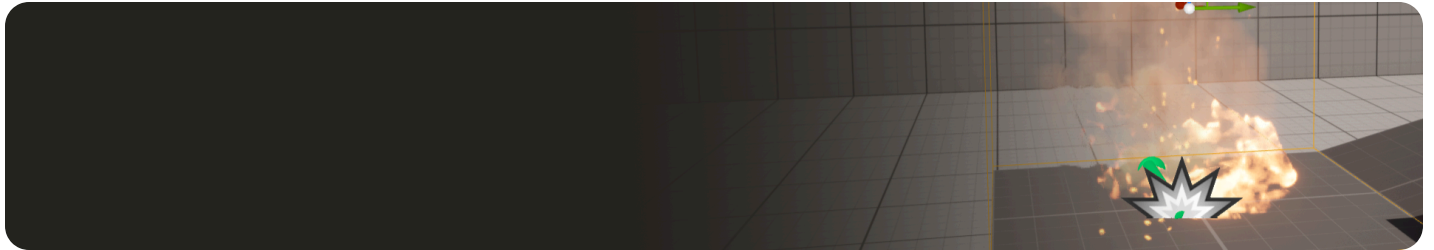


Volume Actors

Reference for the different kinds of Volume Actors in Unreal Engine.



Volumes are three-dimensional **Actors** that change the behavior and characteristics of the area they cover in a Level.

Volumes are support Actors, generally used to detect when certain Actor types have entered a specific area and trigger something in response. Volumes sometimes have built-in effects of their own (either in code or Blueprints).

You can use Volumes to implement effects and behaviors like:

- Causing damage to the player or other Actors inside the Volume.
- Blocking certain Actors from entering the Volume, acting as a collision surface.
- Changing something in the environment (for example, opening a door) when an Actor enters the Volume.
- Changing the lighting or visibility in a Level.

Placing Volumes

Place Volumes in a Level in the same way you place any other Actor:

1. In the **Main Toolbar**, click the **Create** button.
2. Select the **Volumes** category, then drag the Volume you want to place to the Level Viewport.

For more information, see [Placing Actors](#).

After you place a Volume in the Level, you can resize or reposition it. For more information, see [Transforming Actors](#).

Representing Volumes Visually

Volumes don't have a default visual representation when placed into a Level. In other words, a Volume Actor will be completely invisible at runtime (when your game runs). If you want the player to know that the Volume is there, you must add a visual representation to it.

For example, a **Pain-Causing Volume** causes damage to anything inside the Volume. You can use it to mark an area that you want the player to avoid, like poison, lava, or another environmental hazard.

If you place a Pain-Causing Volume into a Level, but don't add any visual representation to it, the player won't see it at runtime:


In-Editor	In-game/PIE

In this case, the player will not know to avoid the Volume, or understand why they take damage when they enter it.

You should provide a hint to the player that explains the Pain Causing Volume:

In-Editor	In-game/PIE

In this case, the fire particle effect provides a visual explanation of why the area is dangerous, while the Pain Causing Volume supports the visuals with the gameplay effect of decreasing the player's health during exposure to the fire.

 You can combine the Pain Causing Volume and fire particle effect into a single Actor to be able to move them and duplicate them easily.

Volume Types in Unreal Engine

This section describes some of the commonly used types of volumes in Unreal Engine. Note that this isn't an exhaustive list. Specialized volumes are described in detail in the documentation for their respective areas.

Collision and Overlap Volumes

Blocking Volume

You can use **Blocking Volumes** instead of collision surfaces on Static Meshes, particularly in the case of walls in structures. This can cause scenes to operate more predictably, since physical objects won't interact with small details like bumps on the floor and walls. It can also improve performance by reducing the cost of physics simulation.

For more information, see [Collision](#) documentation.

Camera Blocking Volume

Camera Blocking Volumes have pre-configured collision settings that block [cameras](#) and ignore everything else. They are intended to define invisible barriers that keep the camera out of unwanted locations.

For example, in a third-person game, the walls of the play area might have decorative coverings, like leafy vines. In this case, you can place a thin Camera Blocking Volume against the wall so that the camera doesn't bump into the vines or go behind the leaves, leaving it to slide smoothly and provide an unobstructed view of the player action.

Kill Z Volume

Use a **Kill Z Volume** to prevent objects from going out of bounds in certain types of games, such as falling off of a cliff or into a pit in a platformer game, or leaving a spaceship without a suit in a science-fiction setting.

The Kill Z Volume calls the `FellOutOfWorld` function on any Actor that enters it. By default, Actors will go through a quick cleanup procedure and then destroy themselves. You can override this behavior for any of your Actor types if your game requires something different. For example, if a key or other item that the player is required to collect in order to continue the game falls into a lava pit, your game might want to teleport the item back up to an area the player can reach rather than destroying it, or at least inform the player that the item was lost and reload the last checkpoint, rather than leaving the game in an unwinnable state.

Pain-Causing Volume

Pain-Causing Volumes, described above, have a set of configurable properties that you can use to specify:

- The type and amount of damage they cause
- How often they damage what's inside
- Whether they cause initial damage

Configure these properties in the volume's **Details** panel.

Physics Volume

If you want to configure the physical setup that affects characters and other physics objects, use **Physics Volumes**.

A common use for a Physics Volume is the creation of watery environments where the player needs to swim. The effects of Physics Volumes are visible and, paired with the appropriate visual representation, they are easy for the player to understand.

The Character Movement Component class uses the current fields to adjust how their owning `Character` moves through the environment. If your game has custom physics, derive your own child class from `APhysicsVolume`.

Trigger Volume

Trigger Volumes can cause events when a Player or other object enters or exits them. Use them with the [Level Blueprint](#) to test out events and gameplay scenarios or functionality without the need for additional Blueprints.

For example, you could place a Trigger Volume in your Level, then create an overlap event for the Volume in your Level Blueprint, which can play a sound, open a door, or start a cinematic scene.



Remember to check the collision settings to ensure that your trigger will react to the intended Actors with the Overlap collision response setting.

Graphics and Audio Volumes

Audio Volume

Audio Volumes add sound to the area they cover. There are two kinds of audio volumes you can use:

- Audio Volumes (legacy approach)
- [Audio Gameplay Volumes](#)

Cull Distance Volume

Cull Distance Volumes cause objects to be culled (not drawn to the screen) based on that object's distance from the camera and the object size. This can help optimize your scene by not drawing objects when they are small enough to be considered unimportant. Size is calculated by the bounding box along its longest dimension, and the cull distance chosen is the one closest to that size.

For more information, see [\[designing-visuals-rendering-and-graphics\rendering-optimization\visibility-culling\CullDistanceVolume\]](#).

Hierarchical LOD Volume

Hierarchical LOD Volumes are used by the [Hierarchical Level of Detail \(HLOD\)](#) system to group Actors into a single HLOD cluster. When generating clusters, the Unreal Engine overrides its normal generation process in deference to manually-placed Volumes.

Lightmass Volumes

Many maps have meshes out to the edge of the grid in the editor, but the actual playable area that needs high quality lighting is much smaller. Lightmass emits photons based on the size of the level, so those background meshes will greatly increase the number of photons that need to be emitted, and lighting build times will increase.

The **Lightmass Importance Volume** controls the area that Lightmass emits photons in, allowing you to concentrate it only on the area that needs detailed indirect lighting. Areas outside the importance volume get only one bounce of indirect lighting at a lower quality.

The **Lightmass Character Indirect Detail Volume** is similar to the Lightmass Importance Volume and generates indirect light samples, not just at Player height above ground, but inside the entire Volume. An example of how this type of Volume is used would be to place the Volume in an elevator shaft, ensuring that indirect lighting will be correct all the way along the shaft, instead of just at the bottom.

For more information, see [Lightmass Basics](#).

Mesh Merge Culling Volume

Mesh Merge Culling Volumes mark the mesh objects they contain so that these objects will be combined into a single mesh. This can improve performance by making a collection of small meshes in a contained area all cull together as one mesh, or by causing HLOD generation to more optimally reduce geometry.

For more information, see [Merging Actors](#)

Post Process Volume

A **Post Process Volume** can override Post Process settings applied to the camera inside the volume. You can use it to implement different kinds of visuals in different areas of your Level. For example, you may want a different depth of field (DoF) for indoor and outdoor areas, or different degrees of bloom depending on localized weather effects, like fog.

For more information and a list of effects you can use, see [Post Process Effects](#).

Precomputed Visibility Volume

Precomputed Visibility Volumes are used primarily for performance optimization. These Volumes store the visibility of Actors for their location in the world. Only place these in areas that the player can access.

For more information, see [Post Process Effects](#).

Precomputed Visibility Override Volume

Use **Precomputed Visibility Override Volumes** to manually override the visibility of Actors for their location in the world if the auto-generated result of a Precomputed Visibility Volume needs to be adjusted. These are also used for performance optimization and should only be placed in areas the Player can access.

For more information, see [Visibility and Occlusion Culling](#).

Level and AI Volumes

Level Streaming Volumes

Level Streaming Volumes are used to aid in the [Level Streaming](#) process. They provide a simple way to encapsulate a level, as well as control when it streams in and out of memory, based on when a Player enters or exits the Volume.

Nav Mesh Bounds Volume

Nav Meshes calculate navigation paths throughout the areas of a level. **Nav Mesh Bounds Volumes** are used to control where Nav Meshes will be built in a level.

Within the Volume, a Nav Mesh is constructed on all surfaces with an appropriate angle to be walked upon. You may overlap as many of these as you need to generate the desired Nav Mesh.

To use the Nav Mesh Bounds Volume, create one or more volumes that enclose the navigable areas of your Level. The Nav Mesh will be built automatically.



You can press **P** at any time in the viewport to visualize the Nav Mesh.

The [Content Examples](#) sample project demonstrates how to implement a Nav Mesh into your Level.