CAMERA CONTROLLER

10/11/2016

Demo Quick Start

This quick start is simply to get the demo up and running in a stand-alone project.

1. Start a new Unity 5 Project.

2. Download and import the Camera Controller asset.

3. Open one of the demo scenes.
...\Assets\ootii\CameraController\Demos\Scenes\

4. Press play.

DEFAULT CONTROLS

WASD	= Move
RMB	= Rotate
LMB	= Orbit
MMB	= Aim

NEVER make the Camera Controller a child of the player

CAMERA CONTROLLER

10/11/2016

Custom Quick Start

This quick start assumes you have a project and you want to use the Camera Controller to manage your camera.

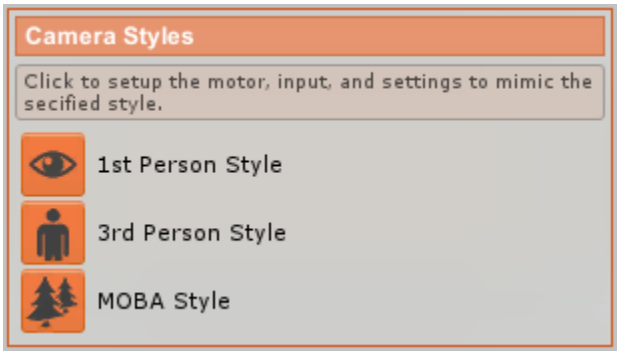
1. Open your Unity 5 Project.

2. Download and import the Camera Controller asset.

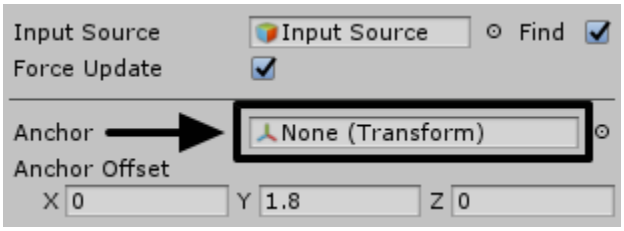
3. Add a new GameObject to the root of your scene.

4. Add a “Camera Controller” component to the new GameObject.
Press the ‘Components | ootii | Camera Rigs | Camera Controller’ menu item.

5. Select the camera style.



6. Set the character that the camera will follow.



7. Set options as desired.

NEVER make the Camera Controller a child of the player

CAMERA CONTROLLER

10/11/2016

Foreword

Thank you for purchasing the Camera Controller!

I'm an independent developer and your feedback and support really means a lot to me. Please don't ever hesitate to contact me if you have a question, suggestion, or concern.

I'm also on the forums throughout the day:
https://forum.unity3d.com/threads/aaa-quality-adventure-camera-released.218776/page-1000

Tim
tim@ootii.com

Overview

The Camera Controller does everything the Adventure Camera did and more. In addition to being a AAA quality 3rd person camera, the Camera Controller is also a 1st person camera, a MOBA camera, a cut-scene camera, and more. Using customizable "motors", the Camera Controller drives the scene camera to create a smooth and feature-rich experience for your players. By enabling and disabling motors, the Camera Controller can fit any 3D game.

Like my other assets, the Camera Controller is built to be expanded on. In addition to using the out-of-the-box motors, you can create your own motors for managing the camera's position and rotation. You can then use the Transition Motor to blend between different camera motors.

Contents

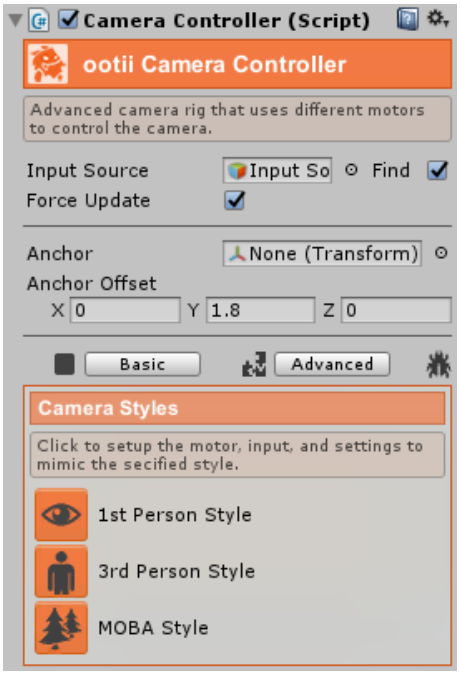
Introduction

The Camera Controller is composed of a couple different pieces. Understanding how these pieces fit together will make using and extending the asset much easier.

Camera Controller

This is the main component and the manager of the camera motors. The Camera Controller not handles the editor GUI, but it also makes sure that motors run when they are supposed to.

Core functionality such as collision detection and field-of-view zooming are done in the Camera Controller and not in the motors.



The Camera Controller has two views: Basic and Advanced.

The basic view is used for quick setup using default values. Under the hood, the basic view simply adds camera motors for you. It then sets default values on those motors to create the behavior you chose.

The advanced view allows you to access the camera motors directly. Here, you can change values, add motors, etc.

Input Sources

If everyone used the same input solution, we wouldn't need this. However, some people use Unity's native input solution, some people like Easy Input, and some people use other assets found on the Asset Store.

That means we have to have a generic way of getting input so motions can be coded once and grab user input from any solution. That's what an Input Source is; a generic way of getting input. By wrapping your preferred input solution in an "input source", the MC and motions themselves can use the input source to tap into your chosen input solution.

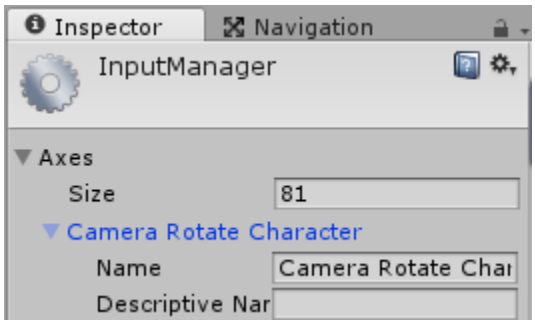
The Camera Controller includes the 'Unity Input Source' which knows how to grab user input using Unity's native solution. Easy Input includes an 'Easy Input Source' and you can create other 'Input Sources' to work with your favorite solution as needed.

Click here to learn more about creating Input sources.

Action Aliases

Throughout the motors, I use 'action aliases' to check if input has occurred and if it's time to activate a motor. An action alias is just a friendly name that's given to a key press or input condition that allows us ask about it without understanding how it works.

Let's look at an example…



The 3rd Person Fixed motor has an action alias property whose default value is "Camera Rotate Character". The motor uses this value to see if we should be rotating the anchor along with the camera:

```
bool lShouldRotate =
InputSource.IsJustPressed("Camera Rotate
Character");
```

The motor doesn't really care how the input was processed. It just cares that a value of `true` or `false` comes back. This is exactly what Unity does with its input system (see above).

Camera Motors

Camera Motors are the work horses under the work horses under the hood. They are responsible for moving and rotating the camera based on input or the environment.

The Camera Controller comes with 7 built in motors:

1st Person View – Motor is used for 1st person games and represents the player's eyes.

3rd Person Fixed – Used with 3rd person games, the motor will move like it's attached to the player with a pole. This motor is great for 3rd person shooters.

3rd Person Follow – Used with 3rd person games, it drags behind the player like it's attached by a rope. This motor mimics modern 3rd person adventure games.

Fixed Motor – This motor has a set position and rotation that doesn't change.

Spline Motor – Allows you to setup a path and the motor will follow the path. This is great for cut-scenes, finishing moves, or providing visual clues in-game.

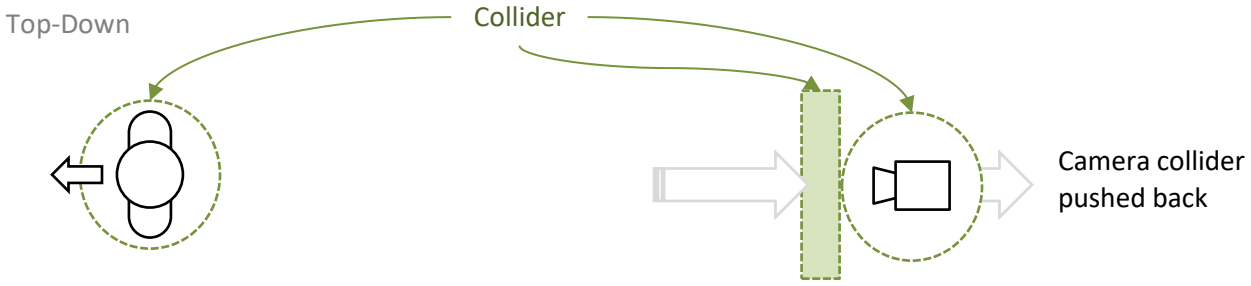
Top Down View – Provides camera movement similar to what you'd find in modern MOBAs or strategy games. This motor is provides top-down experience and could be used for Diablo style games as well.

Transition Motor – This motor allows us to blend from one motor to another. It's great for creating different views based on user input, cut-scenes, etc.

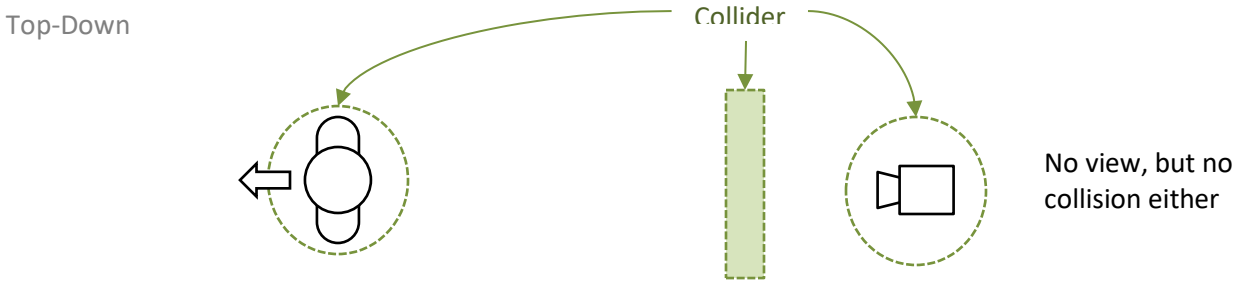
Collisions

The Camera Controller uses "line-of-site" obstruction for collision handling. This means that collisions aren't handled like physics objects. There's a couple of reasons for this:

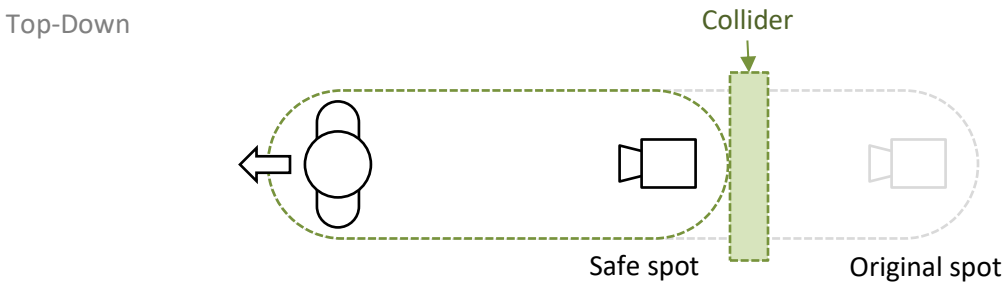
1. With typical physics objects, a cube could push a camera's sphere collider really far away from the character or anchor. If we allowed typical physics collisions, the player would lose control of the camera.



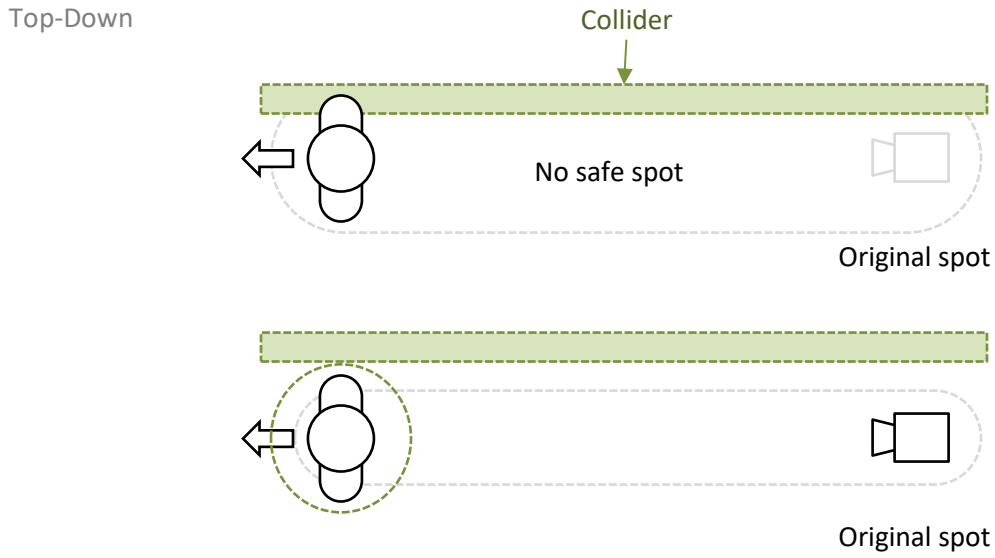
2. If an object comes between the camera and its target, it's possible that no collision actually occurs. So, there would be no physics reaction.



So, instead I use line-of-sight collisions. This means that we test the ability of the camera to see the target or anchor. If something blocks the view, the camera moves forward to the closest "safe" spot.



For best results, you really want to ensure the character's collider is wider than the camera's collision radius. This way we don't get into situations where the camera test skims a wall and returns invalid results for the whole distance.



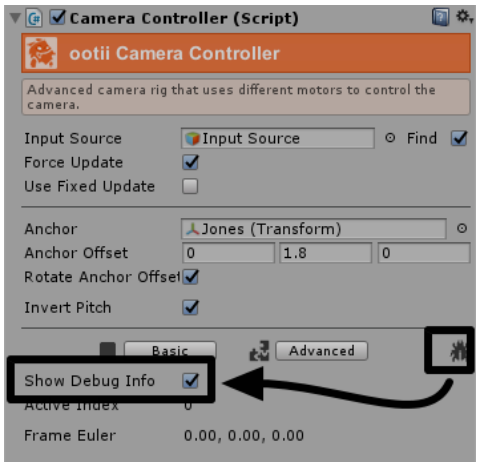
The camera will do its best to handle the situation where there is no safe spot, but preventing the odd situation is best.

Debugging

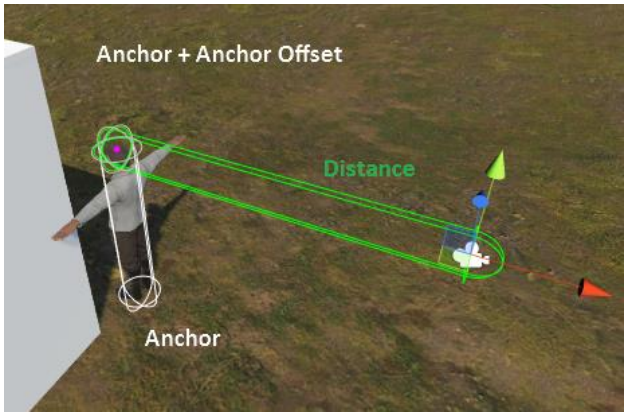
The Camera Controller supports a debug mode that shows you how the anchor and offsets fit into the picture during runt-time. It's important to understand the offsets and you can learn more under Motor & Motor Properties. However to enable debugging, do the following:

1. Add the "Graphics Manager" script to the "Main Camera" in your scene.

2. Check the “Show Debug Info” checkbox under the debug icon.



When you do this, you can visualize the anchor, offsets, and focus point.



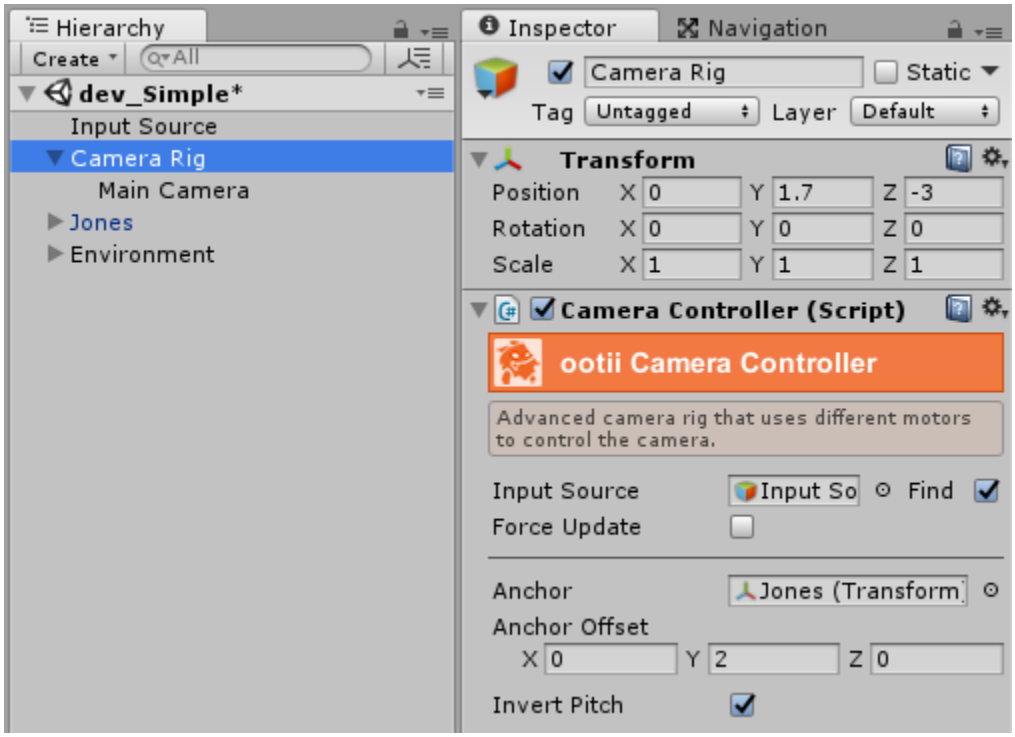
Camera Controller Setup

Initial Setup

When you follow the steps on page 3, you'll be adding the Camera Controller to a new GameObject. You NEVER want to add the Camera Controller as a child of your character (even for a first-person setup).

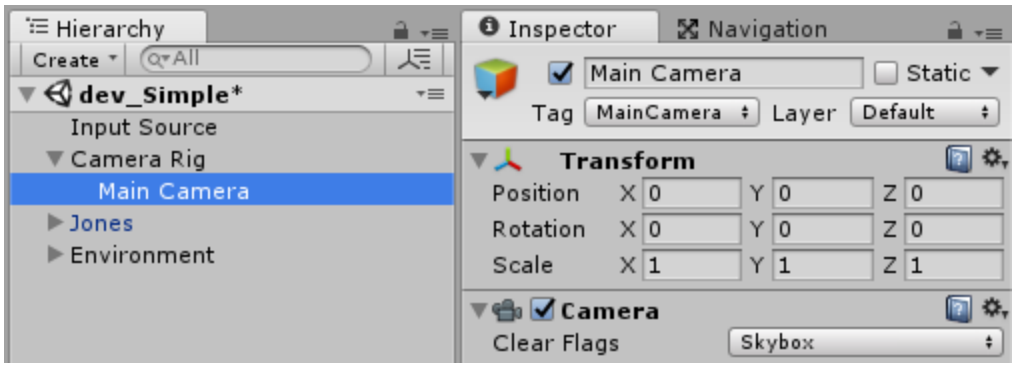
> NEVER make the Camera Controller a child of the player

After following the quick setup, your hierarchy should look something like this:



I named the GameObject "Camera Rig" since it is what moves the Main Camera around. That's what the Camera Controller is added to.

Think of the Main Camera GameObject as the lens. Its transform needs to look like this:



By totally clearing out the Main Camera's transform, it will simply follow the Camera Rig (Camera Controller GameObject) as it moves and rotates.

You would then set the Input Source and Anchor properties as appropriate. The Input Source is how the Camera Controller gets user input like key presses and mouse movement. The Anchor is the object that the camera is following. Typically this is your character.

Force Update

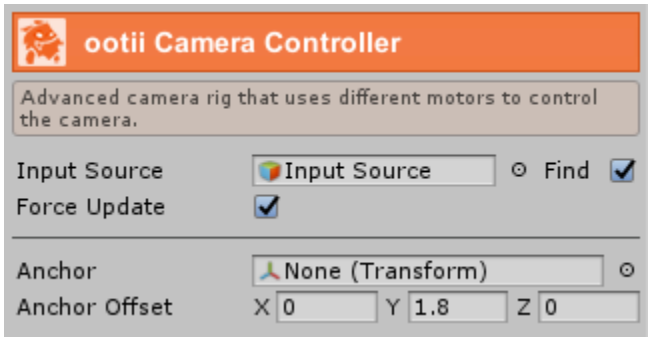
This property typically isn't set by you directly.

This property determines if the camera runs its own update cycle or if it uses an update cycle from an ootii character controller. If you're using an ootii character controller, this check box will become un-checked. If you're not, it should be checked.

The reason for this distinction is that we want the camera to run after the character controller. When using an ootii character controller, we run a separate update cycle that ensures it is.

Key Properties

The Camera Controller has a couple of key properties that need to be set:



Input Source – This is how we'll get input from the mouse, keyboard, or controllers. Set the Input Source that the camera will use to determine when to rotate, zoom, etc.

See Input Sources for more information.

Force Update – This property typically isn't set by you directly.

This property determines if the camera runs its own update cycle or if it uses an update cycle from an ootii character controller. If you're using an ootii character controller, this check box will become un-checked. If you're not, it should be checked.

The reason for this distinction is that we want the camera to run after the character controller. When using an ootii character controller, we run a separate update cycle that ensures it is.

Anchor – This is the character that the camera is following. You don't always have to have an anchor, but typically you will.

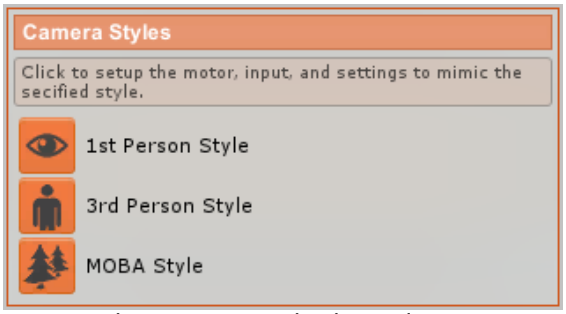
Anchor Offset – Typically the character's root is at its feet. Most cameras want to follow the head. So, we can use this offset to represent the final anchor point the camera will follow.

Note that some motors may use different offsets. In those cases, they have motor-specific properties. However, even those offsets will be based off the Anchor + Anchor Offset that you set here.

Invert Pitch – This option provides a global way of setting the invert for all motors that yaw and pitch from input.

Camera Styles

The Camera Styles section on the Basic view is meant to help with quick setup. They add the required motors, set properties, and help get you started.



Once the style is setup, you can then tweak the motor properties to have the camera behave how you want.

1st Person Style sets up standard first person camera behavior.

3rd Person Style sets up standard third-person camera behavior that mimics games like Tomb Raider.

MOB Style sets up standard top-down camera behavior.

Options

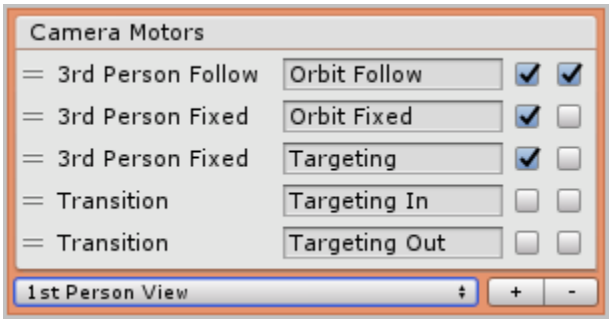
Once you choose a basic camera style, style specific options will appear. Again, these are just for quick setup and in the end they modify the motor properties directly.

You can use the basic view to setup your camera or you can go directly to the advanced view.

Camera Motors

The Camera Motors list is where you'll manage most of your properties. Here, you'll add and remove motors as needed. You may find that a motor type will be added multiple times with different properties. This allows you to setup all styles of gameplay.

List Columns:



Grab Bars – Bars to select and move list items.

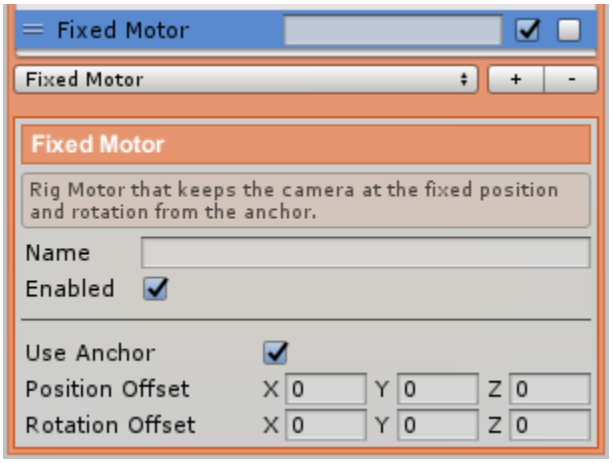
Motor Type – Type of motor the item represents.

Motor Name – Friendly name you can give the motor. You can use this to search for motors using code.

Is Enabled – Determines if the motor can be activated.

Is Active – Determines if the motor is active. Only one motor can be active at a time.

Use the dropdown and +/- buttons to add and remove items from the list. You can also re-order the list by clicking the 'grab bars' and dragging the list item up and down.



When a motor is selected, the motor specific properties will be listed.

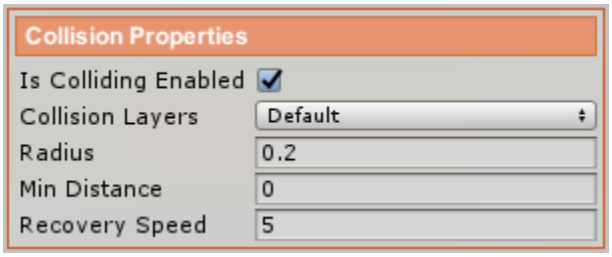
For the Fixed Motor selected to the left, you can see that the properties allow us to change the anchor, add an addition position offset, and add an additional rotation offset.

To learn about each motor's properties, see the Motor Properties section.

Collision Properties

Collision properties are managed by the Camera Controller itself and not the individual motors. So, these properties live outside of the motor list.

The camera uses a line-of-sight obstruction approach. In this way, if an object comes between the camera and the anchor, the camera will move closer to the anchor in order to prevent the view from being blocked. This is slightly different than traditional collision detection which just puts a sphere around the camera… this approach mimics what you'd find in AAA games.

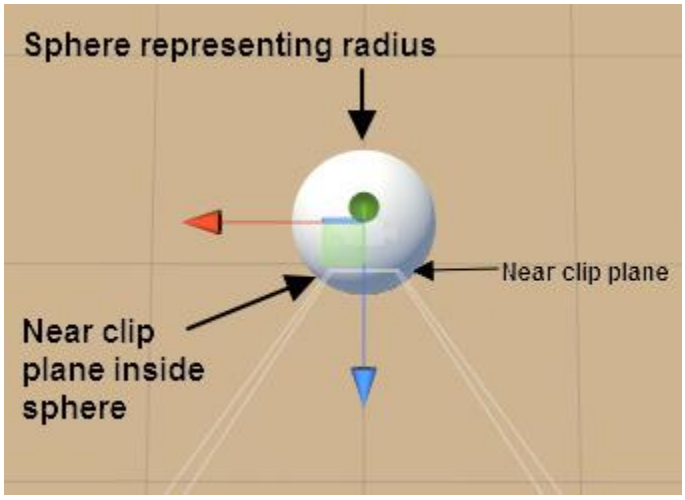


Is Colliding Enabled – Enables or disables collision

Collision Layers – GameObject layers that the camera will collide with.

Radius – This radius represents the size of the collision test area.

It's important that the radius is large enough to cover your camera's near clip plane. If it doesn't, you could get clipping as the near clip plane may penetrate walls and other objects.



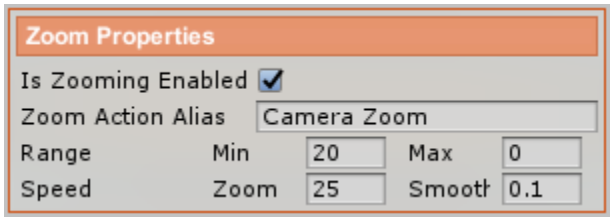
Min Distance – Minimum distance to the anchor the camera can get. In some cases, you may not want the camera to get right onto the anchor even if there is a collision. You can set that distance here.

Recovery Speed – When the view is obstructed, the camera moves closer to the anchor. If the view is no longer obstructed, we don't have to 'pop' back. Instead, we can use this speed to have the camera return to its normal position in a slower and smoother way.

Zoom Properties

The Camera Controller uses the standard field-of-view approach for zooming in. This approach is similar to how telescopes work as it doesn't move the camera closer to the target, but instead the lens changes its field-of-view which magnifies the view.

This approach works well as it doesn't move the camera. If we move the camera forward. If we move the camera forward, we risk obstacles coming between the character and the camera.



Is Zooming Enabled – Determines if we can zoom.

Zoom Action Alias – Input alias used to zoom in and out. By default, 'Camera Zoom' is tied to the mouse wheel.

Range – Determines the minimum field-of-view (most zoom) and max. The max is typically the default field-of-view. Setting the value to 0 will have it set automatically when the game loads.

Speed – Determines how quickly we zoom in and out.

CAMERA CONTROLLER

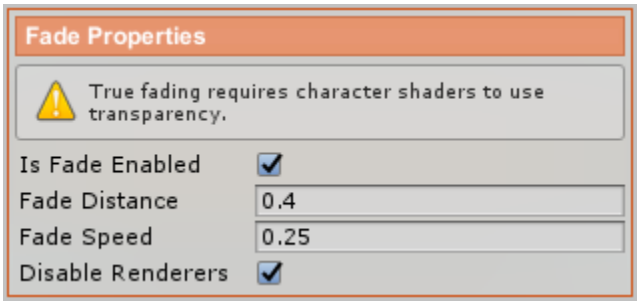
10/11/2016

Fade Properties

The fade properties will cause your character to fade out and become invisible when the camera is too close to the character.

At the end of the fade speed, the renderers can be disabled to prevent any character meshes from being rendered.

It should be noted that meshes will only “fade gracefully” if they have transparent materials associated with them. If the materials are opaque, the meshes won’t be able to fade. However, they can still have their renderers disabled.



Is Fade Enabled – Determines if fading will occur or not.

Fade Distance – Is the distance at which fading occurs.

Fade Speed – Speed (in seconds) that says how quickly the alpha values change to fade in and out.

Disable Renderers – Determines if we disable the renderers once fading out has completed.

Shake Properties

Camera shaking is done through code by calling a function on the Camera Controller component.

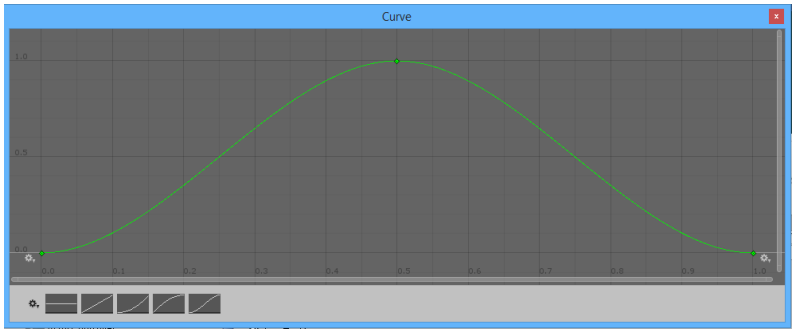
By grabbing a reference to the Camera Controller, you can make the following call:

```
mCameraController.Shake(0.02f, 1f, 0.5f, 1f);
```

The “Shake” function has the following signature:

- Range – How far from the center the random movement will extend.
- Strength X – Multiplier for movement on the x-axis (left/right).
- Strength Y – Multiplier for movement on the y-axis (up/down).
- Duration – Time in seconds we’ll shake for.

In the properties, you probably noticed the **Shake Strength** curve. This curve allows you to specify the strength (or multiplier) that is applied over the duration of the shake. When no curve is set (meaning it is a float 0), I’ll turn that into a basic curve:



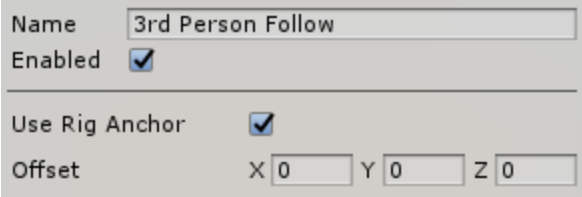
In this way, the strength will go from 0 at the beginning, to 1 in the middle, and back to 0 at the end. This multiplier to the randomized shake value creates a smooth ramp-up and ramp-down to the shake.

You can change the curve to create a more abrupt shake.

Motors & Motor Properties

Each motor controls the camera in a different ways. The properties associated with the camera help define how the motor behaves.

That said, there are some properties that are common to most all motors.



Name – Friendly unique name you can set for the camera motor. This is useful for searching for the camera motor in code.

Enabled – Determines if the motor can be activated.

Use Rig Anchor – Typically, you'll use the Camera Controller's Anchor and Anchor Offset property. These are globally set for all motors. However, if you uncheck this property you can set a custom Anchor and Anchor Offset for this motor.

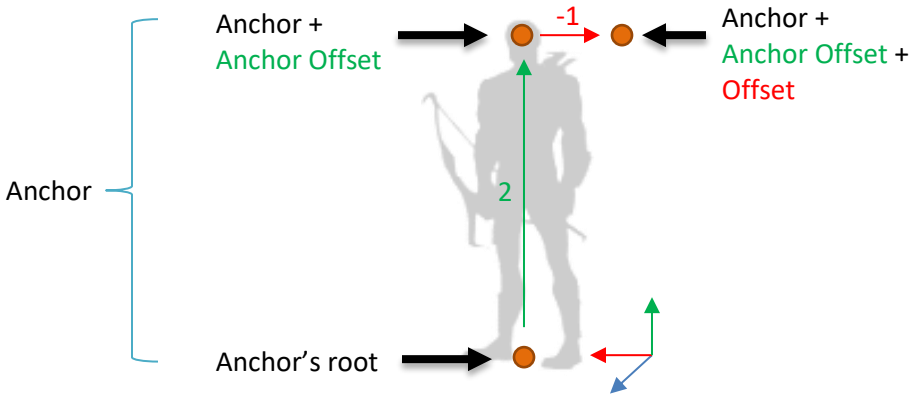
Offset – In addition to the 'Anchor Offset', you can set an offset specific to this motor.

Offsets

One of the advanced features of this camera's motors is the ability to offset from the Anchor + Anchor Offset. This is important as it allows for true AAA 3rd person camera behavior.

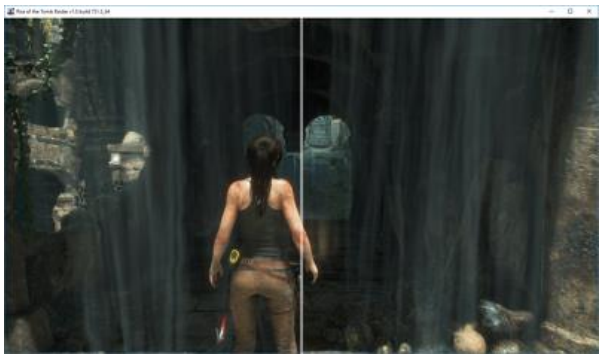
This graphic should help to clarify the different anchor and offset properties. Let's assume:

```
AnchorOffset = { 0, 2, 0 }
Offset = { -1.0, 0, 0 }
```

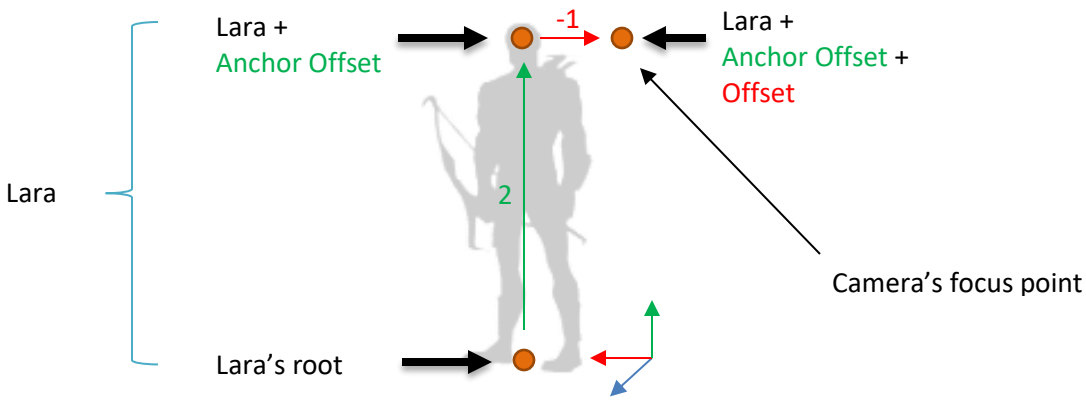


Let's look at Rise of the Tomb Raider…

The images below show Lara in the same scene, but the camera orbited around her. The white line is the center of the screen. You can easily see that the camera isn't directly behind Lara. Instead, there's an offset.

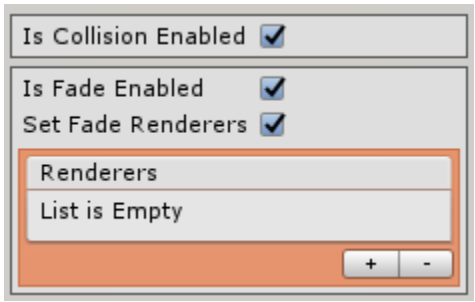


This offset is respected no matter how the camera orbits around Lara. So, thinking about the previous explanation, we can think of the camera setup like this:



Collisions and Fading

While the master collision and fading flags are set on the Camera Controller as described above, each motor also has flags to determine if the features will be enabled for that specific motor



In the case of fading, you can choose to have all renderers fade (the default) or check the **Set Fade Renderers** checkbox to specific which renderers are faded by that specific motor.

When the **Set Fade Renderers** checkbox is checked, the Renderers list is show. Here you can specify exactly which transforms' renderer will fade in and out.

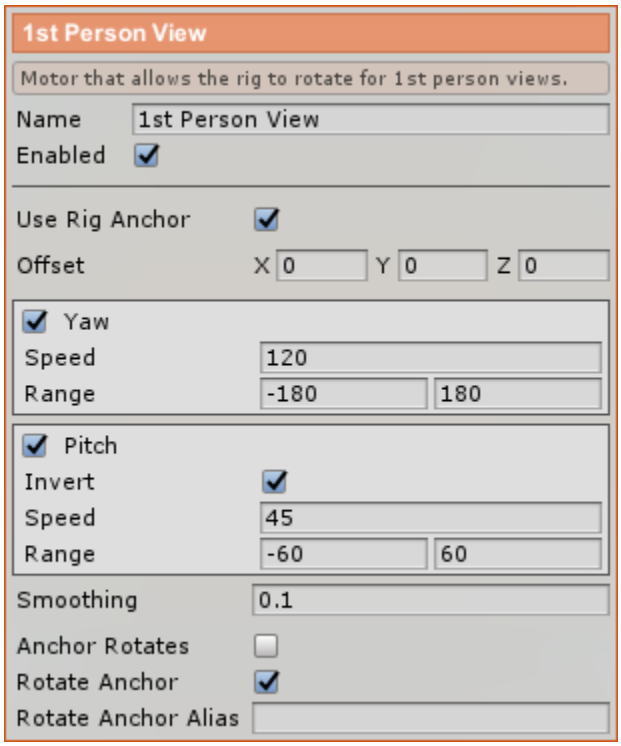
While specifying specific renderers to fade per cameras is useful. However, if you're trying to mix 3rd person and 1st person camera, you'll probably need two specific models for your characters.

CAMERA CONTROLLER

10/11/2016

1st Person View Motor

The 1st person motor mimics traditional first person games. It allows the camera rotate as if they were the character's (anchor's) eyes.



Yaw is the rotation around the character's up axis.

Pitch is the rotation around the camera's right axis.

Speed – Determines how quickly the camera rotates around the axis.

Range – Sets a limit to how far the camera can orbit around each axis. For yaw, using -180 and 180 means there is no limit. For pitch, there is always a limit in order to prevent the camera from flipping upside down.

Invert – Inverts the vertical input values so pitch reacts different.

Smoothing – Provides a velocity based smoothing to the orbit in order to create smoother rotation. Good values are between 0 and 0.3.

Anchor Rotates – Used to determine if the character controller rotates the character (anchor). If so, we need to do some post processing to stay in sync.

Rotate Anchor – Determines if we'll rotate the anchor along with our yaw.

Rotate Anchor Alias – Input alias that determines if the anchor rotation occurs. This way, we only rotation if something like the right-mouse-button is pressed. Typically you'd leave this value blank to mimic games like Rainbow Six as you always want the character rotating with the camera.

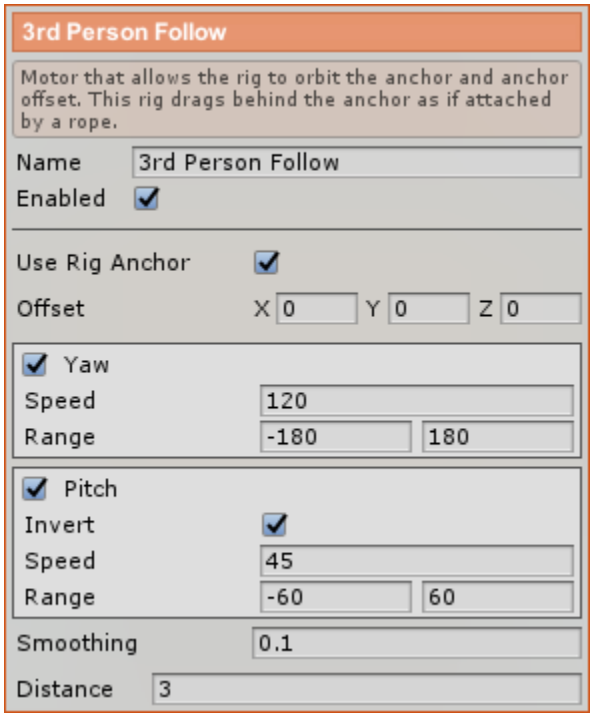
If you are using the **Motion Controller**…
Use the **Walk Run Strafe** motion with this motor.

CAMERA CONTROLLER

10/11/2016

3rd Person Follow Motor

The 3rd person follow camera works similar to cameras in AAA adventure games like Tomb Raider and Shadows of Mordor. The camera will drag behind the character (anchor) as if attached by a rope. What this means is that the character can rotate around the camera.



Yaw is the rotation around the character's up axis.

Pitch is the rotation around the camera's right axis.

Speed – Determines how quickly the camera rotates around the axis.

Range – Sets a limit to how far the camera can orbit around each axis. For yaw, using -180 and 180 means there is no limit. For pitch, there is always a limit in order to prevent the camera from flipping upside down.

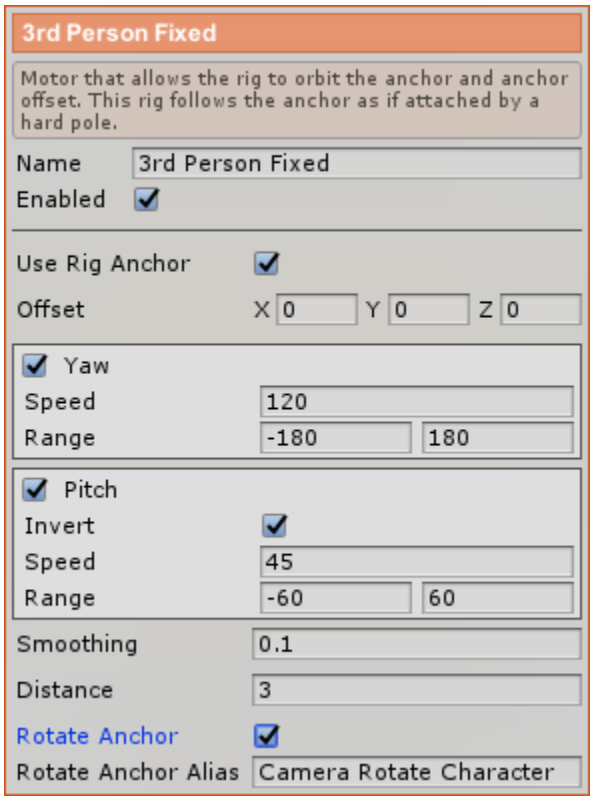
Invert – Inverts the vertical input values so pitch reacts different.

Smoothing – Provides a velocity based smoothing to the orbit in order to create smoother rotation. Good values are between 0 and 0.3.

Distance – Desired distance from the camera's focus point.

3rd Person Fixed Motor

Similar to 3rd person follow, this camera provides a AAA experience similar to what you'll find in 3rd person shooters and holder adventure games. With this motor, the camera will move behind the character (anchor) as if attached by a metal pole. This means as the character strafes left or right, the camera strafes instead of rotating.



Yaw is the rotation around the character's up axis.

Pitch is the rotation around the camera's right axis.

Speed – Determines how quickly the camera rotates around the axis.

Range – Sets a limit to how far the camera can orbit around each axis. For yaw, using -180 and 180 means there is no limit. For pitch, there is always a limit in order to prevent the camera from flipping upside down.

Invert – Inverts the vertical input values so pitch reacts different.

Smoothing – Provides a velocity based smoothing to the orbit in order to create smoother rotation. Good values are between 0 and 0.3.

Distance – Desired distance from the camera's focus point.

Rotate Anchor – Determines if we'll rotate the anchor along with our yaw.

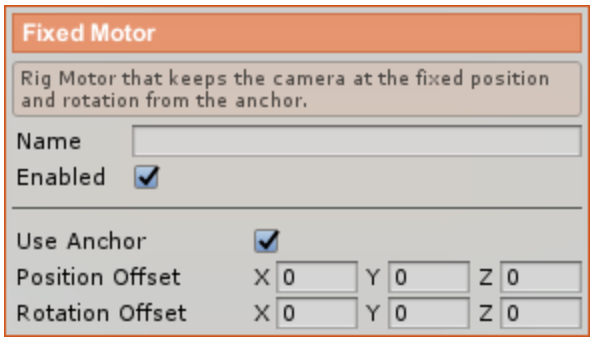
Rotate Anchor Alias – Input alias that determines if the anchor rotation occurs. This way, we only rotation if something like the right-mouse-button is pressed.

CAMERA CONTROLLER

10/11/2016

Fixed Motor

This motor is one of the simplest motors. It just provides a fixed position and rotation for the camera.

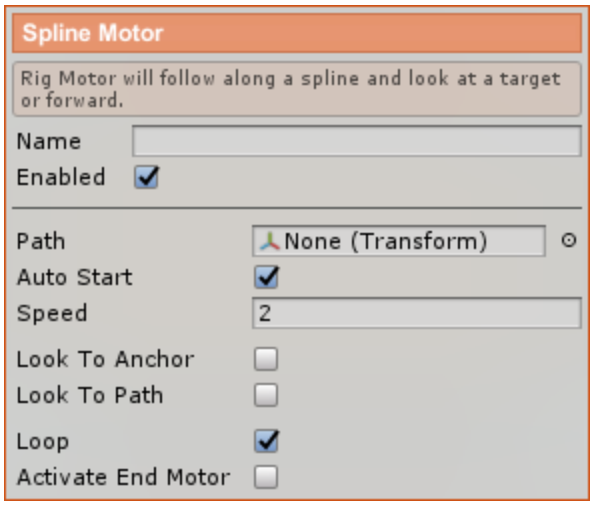


Position Offset – Offset from the Anchor and Anchor Offset.

Rotation Offset – Offset from the Anchor's rotation.

Spline Motor

The spline motor is used to create a camera that follows a path. When activated, the camera will move along the specified path and transition to another motor or simply continue looping.



Path – GameObject that contains the Spline object that is the path.

Auto Start – Determines if the camera starts moving on the path as soon as the motor is activated.

Speed – Units per second that the camera moves.

Look To Anchor – Determines if the camera rotates to always face the anchor as it moves.

Look To Path – Determines if the camera rotates to always face the direction of movement.

Loop – Determines if the camera restarts the path once it completes.

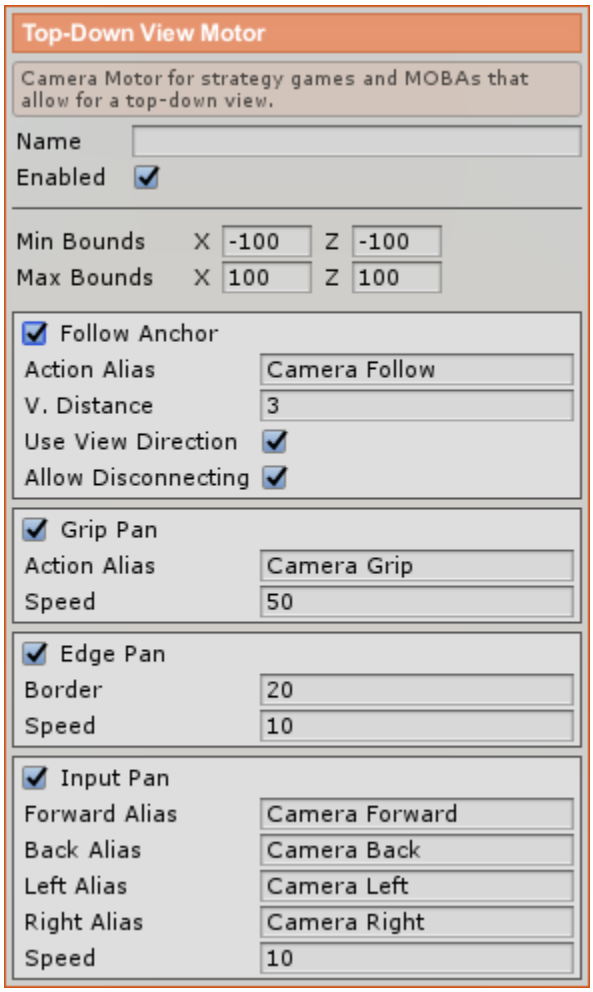
Activate End Motor – Determines if a new motor is activated once the path completes.

Index – Index of the motor to activate when Activate End Motor is checked.

Go here to learn more about splines.

Top-Down View Motor

This motor is used for MOBAs, strategy games, and top-down action games like Diablo. It provides a 'world view' and can be panned different ways:



Min Bounds – Minimum world bounds the camera can move to.

Max Bounds – Maximum world bounds the camera can move to.

Anchor Following

Follow Anchor – Determines if the camera will lock to the anchor.

Action Alias – When set, the input alias that determines if the camera will lock to the character (anchor).

V. Distance – Vertical distance the camera should stay from the character (anchor).

Use View Direction – Determines if the camera uses the camera's angle to focus on the anchor or if the camera's position matches the anchor's position.

Allow Disconnecting – Determines if following the anchor is disabled when another form of panning is used.

Grip Panning

Grip panning is what happens when you click the terrain and drag it around. In reality, the camera is moving even though it seems like the terrain is being pulled.

Action Alias – Input alias that is used to determine if the panning should occur. Typically this is tied to the left mouse button.

Speed – Units per second the camera moves when grip panning.

Edge Panning

Edge panning occurs when the mouse cursor moves to the edge of the screen. When it's close to the edge, the camera move in that direction.

Border – The number of pixels from the edge that the mouse has to be for panning to start.

Speed – Units per second the camera moves when edge panning.

Input Panning

This occurs when specific keys are pressed. Typically, this would be WASD or the arrow keys.

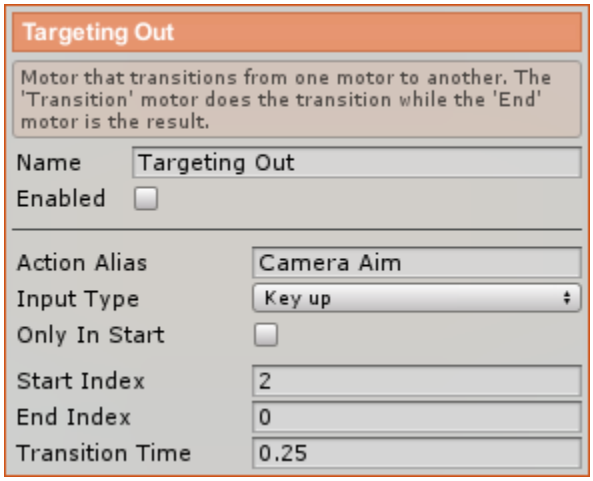
Forward, Back, Left, Right Alias – Input alias that determines panning in the direction should happen.

Speed – Units per second the camera moves when input panning.

Transition Motor

The transition motor is used to move the camera smoothly from one motor to another. This is great for having a cut-scene that ends with the player controlling the character or changing the camera style mid-game.

This motor is the only motor that will self-activate given the first couple of properties.



Action Alias – Input alias that triggers an activation.

Input Type – Determines if the activation occurs on key-down or key-up.

Only In Start – Determines if the motor test for activation only occurs if the "Start" motor is the active motor.

Start Index – Index of the "Start" motor for the transition.

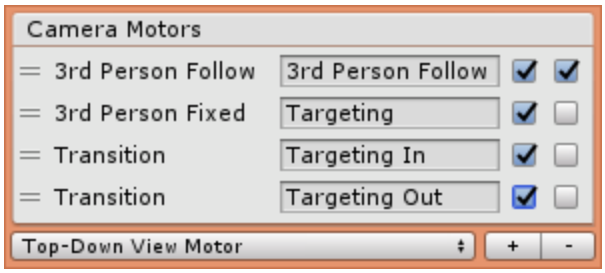
End Index – Index of the "End" motor for the transition.

Transition Time – Number of seconds the transition should take.

Like all other motors, you could have several transition motors setup at one time. This allows you have lots of different camera setups and transition to them at different times.

Targeting View (held)

Let's say you want to have a 'targeting' view activated when the right-mouse-button is held. You would have two transitions: one to enter targeting mode and one to exit. Your setup would look something like this:



The first motor is your standard camera motor when you're just moving the character.

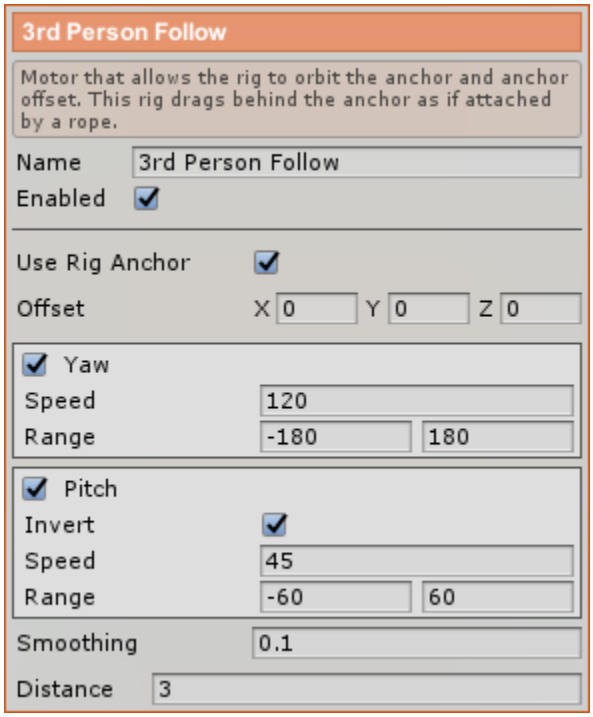
The second motor is what will be used for the targeting view. We'll look at its settings in a second.

The third motors is a transition that gets us from the standard camera motor (the first one) to the targeting view (the second one).

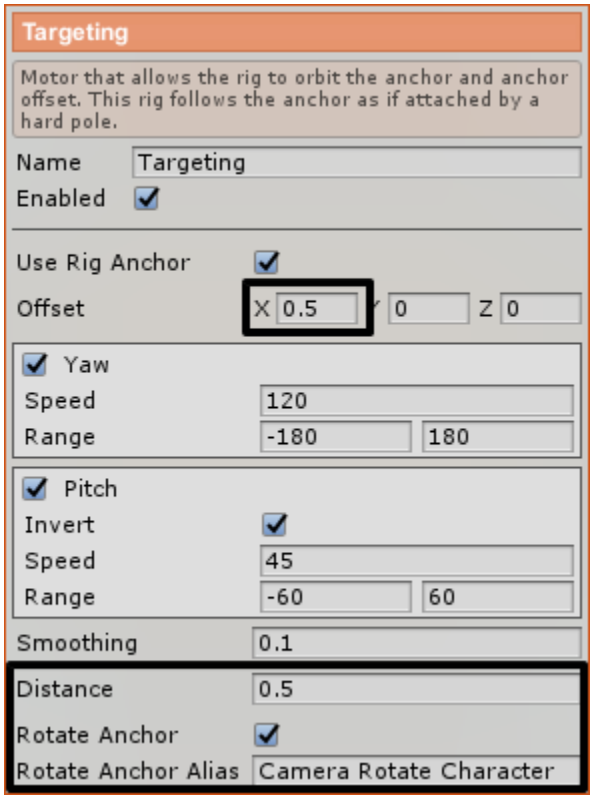
CAMERA CONTROLLER

10/11/2016

The last motor is a transition that gets us from the targeting view (the second one) back to the standard camera motor (the first one).



This first motor is just standard setup. Nothing really special about it.

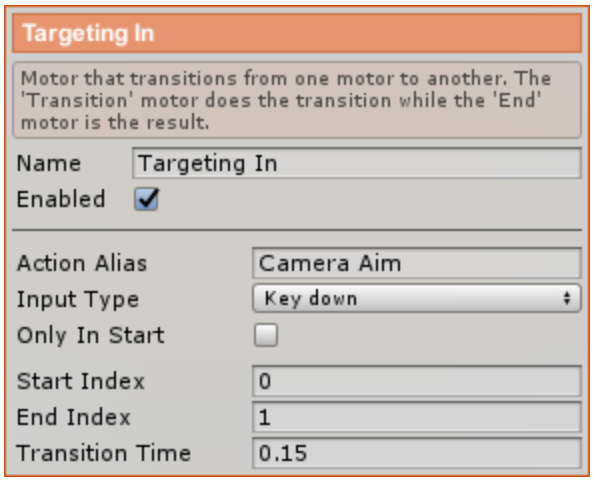


This second motor has a couple of interesting changes.

Offset – Notice that we have the camera off the character's shoulder a bit.

Distance – The camera is also much closer to the character.

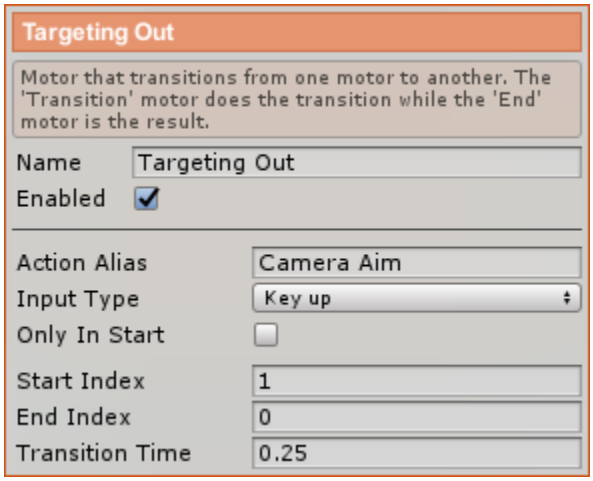
Rotate Anchor - Finally, we allow the camera's rotate the character. In this example, 'Camera Rotate Character' is tied to the right-mouse-button.



The transition to the targeting view (second motor) is pretty simple. 'Camera Aim' is tied to the right-mouse-button and this motor activates when that button is just pressed down.

I blends from the first motor (index = 0) to the second motor (index = 1) over the span of 0.15 seconds.

After the 0.15 seconds, the second motor becomes the active motor.



The second transition motor is similar and brings us back to our normal motor.

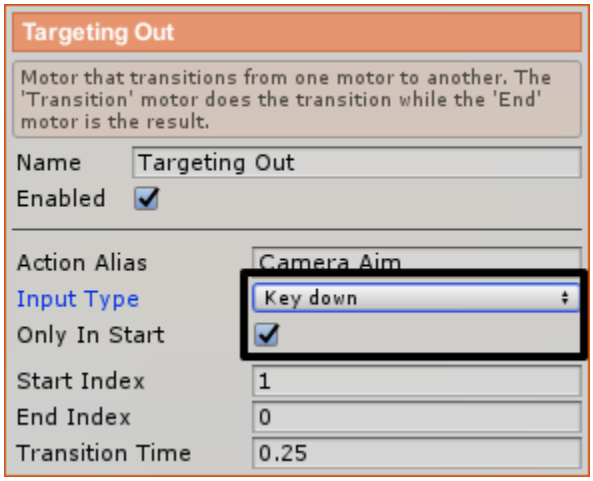
Notice the Input Type has changed to 'Key up'. This means when the right-mouse-button is just released, the motor will activate.

It then takes us from the second motor (index = 1) to the first motor (index = 0) over the span of 0.25 seconds.

After the 0.25 seconds, the first motor becomes the active motor again.

Targeting View (toggle)

Let's say we want the targeting view to be a toggle instead of just when the right-mouse-button is held. We only need to change the last transition. It becomes this:



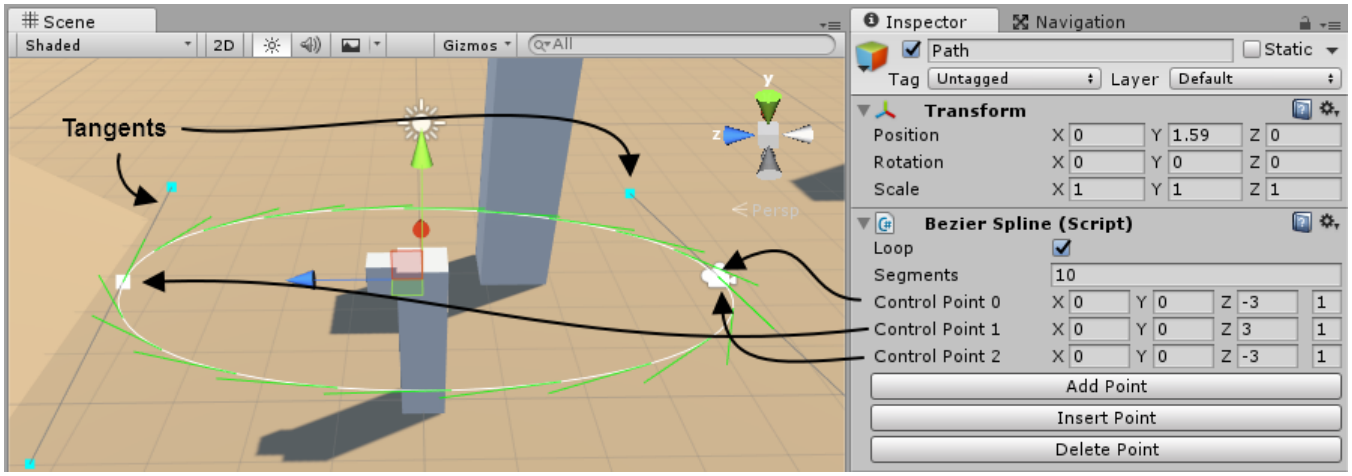
Notice we changed the Input Type to 'Key down'. However, we only do this test when we're in the second motor (index = 1).

So, we'll come out of the targeting view when the right-mouse-button is just pressed again.

Using this approach, we can create all sorts of different camera actions.

Splines

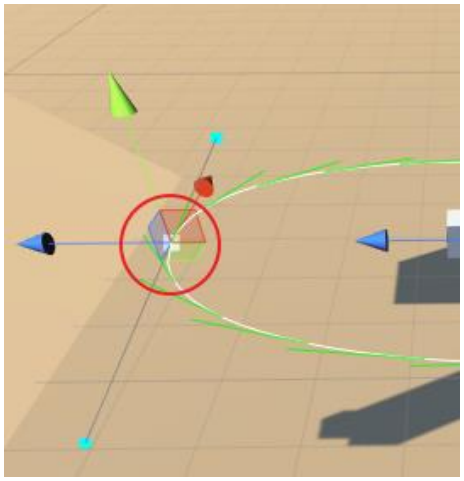
Splines are simply paths that are created using points and tangents.



By adding a **Bezier Spline** component to a GameObject, you can create a path.

In the picture above, there are 3 points (even if you only see 2). That's because the spline is set to 'loop'. So, the first point and the last point share the same position.

You can **click on a point** to select it and move it as needed.



With a point selected, you can also press the 'Insert Point' to insert a point before the selected one.

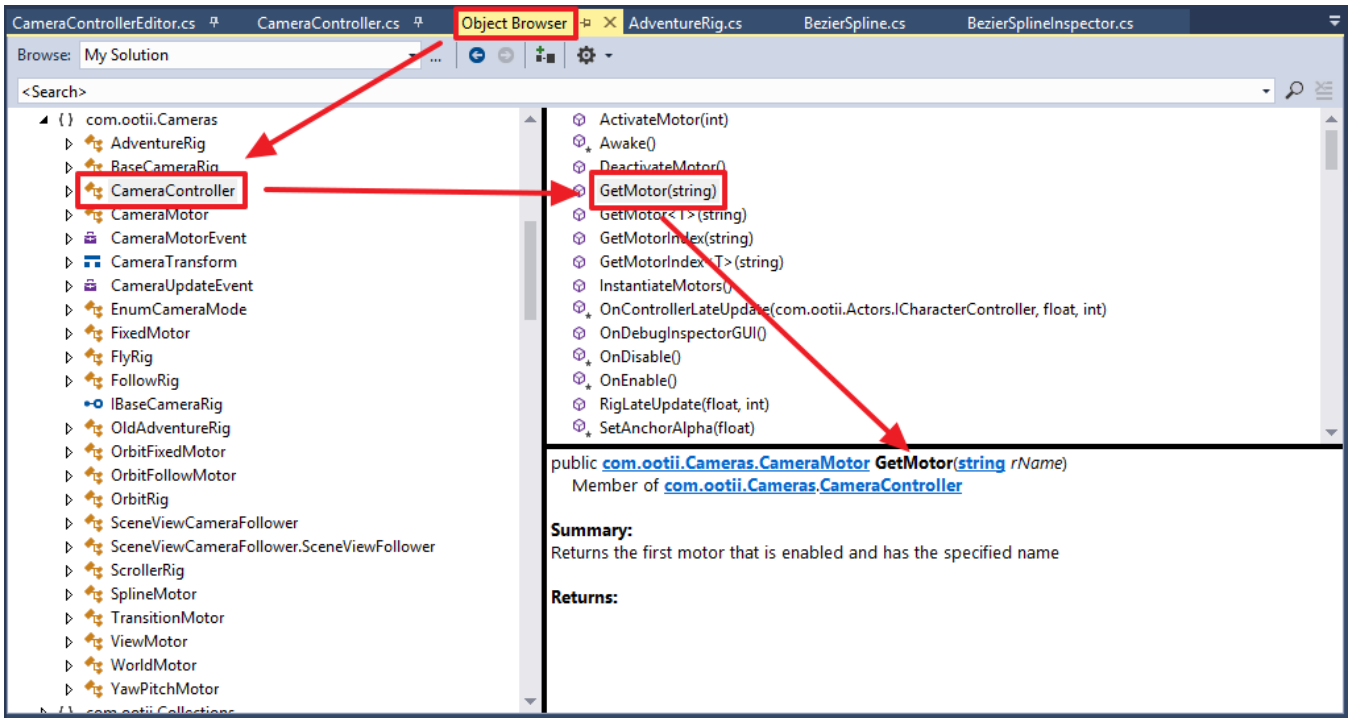
Clicking the tangents and moving them allows you to change the shape of the path.

Code

As with all motors, you can access properties through code in order to change how the Camera Controller works on the fly.

Object Browser

Pretty much everything you can do to the Camera Controller through the editor, you can do through code. The best resource is to actually your IDE's object browser. That will allow you to see all the functions that are available on the CC as well as comments I've made.



Code

Here are some common things you may want to do through code in order to control how your Camera Controller behaves.

Get the Camera Controller

```
CameraController lController = gameObject.GetComponent<CameraController>();
```

Get a camera motor by name

```
CameraController lController = gameObject.GetComponent<CameraController>();
CameraMotor lMotor = lController.GetMotor("Targeting");
```

Get a camera motor by type

```
CameraController lController = gameObject.GetComponent<CameraController>();
```

```
TransitionMotor lTransMotor = lController.GetMotor<TransitionMotor>();
TransitionMotor lTransMotor2 = lController.GetMotor<TransitionMotor>("Targeting Out");
```

Enable (or disable) a camera motor

Note that enabling a motor does not activate it. It just says that the motor CAN be activated.

```
CameraController lController = gameObject.GetComponent<CameraController>();
TransitionMotor lTransMotor = lController.EnableMotor<TransitionMotor>(true, "Targeting Out");
```

Activate a camera motor

```
CameraController lController = gameObject.GetComponent<CameraController>();
TransitionMotor lTransMotor = lController.GetMotor<TransitionMotor>();
lController.ActivateMotor(lTransMotor);
```

Set a motor property

```
CameraController lController = gameObject.GetComponent<CameraController>();
TransitionMotor lTransMotor = lController.GetMotor<TransitionMotor>();
lTransMotor.TransitionTime = 1f;
lController.ActivateMotor(lTransMotor);
```

Change the yaw/pitch speed of the motors

```
CameraController lCameraRig = gameObject.GetComponent<CameraController>();
YawPitchMotor lYawPitchMotor = lCameraRig.ActiveMotor as YawPitchMotor;
if (lYawPitchMotor != null)
{
    lYawPitchMotor.YawSpeed = 2f;
    lYawPitchMotor.PitchSpeed = lYawPitchMotor.YawSpeed * 0.5f;
}
```

CAMERA CONTROLLER

10/11/2016

Frequently Asked Questions

Below is a list of how-to and responses to questions that I get (or expect to get). Hopefully they help provide some quick insight and tutorials.

Pre-Purchase

Can I use this with Rewired or other non-ootii assets?

Yes. I've built the Camera Controller to be very modular. You can simply use a different "input source" to read from any input solution you want. See this YouTube video or the MC documentation for more info.

1st Person View

Why is my character rotating crazy with the Motion Controller?

This camera is built to rotate as the character rotates. If you're not using the Walk Run Strafe motion, the MC is causing pivots and other rotations. This camera motor than tries to compensate.

When building a 1st person game, use this motor and the MC's Walk Run Strafe motion.

When in doubt, ensure that this motor's "Rotate Anchor Alias" is blank… so we always rotate.