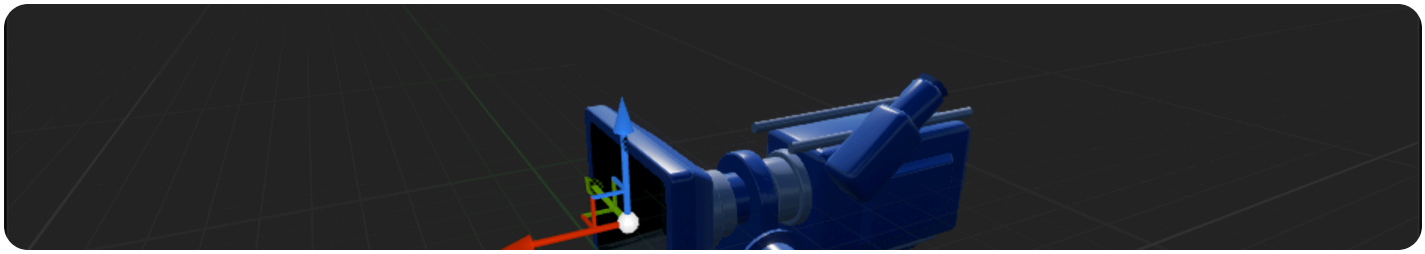
Camera

An overview of cameras

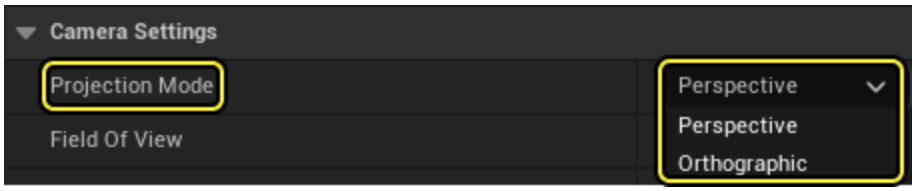


The **Camera** represents the player's point of view, such as how the player sees the world. For this reason, cameras only have relevance to human-controlled players. The PlayerController specifies a camera class and instantiates a Camera Actor (`ACameraActor`) which is used to calculate the position and orientation the player views the world from.

> For basic examples on how to work with Cameras, refer to Using Cameras. For an example of how to layer animations onto cameras, refer to the CameraAnim feature documentation.

CameraActor

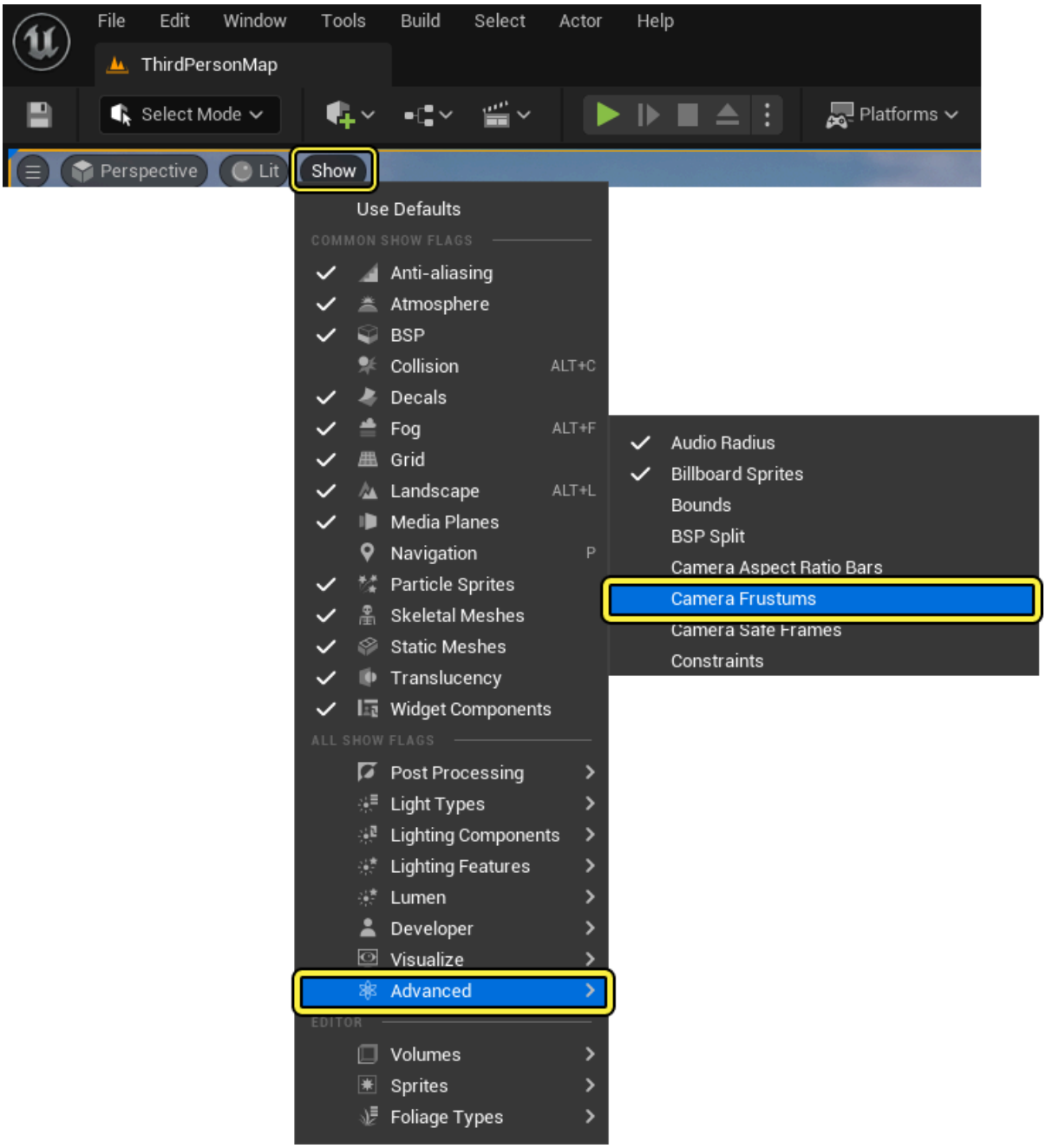
All of the camera's behavior and properties are set up in the CameraComponent. The `CameraActor` class primarily acts as a wrapper for the CameraComponent, so that the camera can be placed directly in the level rather than within another class. When using a CameraComponent in the Editor, You can navigate to **Details > Camera Settings** to set whether the camera is in **Perspective** or **Orthographic** mode.



The vertical field of view (FOV) can be set for perspective mode, and the width in world units can be set for orthographic mode. For both modes, the aspect ratio can be designated and preset aspect ratios for common devices and display types are provided.

> You can add Post process effects to the camera, and it is possible to scale the strength of the post process effects.

Two components are added to the CameraComponent to assist with visual placement in the Editor, although they will not be visible during gameplay. A **FrustumComponent** shows where the camera field of view is located. This does not show by default, but can be turned on by selecting **Show > Advanced > Camera Frustums** in the **Viewport**.



CameraComponent

A **CameraComponent** represents a camera viewpoint and settings, such as **Projection Type**, **Field Of View**, and **Post-Process Overrides**. If the ViewTarget is a CameraActor or an Actor that contains a CameraComponent and has `bFindCameraComponentWhenViewTarget` set to *true*. A related property that can be set for any Pawn is `bTakeCameraControlWhenPossessed`, where the Pawn will automatically become the ViewTarget upon possession by the PlayerController.

Actors and PlayerControllers

PlayerControllers and Actors contain a `CalcCamera` function. An Actor's `CalcCamera` function returns the camera view of the first CameraComponent in the Actor, if `bFindCameraComponentWhenViewTarget` is *true* and a CameraComponent is present, otherwise it gets the Actor's location and rotation. In the PlayerController class, the CalcCamera function behaves similarly, returning the location of the possessed Pawn if it exists, and the control rotation of the PlayerController.

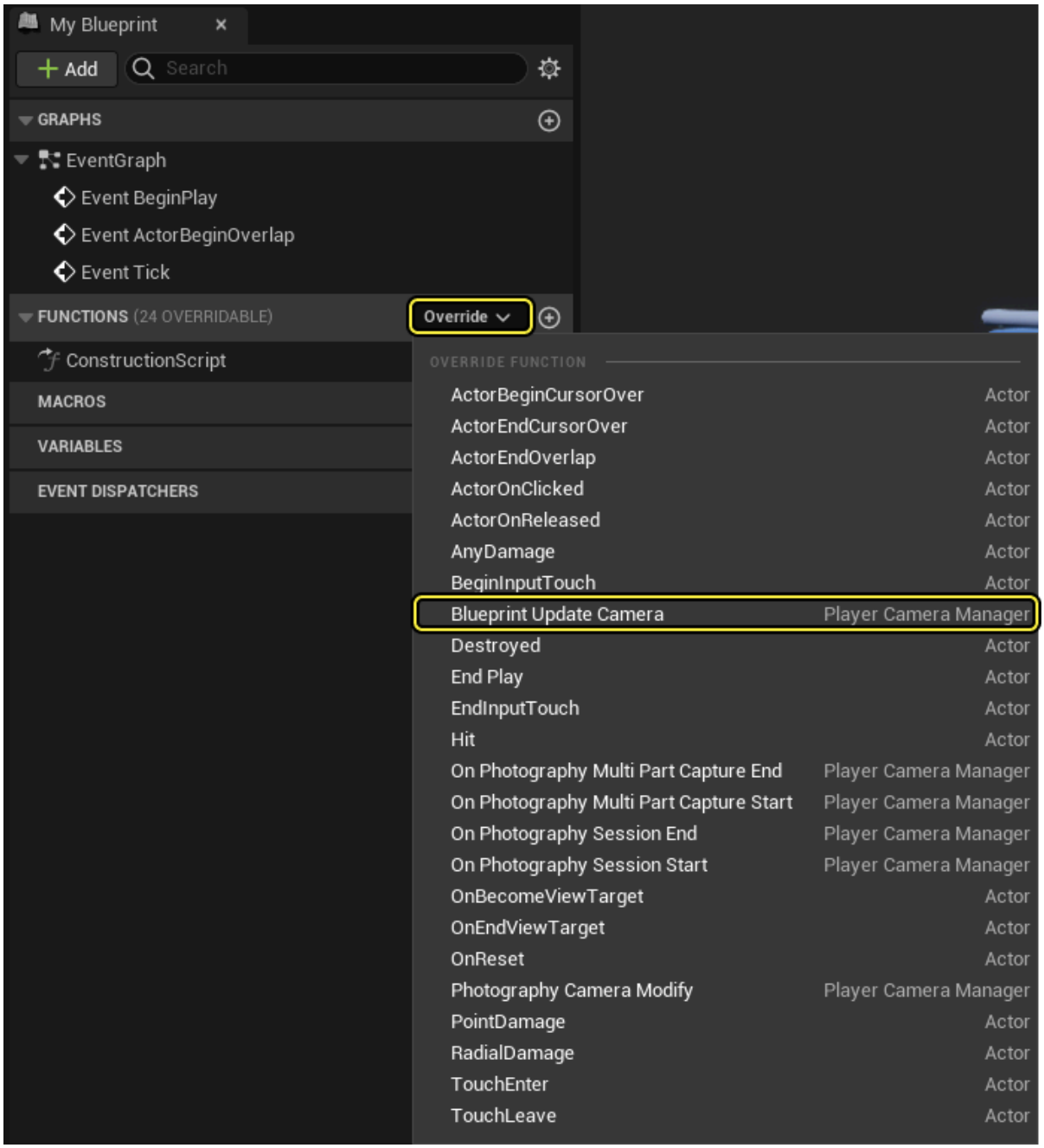
PlayerCameraManager

The **PlayerCameraManager** is responsible for managing the camera for a particular player. It defines the final view properties used by other systems like the Renderer. The PlayerCameraManager can function as a "virtual eyeball" to see the world. It can compute the final camera properties directly, or it can blend between other objects or Actors that influence the camera (blending from one CameraActor to another.)

The PlayerCameraManager's primary external responsibility is to reliably respond to various `Get()` functions, such as `GetCameraViewPoint`. By default, a PlayerCameraManager mantains a view target, which is the primary actor the camera is associated with. It can apply various post effects to the final view state, such as camer animations, post-process effects or special effects such as dirt on the camera lens.

If you need to inherit from the PlayerCameraManager, and you are creating a parent class Blueprint instead of C++, then the PlayerCameraManager contains the `BlueprintUpdateCamera` function to provide your own custom camera implementations.

> When using this function, return *true* to use the returned values, or return *false* to ignore them.



The **UpdateViewTarget** function in PlayerCameraManager queries the ViewTarget and returns that ViewTarget's Point Of View. This function is what calls the BlueprintUpdateCamera function if you have subclassed APlayerCameraManager and are not looking through a CameraComponent.

ViewTarget

The **ViewTarget** struct, defined in PlayerCameraManager, is responsible for providing the PlayerCameraManager with an ideal Point of View (POV). ViewTarget contains information on the target Actor, the Controller of the target Actor (for non-locally controlled Pawns), and the

PlayerState, which is used to follow the same player through Pawn transitions and other changes while spectating.

The camera information passed to PlayerCameraManager through the POV property is in the form of a `FMinimalViewInfo` struct. This struct contains the basic camera information from a CameraComponent, including the **Location**, **Rotation**, **Projection Mode** (Perspective or Orthographic), **FOV**, **Orthographic Width**, **Aspect Ratio**, and **Post Process Effects**. Providing the PlayerCameraManager with access to these values allows the PlayerCameraManager to blend between two camera modes during its camera management.

Camera Responsibility Chain

Game-specific camera behavior can be provided at any point along the camera "responsibility chain", which flows from top to bottom through the following classes before passing to `ALocalPlayer` and ending with Rendering, **Scene View** (`FSceneView`), and other related systems.

Camera Topics

Camera Animations

An overview of camera animation

Orthographic Camera

An overview of setting up a camera with Orthographic projection.

Using Cameras

A How To Guide for Finding Actors in your Scenes in Unreal Engine.