shape

Sprite technology

The Sprite Editor facilitates the setup of art assets for your 2D projects. Configure the Pixels per Unit (PPU) of your 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, it's easy to mix 2D and 3D elements in the same scene. 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 easy 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 best approach for the 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.

More resources

Speed up 2D art workflows

Great tips for making 2D games

Make retro 8-bit and 16-bit games

How to set the mood with 2D lights

Appendix 1: Digital humans in Unity

As games continue to expand on cinematic storytelling, they must tell rich stories about people. In a modern AAA title, this means creating multifaceted characters with more realism. One of the principal challenges in creating *The Heretic* was making Gawain, a digital human shown in closeups that required a high level of detail. This digital human character is available on Unity's Asset Store to use for educational or non-commercial projects.

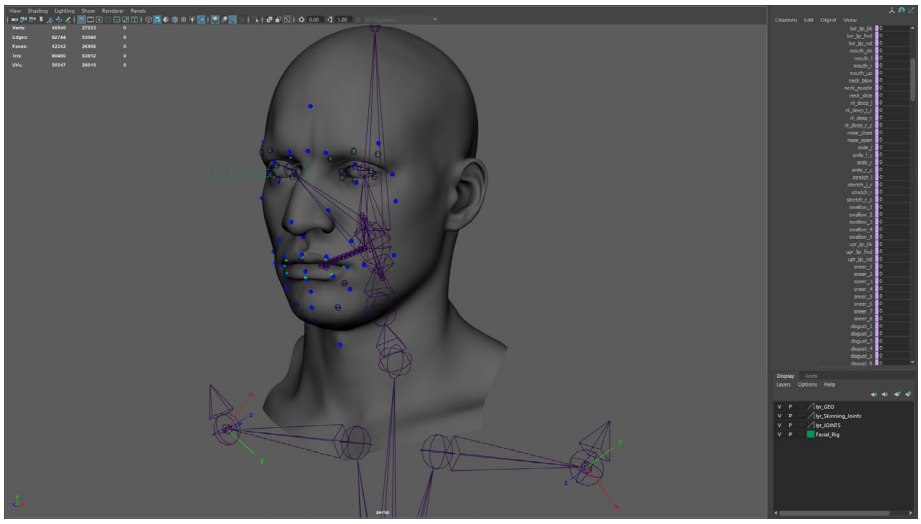
Preparation

The Unity Demo team needed to recreate all of the details that make a human, well, human. To achieve this, they opted for a hybrid of performance capture with detailing added post-capture.

Digital humans have their own set of challenges in video games.

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

For the base facial motion capture, Infinite Realities studio in London 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. This "4D data" could reproduce subtle nuances of the actor's facial movements.



The facial animation rig used over 300 blendshapes.

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

A 4D scan similarly captures nuances of the actor's performance, but this approach can be noisy and miss data.

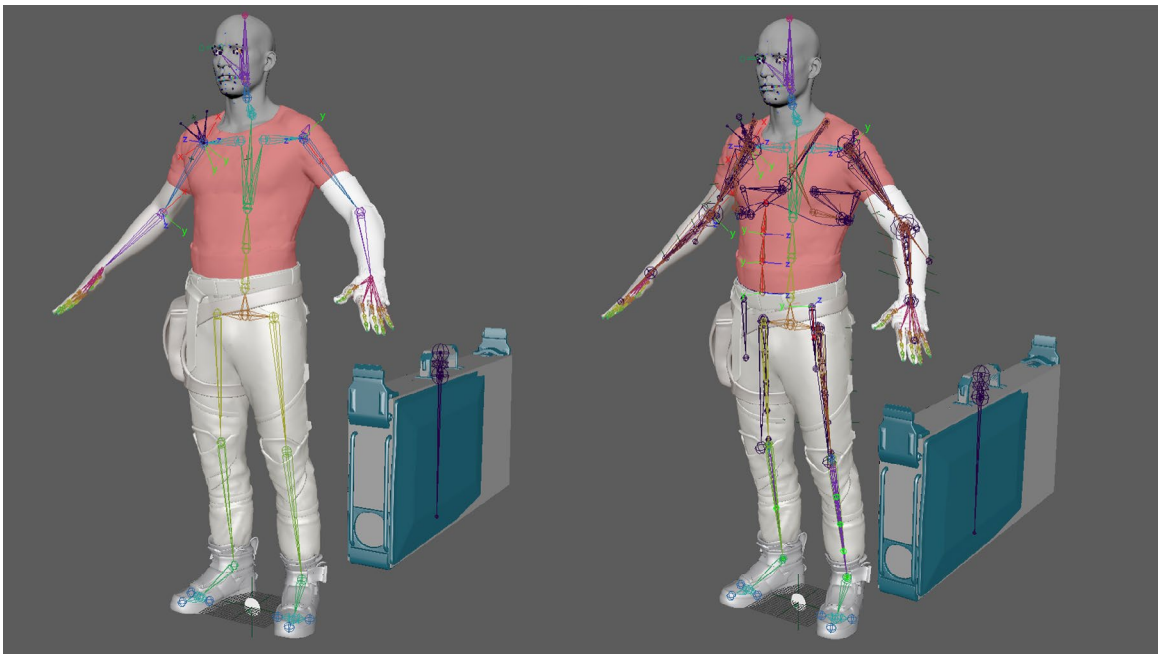
The 4D pipeline used for *The Heretic* showcases what's possible with Unity. With the proper source data and cleanup, you can push the boundaries of your facial performances.

Since the film was made in a real-time engine, the character rig had 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.

The base layer consisted of the main skeleton that was responsible for transferring the body motion between 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. More specifically, 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.

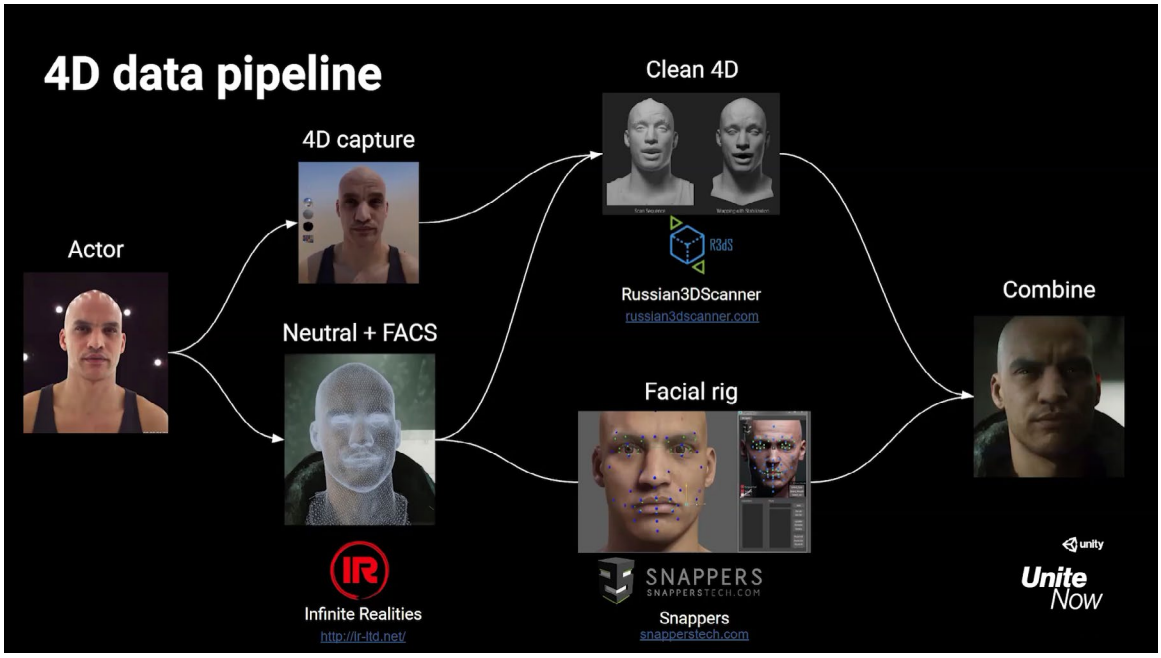
Finally, the Snappers' facial rig was referenced in the Scene before exporting to Unity.



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

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

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



A breakdown of the 4D data facial capture pipeline used for this production

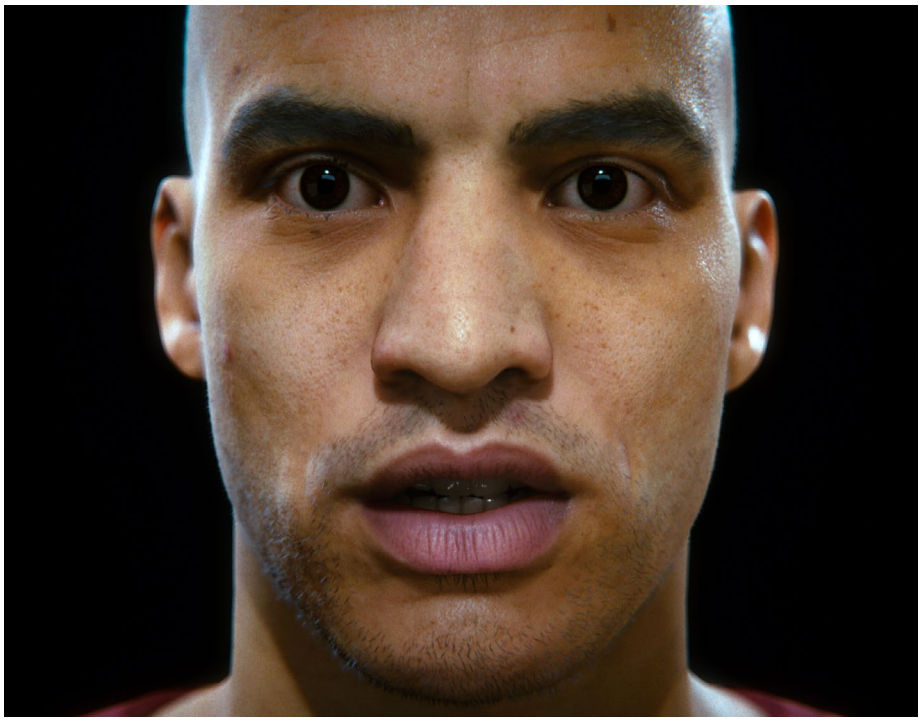
Read this behind-the-scenes blog post for more information about the 4D and motion capture process, and watch the Unite Now session Meet the Devs: *The Heretic* short film for further details regarding the pipeline.

Shading

Capturing accurate motion is just the first challenge in building a believable human character. The second is shading and rendering, especially for the skin and some regions of the face.

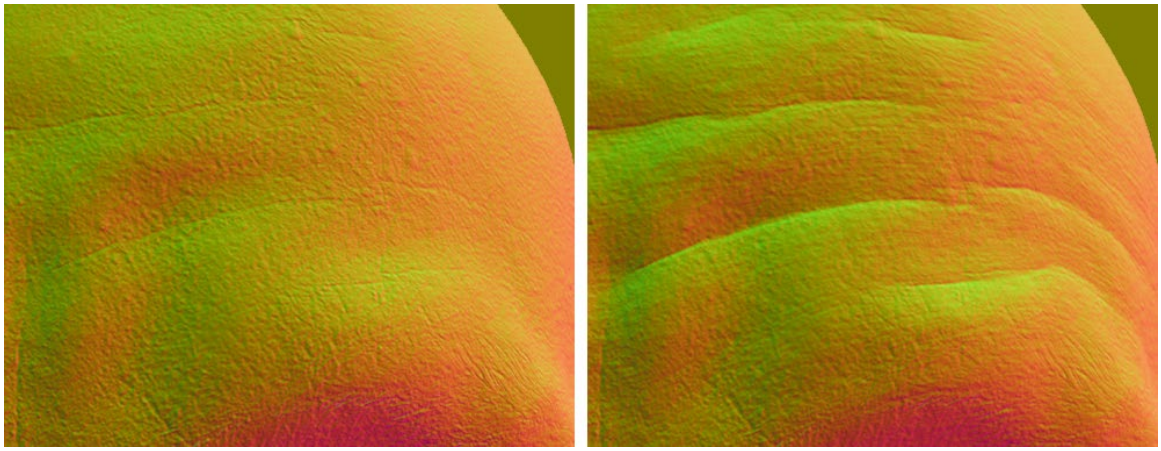
Custom shading in Unity needs to address specific features:

— Skin
— Eyes
— Teeth
— Tearline
— Eyebrows
— Eyelashes
— Stubble

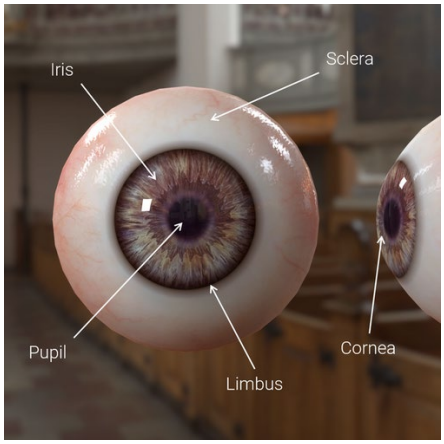


The human face is a shading challenge.

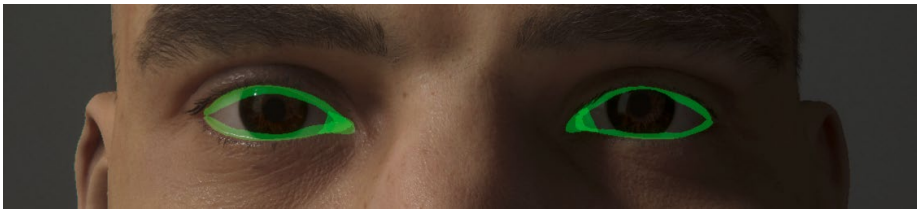
The HDRP's existing Lit and Layered Lit shaders both provided a solid foundation for these features. For example, work on the skin and teeth harnessed HDRP's existing support for subsurface scattering, which can simulate the way light penetrates and moves within an area under organic surfaces.



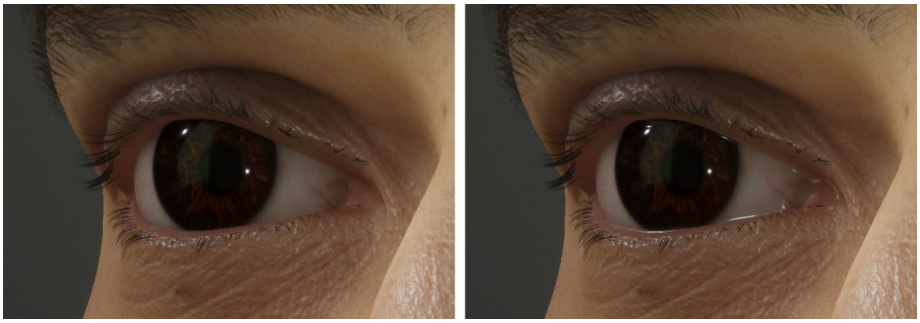
Pose-driven wrinkles added on top of the underlying 4D captured skin



The Eye Shader is available in HDRP.



A separate tearline mesh controlled the wetness of the eyes, where the eyeballs meet the eyelids.

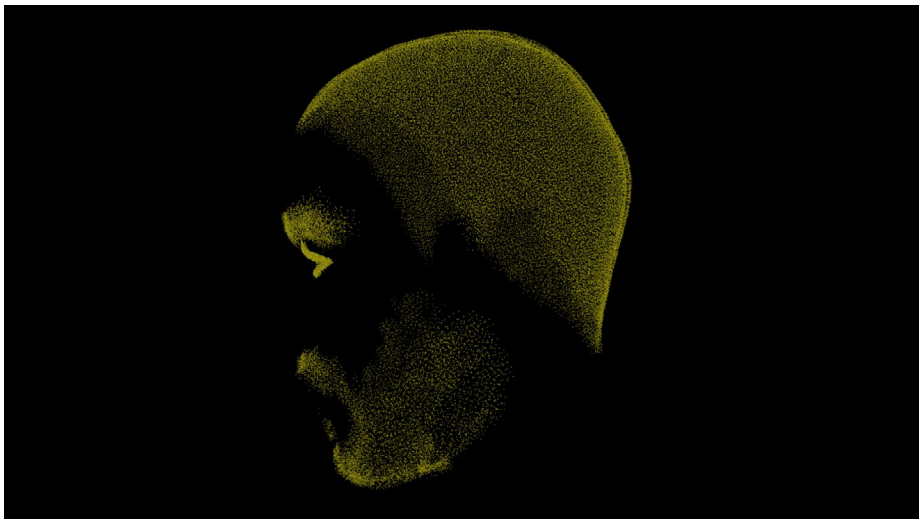


Adding the tearline (right) captured the wetness between the lid and eye.

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

The eyes received special treatment so that the cornea's shininess 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.

The character design of Gawain purposely omitted long hair, but you might be surprised to learn just how much short hair is still present. Senior developer Lasse Jon Fuglsang Pedersen created a skin attachment system to hold the brows, stubble, and lashes in place. This way, 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.

Together, all of these parts produced the digital human known as Gawain. Here are just some numbers that reflect the work that it took to create him:

— Vertices: 28k for the head, 57k for the body, and 43k for the jacket

— Maya skeleton rig: Approximately 360 joints

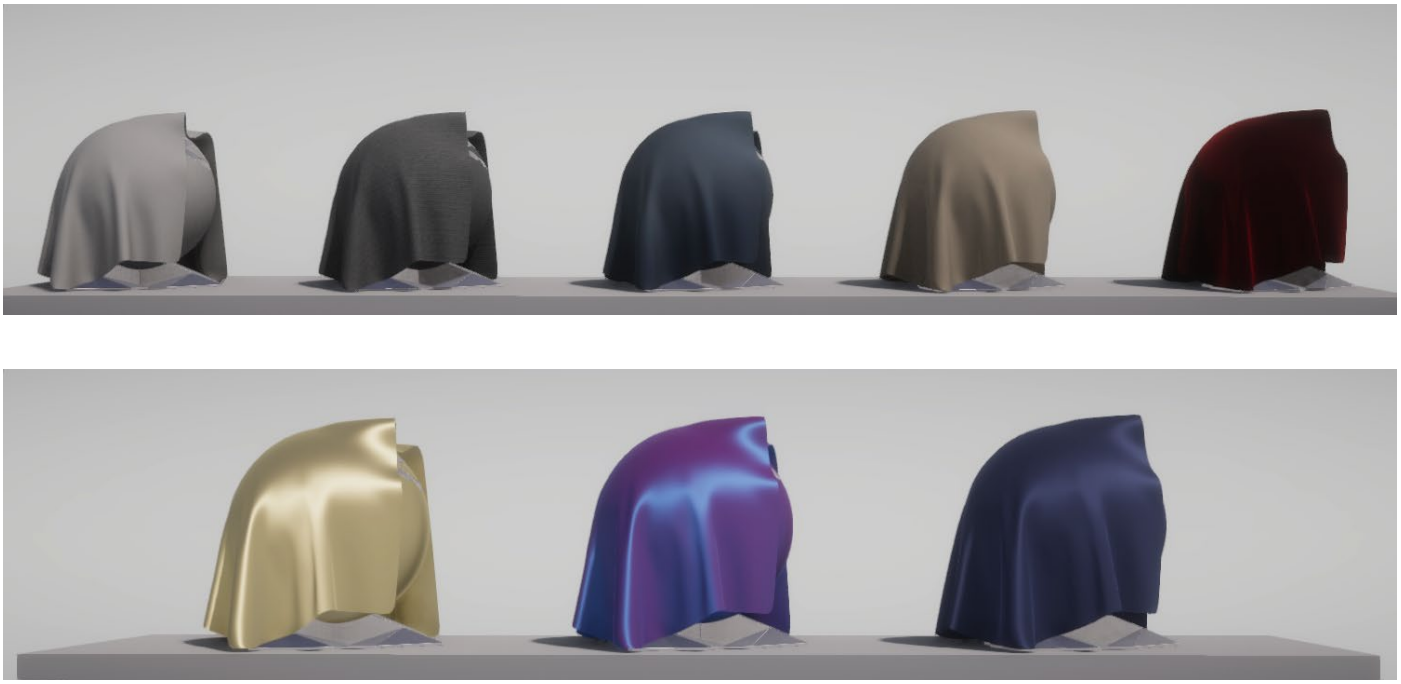
— Base texture data: 4k maps for albedo, normal, cavity, thickness, etc.

— Pose-driven texture data: 48 activation masks, 16 × 3 additional 4k maps for albedo, normal, and cavity

— Facial Blendshapes: 318

Props and clothing

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

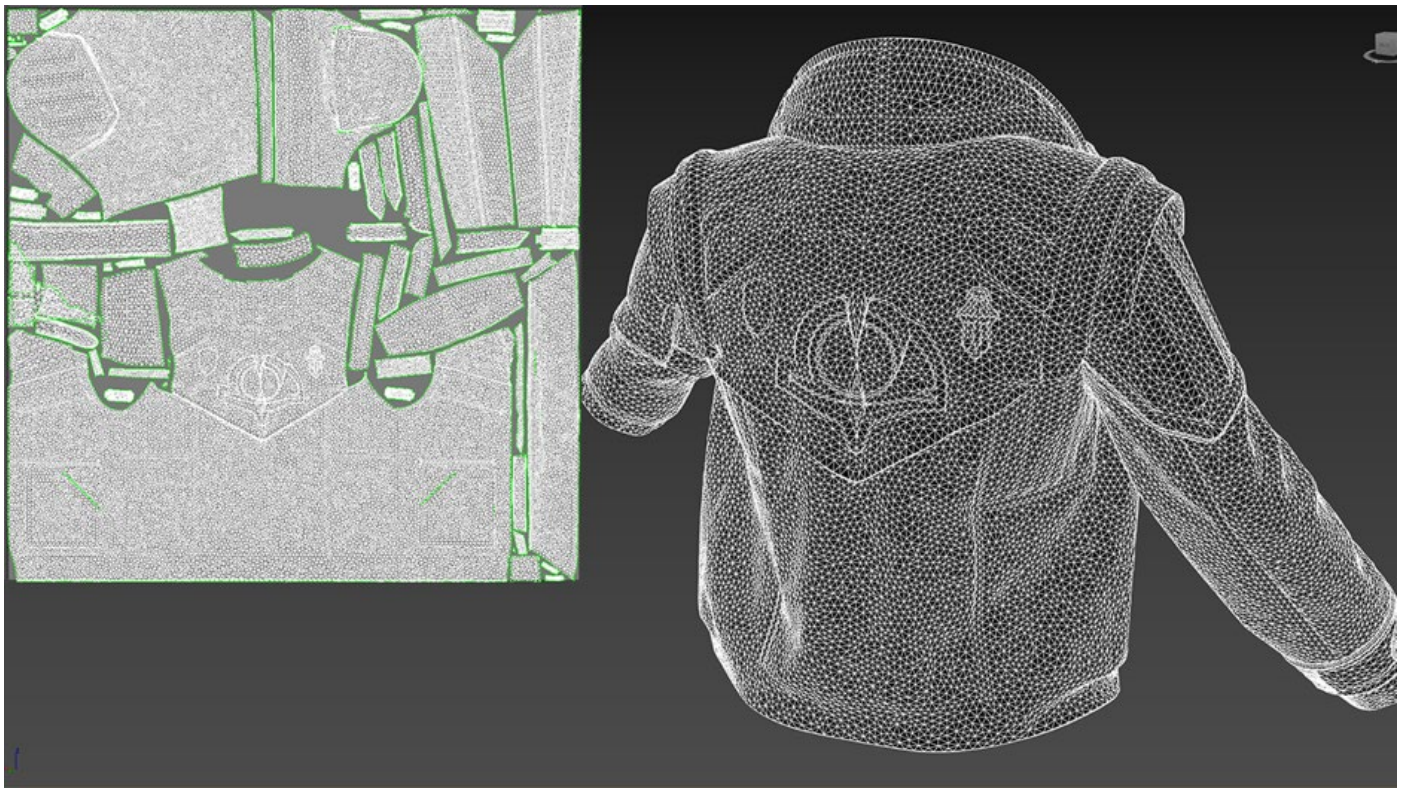


Cotton and Silk shaders can meet the needs of clothing or fabric.

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.

Render test of the fabric materials

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.



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

Appendix 2: Team workflows

Building games is a collaborative art. Unity can help your team create together, faster. Use source control to integrate changes and updates from everyone as they work.

The Unity teams and Accelerator services can also assist you with wrangling your developers and artists, either locally or on the cloud.

Source control

Unity has integrations with two version control systems: Perforce and Plastic SCM. Set the Perforce or Plastic SCM servers for your Unity project.

You can also use an external system, such as Git, including Git LFS (Large File Support) for more efficient version control of your larger assets, like graphics and sound resources. For the added convenience of working with the GitHub hosting service, install the GitHub for Unity plug-in. This open-source extension allows you to view your project history, experiment in branches, commit your changes, and push your code to GitHub without leaving Unity.

Unity also maintains a .gitignore file that can help you decide what should and shouldn't go into the git repository, and then enforce those rules.

Artists are usually the biggest workforce in game development.

Unity Teams

Unity Teams is another option for streamlining your team workflows. Unity Teams allows you to store your entire project in the cloud, so that it's backed up and accessible anywhere. This makes it simple to save, share, and sync your Unity projects with anyone.

Unity Accelerator

The Unity Accelerator removes waiting time by caching copies of your team's assets. This means that only one person needs to perform the actual import for the results to automatically be cached to Unity Accelerator. The next time a team member goes to import the same version of the asset, the Unity Editor will check the cache before starting the import process on their local machine.

Unity Build Server

Building your project can cut into your team's productivity. Consider offloading that process to network hardware using the Unity Build Server. This will help your creative team work on the project as often as needed, and allow them to make iterations more autonomously.

As your Unity project grows in size and complexity, generating a build consumes more and more time. If you are using your development workstations to build a project, you will lose productivity while your team waits for the build to complete.

Unity Build Server runs Unity in batch mode, exclusively for building Unity projects. Team members can request builds on demand, at their own pace. This reduces wait time for bug fixes and releasing new features for testing. Building on separate machines also minimizes each developer's downtime and allows everyone to iterate more quickly.

Editor tooling

UI Builder is a visually oriented tool that lets artists and designers create UI for the Unity Editor. It provides a visual set of tools for using the underlying UI Elements framework, including the stylesheet, hierarchy, and standard controls like buttons, scrollers, toggles, and text fields. Workflows in UI Builder are designed for rapid testing and iteration, and can provide a live and interactive preview of the UI as it is being created, so that the UI designer can see exactly how the final UI will look and feel.

Unity for artists

You can find additional art and technical tips on the Unity blog; start by searching the #unitytips hashtag on Unity community forums and Unity Learn.

The Unity Developer Tools microsite provides some of the best resources for developing with Unity, from documentation, the Knowledge Base, and Issue Tracker, to our latest roadmap and release information. Additionally, get further details on Unity subscriptions and additional products.

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


unity