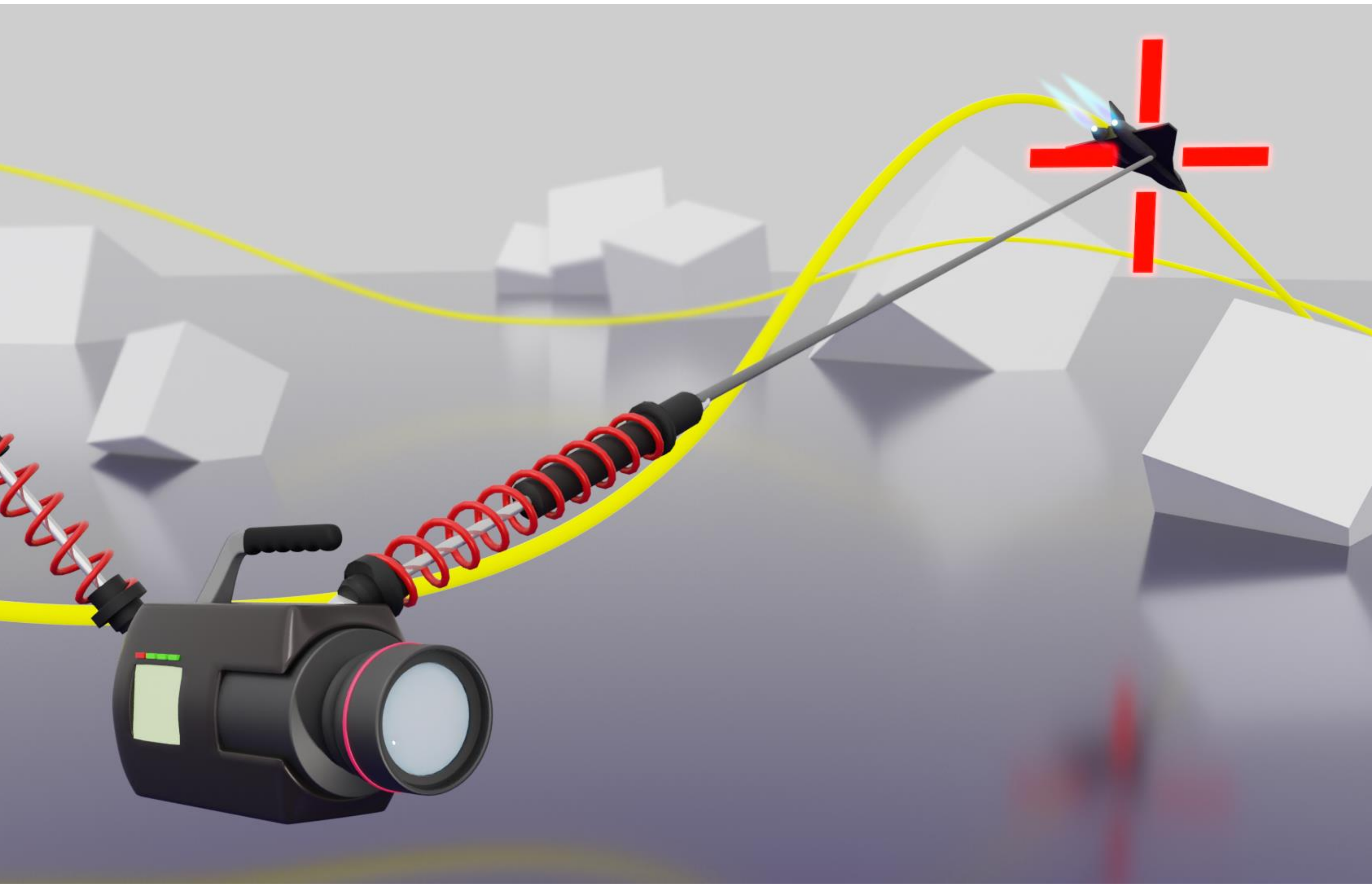
Dynamic Camera Project



Documentation

Version: 1.0

By: Eveld-Tech

1 Index

2 Introduction

The Dynamic Camera Project is a physics-based solution for camera tracking in Unity 3D. The Dynamic Camera Project is specialized for 2D and 2.5D games. This package was created to have a good working camera tracker that can track dynamic objects and is frame independent. This means that the behavior of tracking is the same for 10 fps and 1000fps. Additionally it also makes camera effects easy, such as offsetting, attracting/repelling, camera on rail effects and zooming by using “Dynamic Camera Effectors”. Which are created very easily with the tool that can be used in the Unity Editor.

This documentation is aimed for users with minimal scripting knowledge.

Quick reference:

- Setup the tracker, see 3.5.
- Setup effectors, see 4.5.
- Effector editor tool tutorial, see 5.3.
- Custom camera controller setup, see 6.1.
- Screen shaker, see 8.2.
- Demo information, see 9.

The demo scenes in the package are used with the built-in renderer. If the package is opened in a project with the Universal RP, make sure to upgrade the materials in the folder to Universal RP materials.

3 The Dynamic Camera Tracker

This chapter covers the camera tracker and how to use it in your project. The main scripts considered are: `DynamicCameraTracker.cs, DynamicCameraController.cs and DynamicCameraFunctions.cs.`

3.1 Inspiration

The camera tracker is inspired by a critically damped spring damper system that tracks a constant acceleration function (see Figure 1 and Figure 2). The advantage of this model is that there exists an analytical model of this system as it is the solution to a second order differential equation. However it is not a one to one translation, because there are some important differences. Frame lag must be taken into account when information is extracted from the internal physics engine. Details are in the source code.

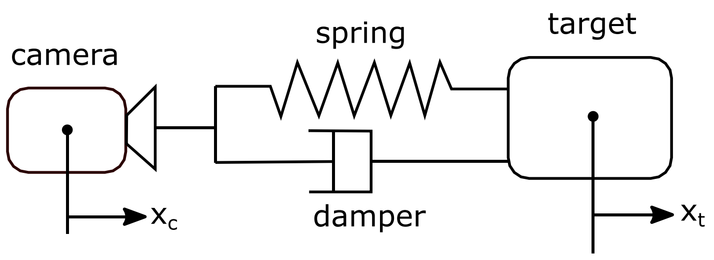


Figure 1: This is the model used for the camera tracker in 1D. The camera is connected with a spring and a damper to the target. The camera has a mass of 1kg and the target is a kinematic body. This means that the forces it experiences from the spring and damper have no effect on itself. The camera is the dynamic object, it gets pulled towards the target, because of the forces it experiences from spring and damper. The forces are zero once x_c lines up with x_t.

3.2 Types of tracking

The tracker (`DynamicCameraTracker.cs)` keeps an internal state of its physical quantities: position and velocity (mass = 1). The user can update the tracker its time step by giving a target position to track as input along with the target its velocity and acceleration if available. This guides the tracker towards the target position smoothly as you can see from Figure 2.

To achieve the best accuracy in tracking it is desired to have as much information about motion of the target. The states position and velocity are easy to get, however the acceleration has to be calculated on the fly (see 3.6). The tracker responds as follows on additional information of the target:

- **Position:** This tracks the target, but it cannot track the velocity properly. Therefore the position of tracker will lag the target by an magnitude of $d = \text{smoothTime} \times v_{target}$. An advantage of this is that the tracker will not overshoot the target when it suddenly stops. If the tracker lags the target too much it can be distance limited by setting an appropriate max distance . This is a nice general approach for target tracking.
- **Position + Velocity:** This tracks the target a lot better. It can match the target exactly when moving at constant speed. However there is still a bit of lag when the target accelerates or decelerates. The lack of information about acceleration causes the tracker to overshoot the

target if it comes to a sudden stop. However this can be resolved by enabling the anti overshoot option.

- **Position + Velocity + Acceleration:** This tracks the target as good as it gets. It can match the target's position precisely even while under constant acceleration (like gravity). It responds to quick changes in velocity as well and quickly converges back to the target. It still can overshoot when the target suddenly stops, but significantly less then with the velocity. Note that the acceleration component is equal to the force as this is a system with a mass of 1 kg. This also mean that you can use the acceleration component to nudge the camera in a direction when the player gets hit. This type of tracking is a bit harder to tweak to feel nice to a player. It also depends on what kind of tracking is desired.

The user can determine for themselves what kind of tracking information is used. If you only want to have positional tracking then you have to set or multiply the velocity and acceleration by zero. It is possible to switch dynamically between the tracking modes, although that requires some scripting from the user.

A remark on overshooting. The severity of overshoot can be reduced by changing how fast a target comes to a stop. A gradual stop is easier to track then a sudden stop. A sudden stop from high speeds will lead to an overshoot no matter what you do. Although the tracker overshoots, it still looks smooth to a player! If you think about it, overshooting is a logical consequence (physically speaking) of tracking a target that goes fast and comes to stop. Imagine you run up to someone and maintain the same speed. Your goal is to match his position by running next to him all the time. Now it so happens that the person you are tracking collides with a lamp post and comes to a sudden stop. You can't match that position that quickly and you overshoot your target! Luckily there is also an option to prevent overshooting per axis (see 3.3)!

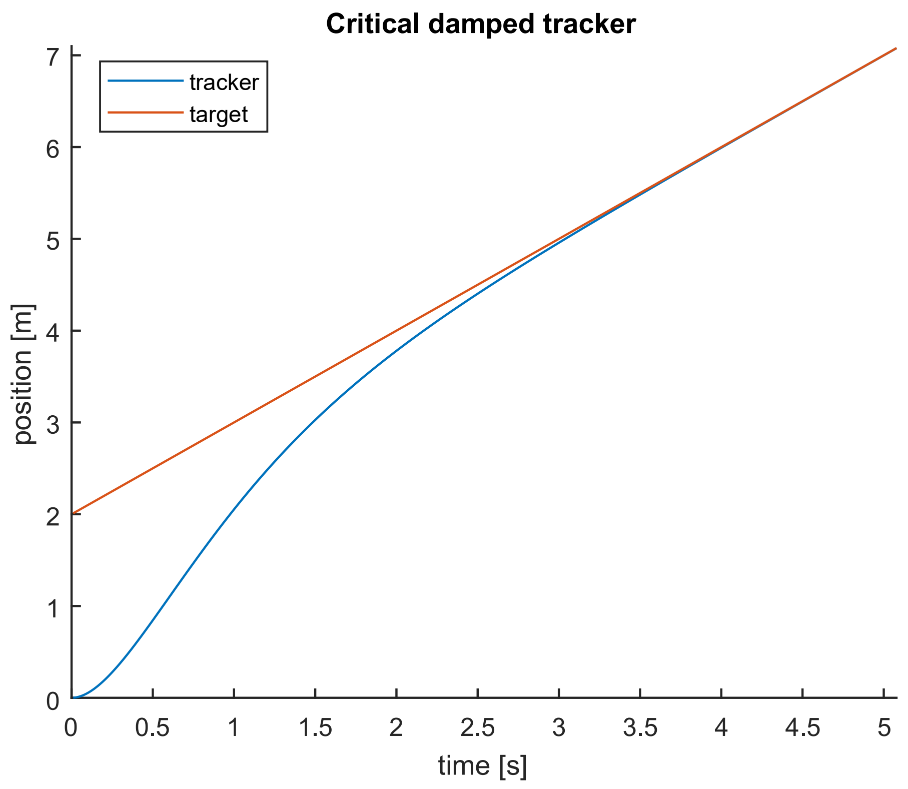
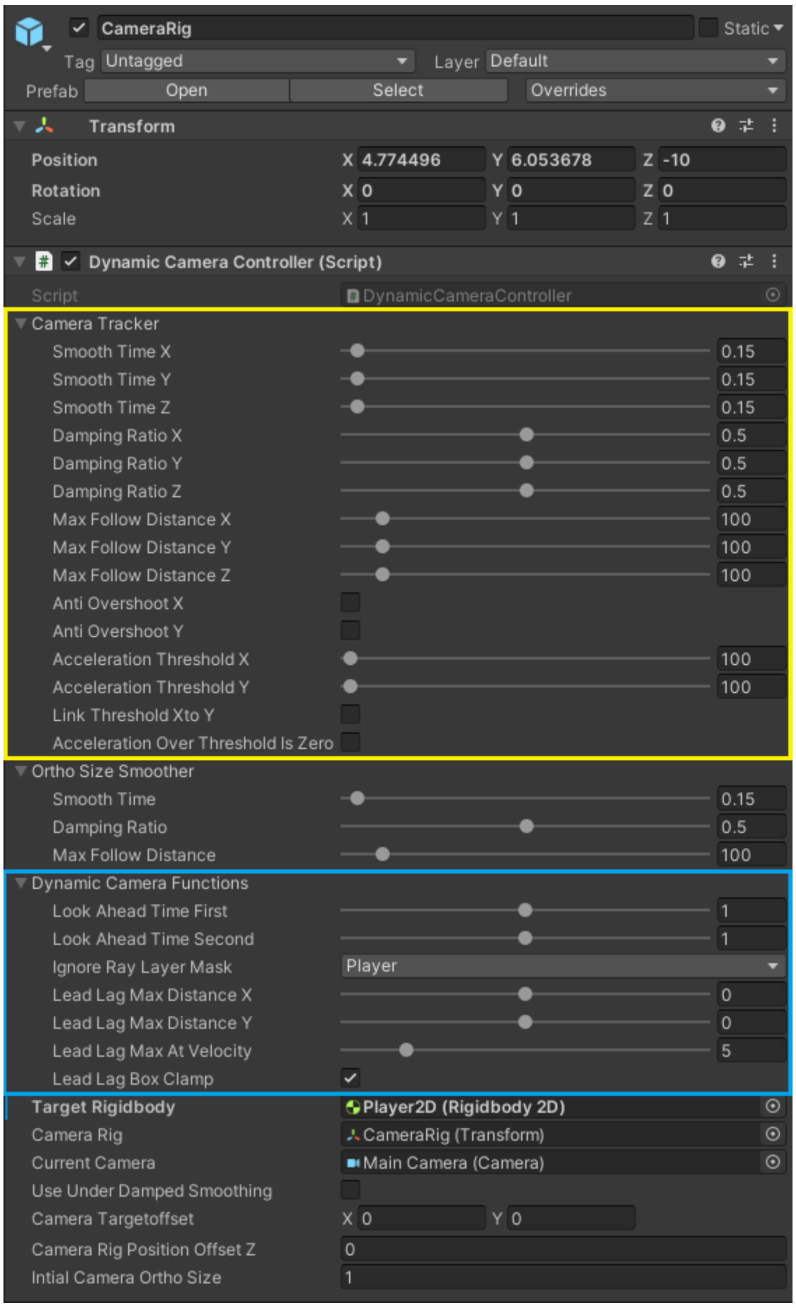


Figure 2: Plot of a 1D second order system that tracks a constant velocity. As you can see the blue line (the tracker) catches up with the red line which is the target.

3.3 Basic functionality

This section will be explained by using the DynamicCameraController.cs script in the inspector window which is attached to the prefab: CameraRig.prefab in the DemoAssets folder as can be seen in the figure below.



CameraRig ✓ Static
Tag Untagged Layer Default
Prefab Open Select Overrides

Transform

Position	X 4.774496	Y 6.053678	Z -10
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Dynamic Camera Controller (Script) ✓

Script	DynamicCameraController
Camera Tracker	
Smooth Time X	0.15
Smooth Time Y	0.15
Smooth Time Z	0.15
Damping Ratio X	0.5
Damping Ratio Y	0.5
Damping Ratio Z	0.5
Max Follow Distance X	100
Max Follow Distance Y	100
Max Follow Distance Z	100
Anti Overshoot X	☐
Anti Overshoot Y	☐
Acceleration Threshold X	100
Acceleration Threshold Y	100
Link Threshold Xto Y	☐
Acceleration Over Threshold Is Zero	☐
Ortho Size Smoother	
Smooth Time	0.15
Damping Ratio	0.5
Max Follow Distance	100
Dynamic Camera Functions	
Look Ahead Time First	1
Look Ahead Time Second	1
Ignore Ray Layer Mask	Player
Lead Lag Max Distance X	0
Lead Lag Max Distance Y	0
Lead Lag Max At Velocity	5
Lead Lag Box Clamp	✓
Target Rigidbody	Player2D (Rigidbody 2D)
Camera Rig	CameraRig (Transform)
Current Camera	Main Camera (Camera)
Use Under Damped Smoothing	☐
Camera Targetoffset	X 0 Y 0
Camera Rig Position Offset Z	0
Intial Camera Ortho Size	1

Figure 3: Inspector window when the prefab is selected in the hierarchy.

The camera tracker script does not inherit MonoBehavior, therefore it is used by the DynamicCameraController.cs script. This script updates the camera tracker every frame. Note that this script is a template of how to use the camera tracker. You can create your own camera controller script very easily. The main components of the camera tracker is marked by the yellow box. The fields are:

- **Smooth time (XYZ):** Can be interpreted as the time it roughly takes to reach the target. In mathematical terminology, this is a term of the time constant of the differential equation. To be precise $\tau = \text{smoothTime}/2$. Note that the smooth time has to be greater than zero!
- **Damping ratio (XYZ):** This value determines how much the tracker oscillates around the target. Zero means no damping at all and 0.99 means almost at critical damping. The damping ratio has to be greater than zero and less than 1!
- **Max Follow Distance (XYZ):** This is the maximum follow distance the camera tracker can have with respect to the target. If the tracker exceeds this distance, the tracker will be clamped to this distance. Note that setting the max follow distance to small values can have an undesired effect when combined with effectors such as snapping instantly to new the displaced target.
- **Anti-Overshoot (XY):** This prevents the tracker from overshooting the target when the tracker exceeds the target position and while the tracker its velocity is greater than the target. The options are separated in their axis because the user might want to have only anti overshoot on the X axis and not the Y axis. The anti-overshoot options should be used carefully as it may give a snappy feeling to the tracker. It may require extra tuning of parameters to make it feel good.
- **Acceleration threshold (XY):** This is the max acceleration the tracker can respond to. This means that you can limit the tracking of acceleration after a collision for example. This results in higher tracking error at the moment after the collision, but it won't have a sudden jump towards the target. Another example is that you can tweak and limit the acceleration when the player is accelerating and do not limit it when stopping. This results in the camera gradually following the player at the start and quickly stopping when the player also stops.
- **Acceleration = 0 if > threshold:** The acceleration that the tracker will respond to is set to zero if the acceleration threshold was exceeded.

You can toy around with the settings in the demo scenes to see how the tracker behaves. **Note that you can change the range values of the inspector inside the scripts if necessary.**

3.4 Advanced functionality

The camera controller also comes with special functions that have an impact on how the camera tracker behaves. These functions are contained in the DynamicCameraFunctions.cs script and are marked by the green box. The fields are:

- **Look ahead time first:** This time is used to calculate the distance the rigid body looks forward with. It is calculated as $d_1 = vt_1$ where t is the look ahead time. This distance is used to cast a ray along the direction of the velocity to detect a collision. If there is an upcoming collision, the velocity gets compensated in such a way that the tracker goes towards the future position after the collision.
- **Look ahead time second:** This is similar to the first look ahead time. The difference is that this look ahead time checks what is going to be collided after the first collision. If there will be a second collision, the velocity won't be compensated as much. If there is no collision but there is a first collision, then this has no effect. **Important note:** the look ahead only affect the velocity and not the target position. However by shifting the velocity the tracker does offset the target position!
- **Ignore ray layer mask:** This must be set to the rigid body its layer. This is only relevant if look ahead times are used.

- **Lead Lag Max Distance XY:** The maximum distance you can lead/lag the target. A negative value will lag the target.
- **Lead lag max at velocity:** The maximum lead/lag distances is achieved when above this velocity. The lead/lag distances is interpolated between 0 and 1 when below this value .
- **Lead lag box clamp:** This clamps the lead/lag to a box shape. This means you can control the x-axis and y-axis individually. Also use this setting if you want to lock an axis. When disabled it clamps the distance as an ellipsoid/circle shape where the direction of velocity determines the radius value.

These functions add extra possibilities to the camera tracker. However these are not needed to have the tracker functional. Note that it is required to have DynamicCameraFunctions.cs if you want to make use of the effector displacements with clamped velocity and accelerations!

The lead/lag is tricky to use in effectors as they extrapolate the tracker its position. This means that it always has an offset in rail effector for example. If necessary, you can turn lead/lag off inside effectors. However for small lead/lag positions the discrepancy is negligible. In the DynamicCameraController.cs you can read more info about this, as well as two ways to have lead/lag.

3.5 Quick setup

Drag the CameraRig.prefab into the scene. Set the target rigid body to the rigid body you want to track in the inspector. Make sure to set the CameraRig.prefab at a good Z position for your game. Press play and it works out of the box. In case it does not work properly, make sure to set the Ignore Ray Layer Mask to the rigid body its layer.

3.6 Setup by scripts

As the DynamicCameraTracker.cs is serialized you can use this as public field in scripts. However for a good start position it requires initialization for the initial conditions:

```
public DynamicCameraTracker cameraTracker;
public Transform cameraRig;

void Start()
{
	// initialize the tracker position at the camera rig and velocity to zero
	cameraTracker.SetInitialConditions(cameraRig.position, Vector3.zero);
}
```

Then for Updating the tracker we need:

```
private Vector2 targetAcceleration = Vector2.zero;
private Vector2 previousTargetVelocity = Vector2.zero;
public Rigidbody2D targetRigidbody;

private void FixedUpdate()
{
       // If you want to use acceleration, calculate acceleration like this
       // (must be in fixed update for most stable results):
       targetAcceleration = (targetRigidbody.velocity - previousTargetVelocity)
                                          / Time.fixedDeltaTime;

       previousTargetVelocity = targetRigidbody.velocity;
}


void LateUpdate()
{

       if (cameraRig != null)
       {
       // calculate the future camera position based on all available parameters

              cameraRig.position =
       cameraTracker.CriticalDampedStep(targetRigidbody.transform.position,
       targetRigidbody.velocity, targetAcceleration, Time.deltaTime,
       MassSpringDamperFunctions.ClampType.Circle);

       // NOTE: use cameraTracker.CriticalDampedStableClampStep(…) if you want to
       // using clamping for the positional tracker.


       }
}
```

This is the most basic tracking script you can have. It can be even simpler if you multiply the velocity with zero and set the acceleration to zero. Then you have only positional tracking. Note that the acceleration is calculated by taking the difference of the last velocity and the current velocity divided by the fixed time step. Technically speaking the acceleration always lags up to one frame from the true value, but this is the best way to calculate it.

In this case we used the CriticalDampedStep(…) update for updating the new camera position. This function uses critical damping and has therefore no oscillations. You can set the velocity and acceleration terms to zero if you want to have only positional based tracking. This induces lag to the target tracking with a maximum distance $d = v \times \mathrm{smoothTime}$. This means it can never converge to the target position.

4 Dynamic Camera Effectors

Camera effectors are an important part of this package. They allow for easy camera effects for your 2D/2.5D projects

4.1 What are effectors and what do they do?

Effectors create a displacement for a point inside of them. The direction of displacement depends on what the user wants. For example if a circle is used as effector we can make the displacement attract towards the center as shown in Figure 4. The displacement is used for displacing the camera, however you are free to use it for displacing other objects as well. The displacement is a three dimensional vector where the z-component is used for displacement in the z-direction and zooming! The displacement is instantaneous, however in combination with the Dynamic Camera Tracker we get a very smooth transition during displacements.

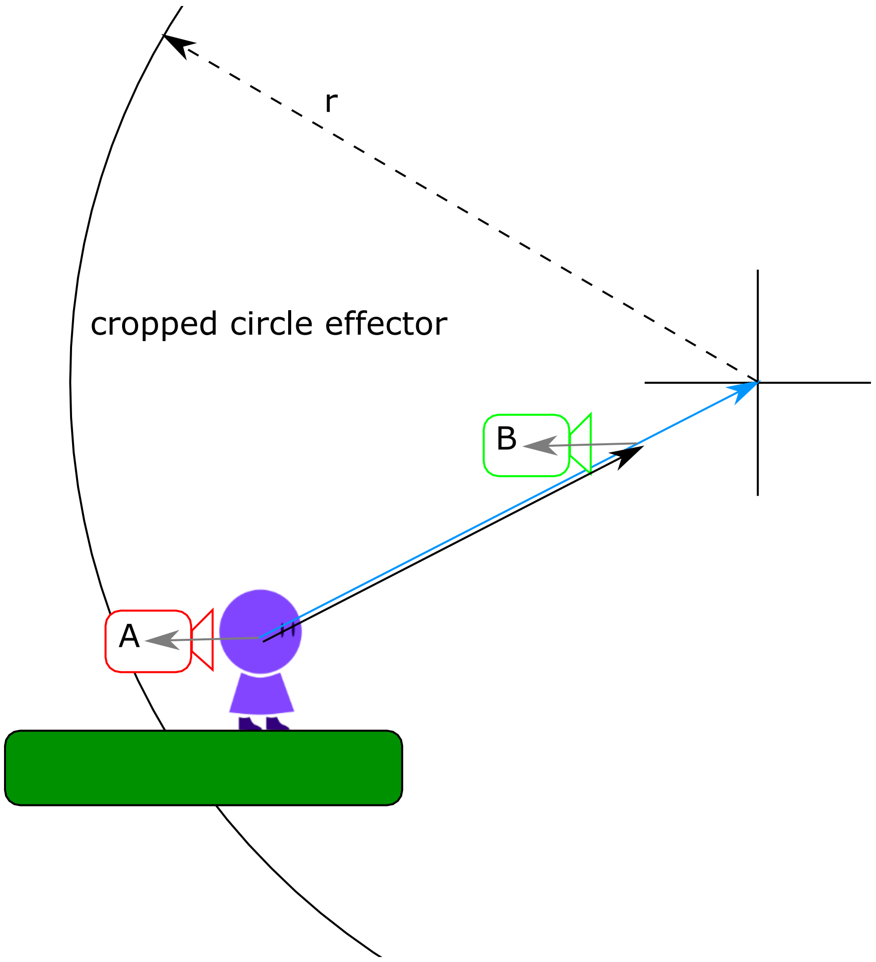


Figure 4: The purple player is inside the effector and has camera A attached to it. This effector is a circle effector with radius r and the plus sign in the center. The effector is set to attract towards the center with a certain strength. The direction of attraction is the blue line and the displacement factored by the strength is the black arrow. The displacement is then applied to the camera so the new camera position ends up in position B.

4.2 General effector options

All effector have a set of general options for displacements. These are:

- **Is Enabled:** (true/false) whether the effector is enabled.
- **Effector Name:** the name of the effector. It is best to use a unique name if you want to look up the effector.
- **Feather amount:** (0..1 factor) the feather amount determines how fast we transition from 0 influence to one influence. This allows for gradual/rapid transitions. The feather amount determines the influence (a range from 0 to 1) of the effector at a given point in the effector. The boundary of the effector has usually an influence of 0 and closer to the center it is one, the effector influence is linear interpolated between these values. The feather amount displaces the boundaries so that we can bring the effector influence to 1 of the whole effector when the feather amount is set to 1.
- **Invert feather region:** (true/false): Flips the region of the feather. If true, it makes the boundaries of the effector 1 and 0 towards the center.
- **Use Shaped Region:** (true/false) whether to use the special shaped region for the effector. This region determines where the effector is active and can have complex shapes.
- **Repel:** (true/false) repel flips the attraction.
- **Invert Strength:** Flips strength values. For example the strength of a circle effector is at its center. This flips the strength so that the boundary strength is this strength and zero at the center.
- **Distance equals strength:** the distance towards the center or center line is now the displacement. This displacement is still factored by the feather amount, but is independent of the strength. It is affected by the depth strength! This is a really useful setting to get an effector up and running quickly.
- **Unilateral displacement:** unilateral displacement creates a displacement that is always in one direction. This means that the displacement is parallel on every location inside the effector. For example, a circle effector does this by having the handle of the radius point in the desired direction with respect to the center.

All effectors have strength settings:

- **Strength:** determines the amount of displacement in the XY direction. The strength is designed to be positive and those displacements will never exceed the boundaries of the effector. However the strength is allowed to be negative and the displacements are not limited by the boundaries of the effector.
- **Depth Strength:** is used for zoom effects. For 2.5D it displaces the camera in the Z direction and for orthographic cameras it changes the orthographic size based on the depth and a scaling factor. A positive depth strength corresponds to zooming in and a negative value corresponds to zooming out.

Some effectors have specialized settings, such as the multi effector where you can select to loop it around or use ending caps.

4.3 Types of effectors

There are six types of effectors, each with a characteristic displacement. The effector types are:

- **Circle Effector:** a circle shaped effector see Figure 5. The main point of attraction is towards the center.
- **Semi Circle Effector:** A slice of a circle see Figure 5. The main point of attraction is towards the center.

- **Torus Effector:** a 2D torus type of effector see Figure 6. The main point of attraction is towards the major radius line.
- **Semi Torus Effector:** a slice of a torus see Figure 6. The main point of attraction is towards the major radius line.
- **Box Effector:** technically a symmetric trapezoid shaped effector see Figure 7. The main point of attraction is towards the center. This has two nodes with strength and the strengths are linear interpolated in between them. Red is node 1 and the light blue node is node 2.
- **Multi Effector:** a multi connected type effector see Figure 7. This effector connects the box effectors and specialized semi torus effectors together. The main point of attraction is towards the center line of the connected points.

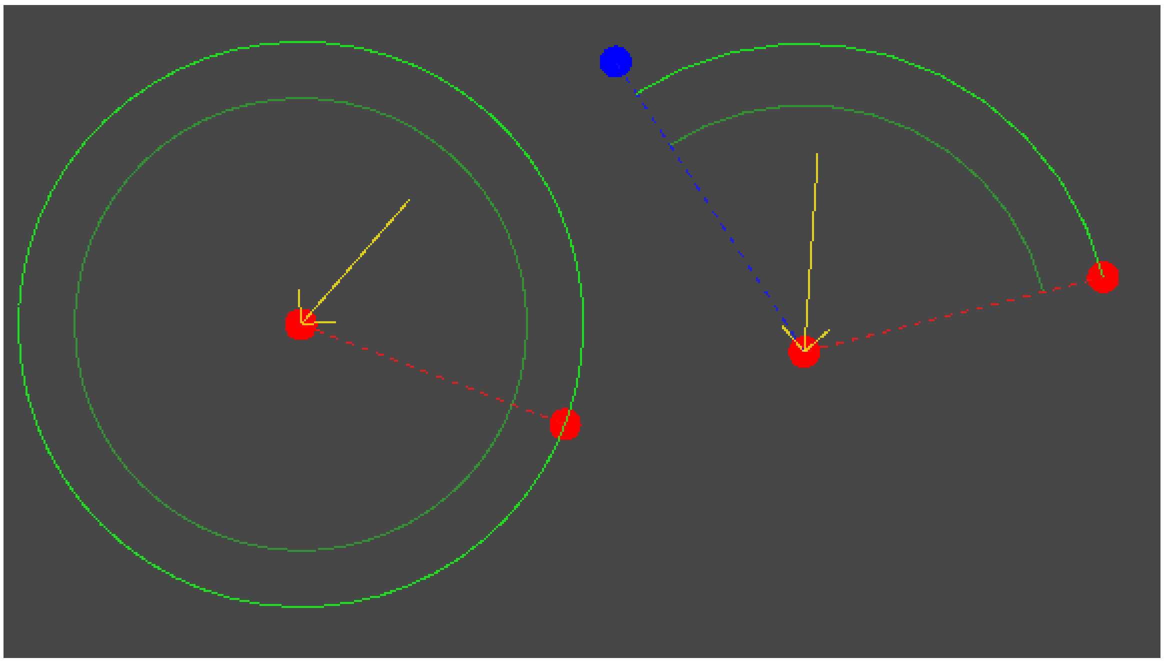


Figure 5: Circle effector and Semi Circle Effector. The green line is the boundary and the slightly darker green line is the feather amount (set to 0.8 here). The yellow arrow shows the direction of displacement with default settings.

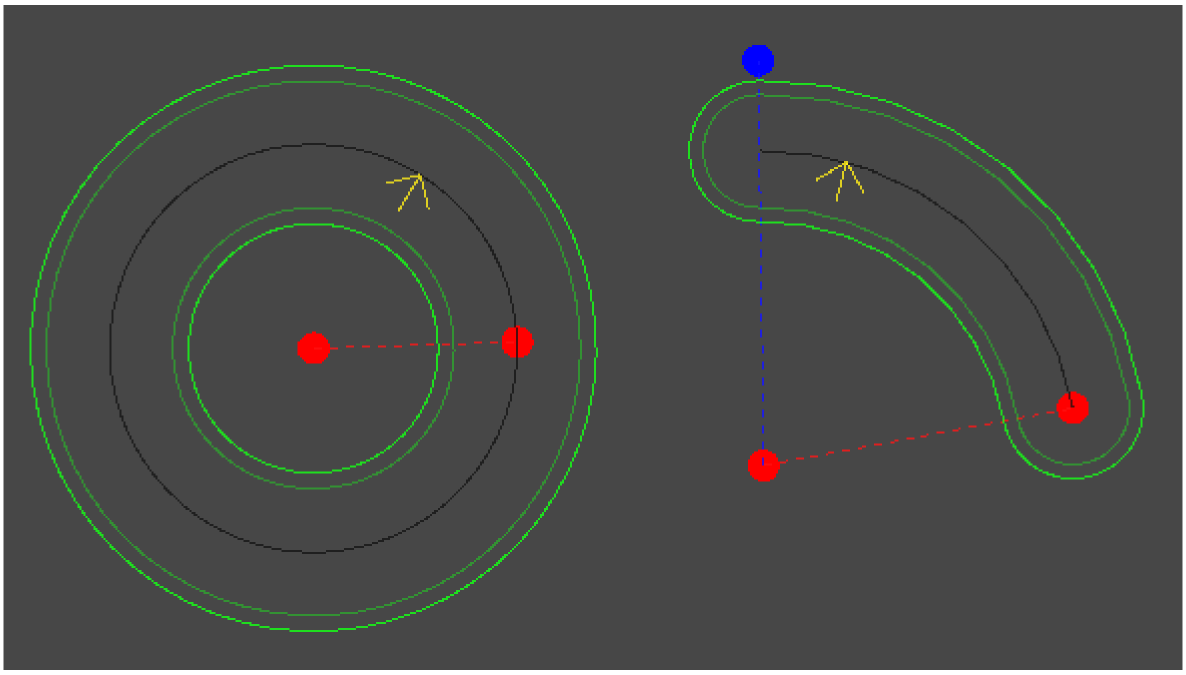


Figure 6: Torus Effector and Semi Torus Effector. In this case the semi torus effector is drawn with circle caps. The green line is the boundary and the slightly darker green line is the feather amount (set to 0.8 here). The yellow arrow shows the direction of displacement with default settings. The black line is the major radius of the torus.

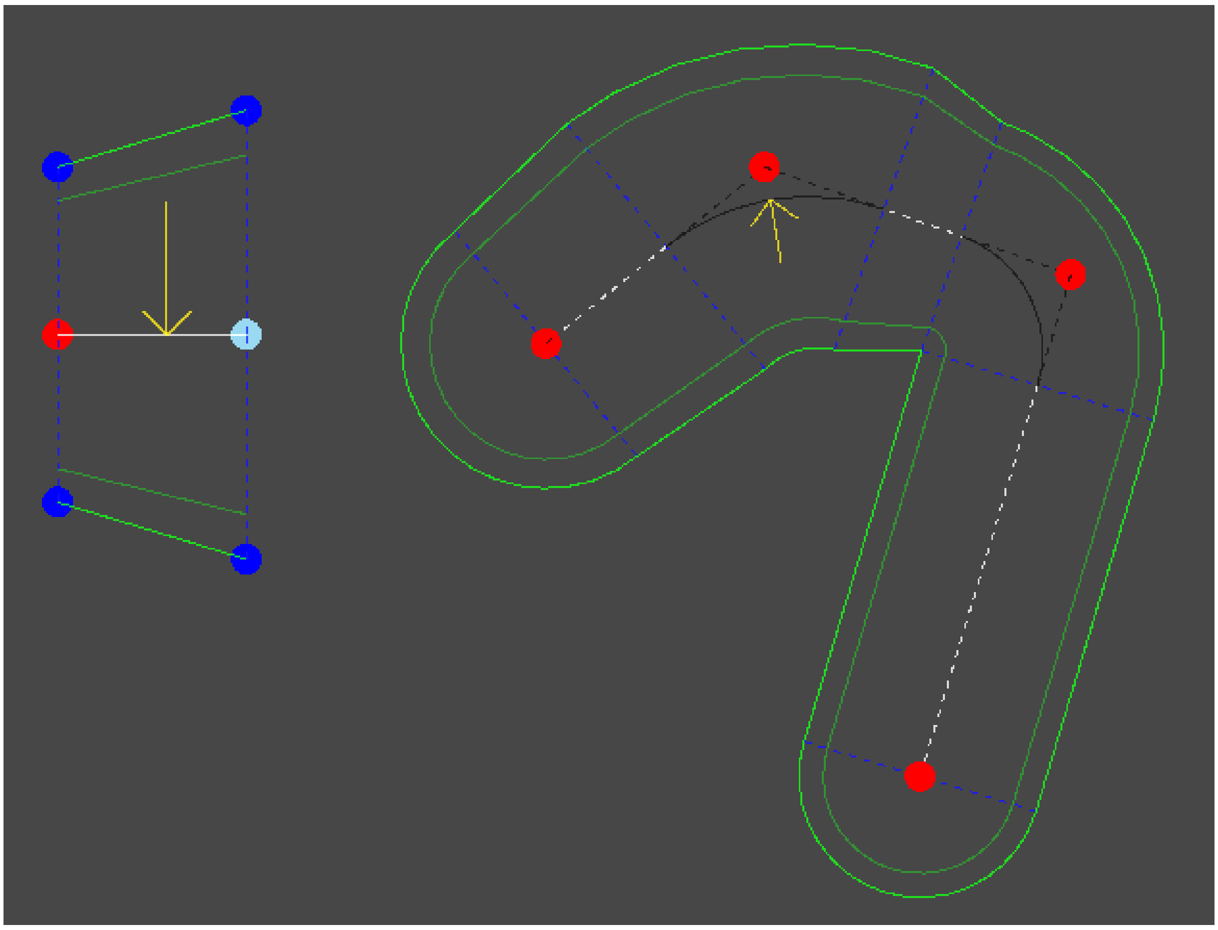


Figure 7: Box and Multi Effector with ending caps. The green line is the boundary and the slightly darker green line is the feather amount (set to 0.8 here). The yellow arrow shows the direction of displacement with default settings. As said before the box effector is not a true box but a symmetric trapezoid. The Multi effector is a linkage of box effectors and a specialized semi torus effector with variable minor radius.

4.3.1 Shaped region of the effector

All effector regions of activity can be customized by using shaped regions. The effector only displaces points when they are **inside of the effector and the shaped region**. These regions are defined by the user (see 5.3.3). The shaped region can have any shape. An example of a shaped region is given in Figure 8.

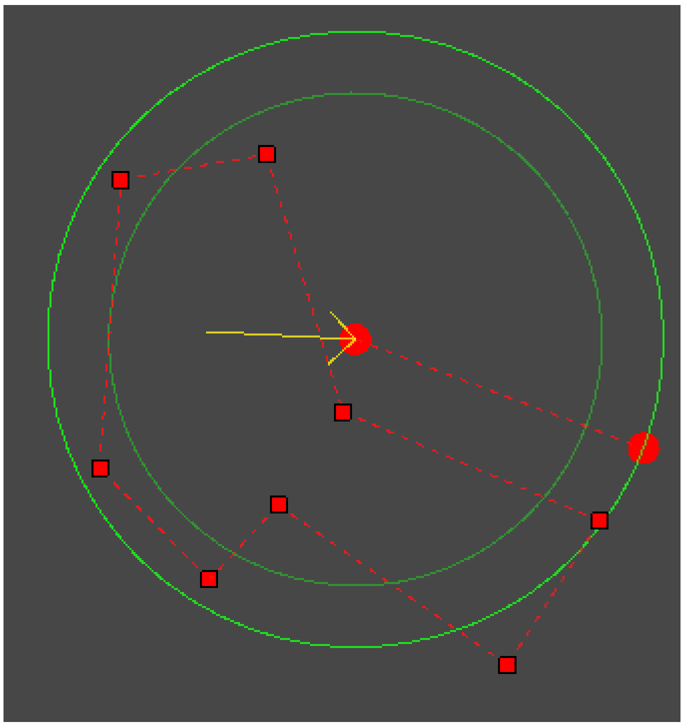


Figure 8: A circle effector with its shaped region marked by the red squares and dashed lines . The effector is only active inside the shaped region and the effector itself! The yellow arrow is the displacement, not that the displacement can exceed the shaped region boundary.

4.4 Examples of effector usage

This section covers some inspiration on how to use the effectors for cool effects. All effectors are created with the effector tool, so you can recreate it easily.

4.4.1 Focus camera on a fixed area

You can use effectors to focus on a fixed area, see Figure 9. (Semi) circle effectors can fix the camera in the XY direction so you can have a focus on something as soon as you enter the effector. Other effectors can not do this as they only lock one degree of freedom.

To create this type of effector (similar as in the demo) you:

- Create a circle effector or semi circle effector.
- Set the **feather distance** to a high value (0.8 used in the screenshot)
- Set **dist = strength** to true.
- Tweak the **depth strength** to have a zoom effect.

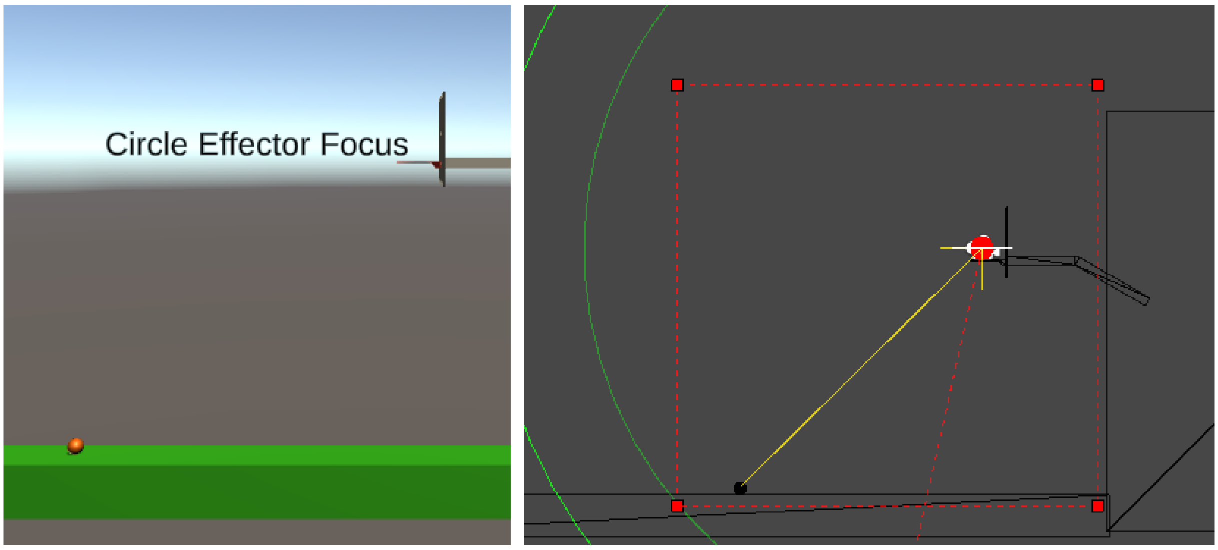


Figure 9: The orange ball is the player inside the circle effector. The circle effector displaces the camera to its center and it get fixed there.

4.4.2 Prevent the camera from entering room transitions

Effectors can be used to prevent the camera from entering room transitions. A box effector is used for this effect in Figure 10. This is also a very simple effect to create.

To create this type of effector (similar as in the demo) you:

- Create a **box effector**
- Set the **feather distance** to a high value (1 in this case)
- Set **dist = strength** to true.
- Set **repel** to true.
- Tweak the **depth strength** to have a zoom effect.

Or instead of repelling you can set repel to false to focus on the center of the room transition until the player is out of it.

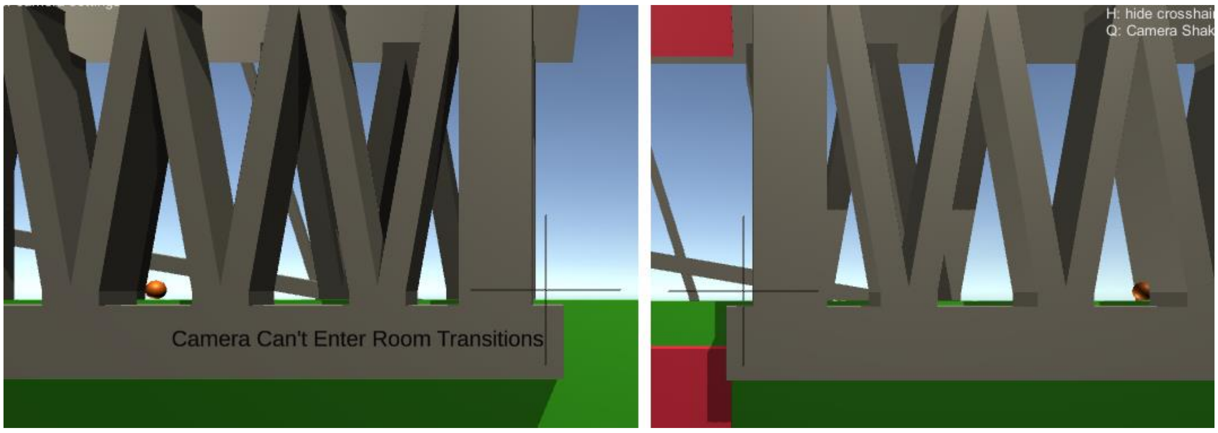


Figure 10: The orange ball is the player where the camera is normally focused on. The crosshair marks the location of the camera center. The ball travels from right to left. As you can see, the camera stays out of the room during the transition from right to left and the other way around. The transition of the camera locations happens smoothly and does not jump instantly based on the smooth time of the camera.

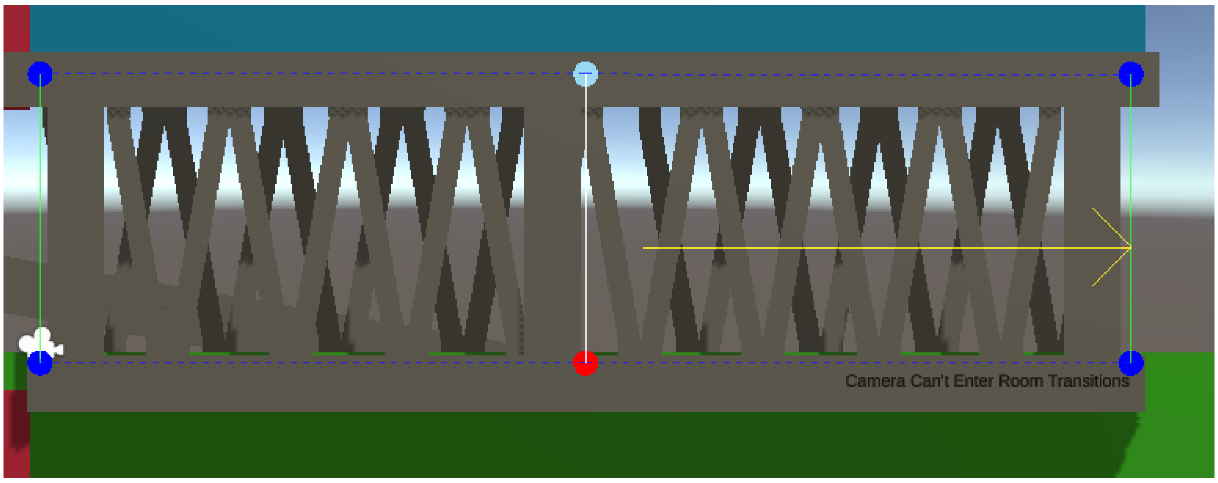


Figure 11: The box effector placed over the room transition to keep the camera out of it. The yellow line is the displacement vector and feather amount is set to 1.0. You can see the camera on the left corner (the screen shot from Figure 10 is at this point). The displacement flips direction once the middle line is crossed. You can also have a focused transition by setting repel to true.

4.4.3 Rail guides for the camera

A rail effector is made by using the multi effector:

- Create a multi effector.
- Add points along the path you want to have function as rail.
- Shape the points to whatever you like.
- Set feather amount to a high value
- Set **dist = strength** to true.

This effector pushes the camera towards the center line of the multi effector resulting in a rail effect when moving parallel to it.

4.4.4 Zoom effect only

To create a zoom effect effector you have to:

- Create any effector
- Set Depth Strength to any value you like. Negative values for zooming out and positive for zooming in.

This is the simplest way of creating a smooth zoom effect for a 2.5D camera. However for smooth zooming with an orthographic camera, it is required to have some additional code (see 6.2).

4.4.5 Offset effect

To create an offset effect you have to:

- Create any effector.
- Set **Unilateral Displacement** to true.
- Set the desired **strength** values.
- Shape the effector in such a way you get the desired direction.
- Optional: Limit the area of the effector by adding a shaped boundary (see 5.3.3).

4.5 Quick Setup

See the editor tool (5) for adding effectors. Once an effector has been added it can be used in scripts by calling `DCEffectorManager.GetDisplacementAt(…)`. If you want to use the camera tracker you can use the `dynamicCameraFunctions.DisplaceByEffectors(…)` method. This is a better option than the previous method, because this method clamps the tracker velocity when necessary. This results in a stable camera in effectors during clamping or rail effects.

4.6 Setup with scripts

Although it is possible to create and setup effectors by using scripts, it is not recommended. There are a few important functions for using effectors with scripts.

4.6.1 General usage

To get the displacement of an effector use the method

```
DCEffectorManager.GetDisplacementAt(position, out DCEffectorOutputData, bool, bool))
```

This is the most basic method for getting displacement. This displacement does not account for velocity locking or clamping. To take this into account you have to use:

```
public DynamicCameraFunctions dynamicCameraFunctions;

void LateUpdate()
{
      targetPos = dynamicCameraFunctions.DisplaceByEffectors(targetPos, ref targetVel,

                  ref targetAccel, out displacementOutput);

}
```

This displaces the target position and adjust the target velocity and target acceleration for the tracker so that I can hook onto rails or lock itself on effectors.

4.6.2 Advanced effector usage

You can move effectors around in scripts by using:

```
// Example how to move effectors
public string effectorName;
public DCEffector effector;
private Vector3 effectorOffset;

void Start()
{
       // look up effector and store it
       effector = DCEffectorManager.GetEffectorByName(effectorName);
       if (effector != null)
       {
              // offset between moving object and effector
              effectorOffset = (Vector3)effector.rootPosition - transform.position;
       }

    }

private void Update()
{
       if (effector != null)
       {
            // move the effector wrt the moving object
            effector.MoveEffectorTo(this.transform.position + effectorOffset);
       }
}
```

This has the effect of parenting it to the object without inheriting that object its rotation.

For completeness, if you want to create an effector programmatically during runtime, you must add it to the global list in the DCEffectorManager.cs by using:

```
// To add an effector to the effector manager use
DCEffectorManager.AddEffectorToGlobalList(yourEffector);

// You also have to make sure you remove the effector when you don’t need it anymore,
// because effectors are persistent during scene swaps if not removed.
DCEffectorManager.RemoveEffectorFromGlobalList(yourEffector);
```

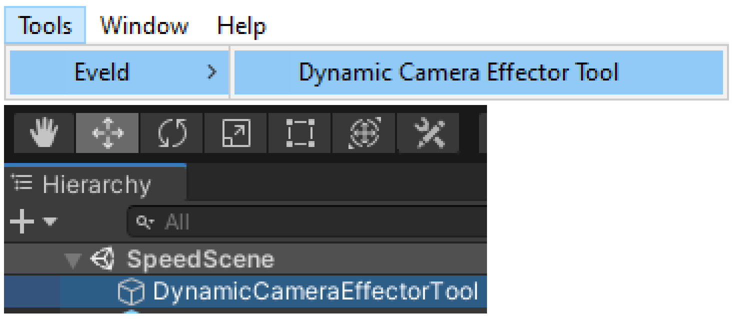
You have to use this method, else it won’t be in the look up list when you want the displacement of the effectors.

5 Editor Tool for creating Effectors

This section covers a short tutorial on how to create effectors in the editor with the special tool. The tool allows the user to create effectors fast and easy with an intuitive editor window with full undo/redo support.

5.1 Adding it to the Unity hierarchy

In the Unity editor tool bar go to: Tools>Eveld>Dynamic Camera Effector Tool. This adds a DynamicCameraEffectorTool game object in the hierarchy. Note that you can have multiple tools in one scene.



Alternatively you can add the tool without going through the menu. In that case, you have to create a game object and add the DCEffectorManager.cs as component to it.

The DynamicCameraEffectorTool game object has DCEffectorManager.cs as component. This acts as the main container of effectors during run time. It stores the effectors locally in the editor environment.

5.2 Editor controls

The editor uses mouse and keyboard controls. Keyboard controls are used for general operations and with the mouse you can customize the effectors.

5.2.1 General controls

The tool functions become active as soon as you focus on the scene window.

The keyboard controls are:

- **Key A:** deselect by one stage, this means that if you have selected a handle it will deselect this first. Then on the second press it will deselect the effector.
- **Key D** : deselect the tool its game object, giving back control to the default scene editor window.
- **Key C:** move the entire effector, confirm by mouse click. Pressing B again cancels the movement.
- **Key V**: move the shaped region of the effector, confirm by mouse click. Pressing V again cancels the movement.
- **Key B:** delete the current select effector.

To add effectors, use the keyboard numeric keys (not the numpad ones):

- **Key 1:** add a circle effector at the mouse cursor location.

- **Key 2:** add a semi circle effector at the mouse cursor location.
- **Key 3:** add a torus effector at the mouse cursor location.
- **Key 4:** add a semi torus effector at the mouse cursor location.
- **Key 5:** add a box effector at the mouse cursor location.
- **Key 6:** add a multi effector at the mouse cursor location.

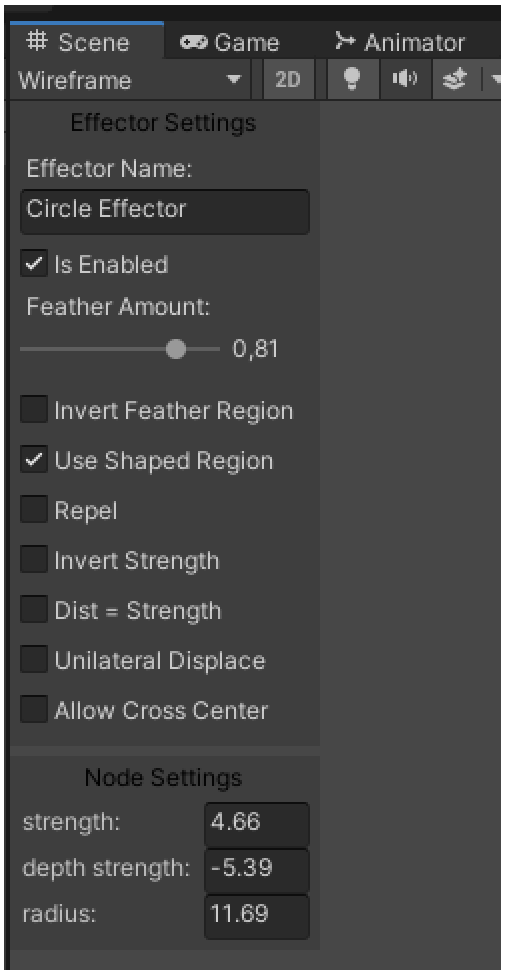
The mouse is used to manipulate and adjust effectors:

- **Left mouse button:** selects effectors and selects handles. When a handle is selected you can drag it by holding the left mouse button. The left mouse button is also used for adding/inserting points to the shaped region and multi effector paths.
- **Right mouse button:** a single click is used to remove points from the shaped region and the multi effector path.

The control keys are also shown as reminder in the inspector window when the tool is selected.

5.2.2 Effector property controls

A GUI in the scene window pops up when an effector is selected. You can adjust any property of the effector inside of this GUI. An example is given in Figure 12 and Figure 13. You can adjust all settings to your liking (see 4.2).



Effector Settings

Effector Name:

Circle Effector

[✓] Is Enabled

Feather Amount: 0,81

[ ] Invert Feather Region

[✓] Use Shaped Region

[ ] Repel

[ ] Invert Strength

[ ] Dist = Strength

[ ] Unilateral Displace

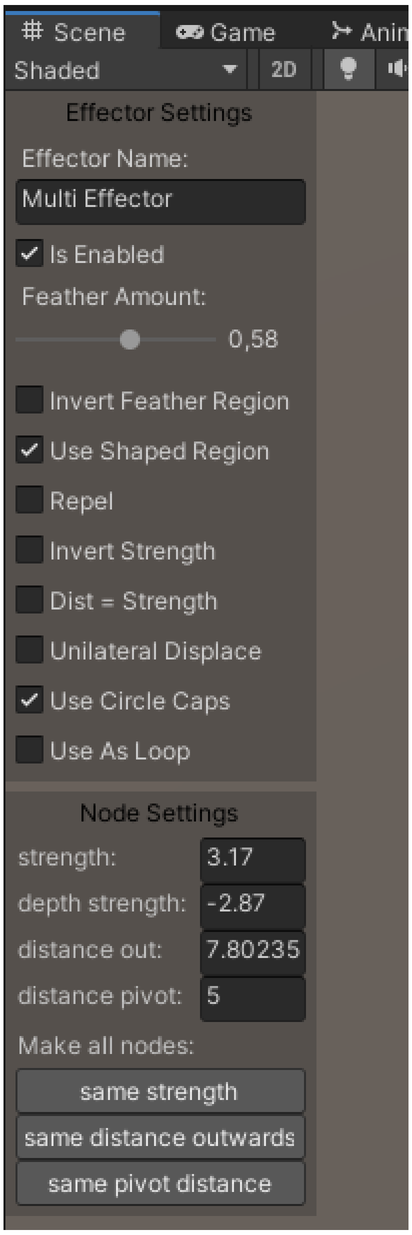
[ ] Allow Cross Center

Node Settings

strength:	4.66
depth strength:	-5.39
radius:	11.69

Figure 12: Effector settings GUI pops up when an effector is selected. In this case the Circle Effector is selected. You can adjust any setting here. Generally speaking the Effector Settings are similar for

every effector, but some effectors have an option extra. The node settings is where you adjust strength values. Some effectors have different node properties which you can also adjust.



Effector Settings

Effector Name:

Multi Effector

✓ Is Enabled

Feather Amount:

0,58

☐ Invert Feather Region

✓ Use Shaped Region

☐ Repel

☐ Invert Strength

☐ Dist = Strength

☐ Unilateral Displace

✓ Use Circle Caps

☐ Use As Loop

Node Settings

strength:	3.17
depth strength:	-2.87
distance out:	7.80235
distance pivot:	5

Make all nodes:

same strength

same distance outwards

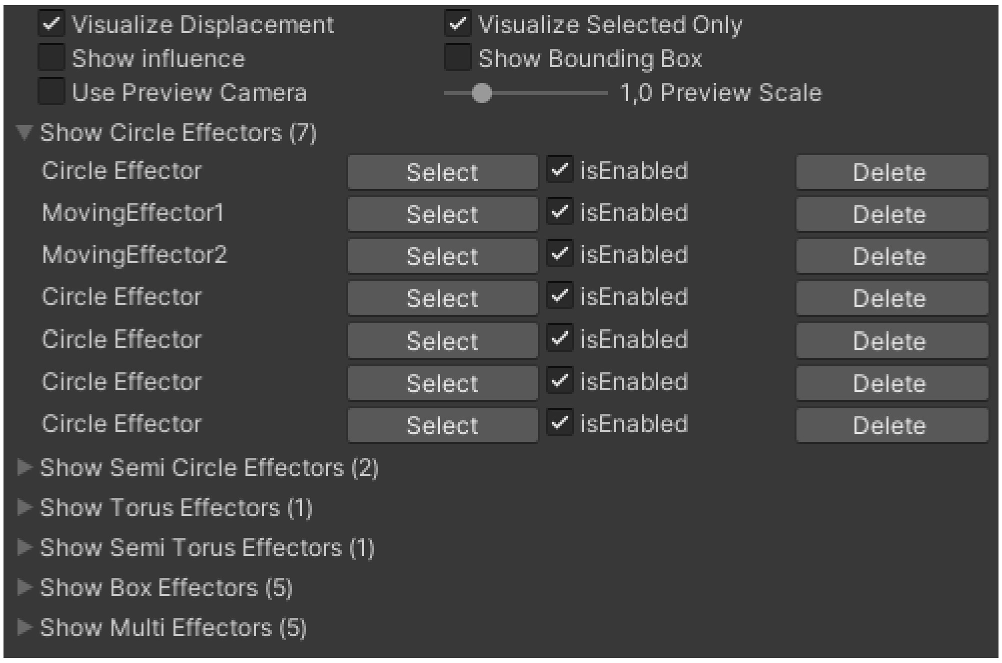
same pivot distance

Figure 13: The multi effector is selected in this case. It can be seen that the effector settings are mostly similar to the circle effector of Figure 12. The Node Settings are specialized for the multi effector, this allows for adjusting the strengths and other settings for every individual node. The node settings only pop up when you have a path node selected.

5.2.3 Inspector controls

The inspector adds some nice functionality to make creating effectors easy. In Figure 14 is a screenshot of the inspector shown. The settings on top part make visualizing the effector easy. For example, you can visualize displacement when moused over an effector. Show a preview camera in the bottom right corner and see how the camera is displaced at the mouse curser its location (see Figure 15). Note that this displacement is instantaneous in editor mode, the smoothing happens in run time.

Additionally it comes with a fold out menu for the effector types created in the current scene. The foldout menu makes it easy to look up effector by name, select, delete and enable effectors.



✓ Visualize Displacement ✓ Visualize Selected Only
☐ Show influence ☐ Show Bounding Box
☐ Use Preview Camera 1,0 Preview Scale

▼ Show Circle Effectors (7)

Circle Effector	Select	✓ isEnabled	Delete
MovingEffector1	Select	✓ isEnabled	Delete
MovingEffector2	Select	✓ isEnabled	Delete
Circle Effector	Select	✓ isEnabled	Delete
Circle Effector	Select	✓ isEnabled	Delete
Circle Effector	Select	✓ isEnabled	Delete
Circle Effector	Select	✓ isEnabled	Delete

▶ Show Semi Circle Effectors (2)
▶ Show Torus Effectors (1)
▶ Show Semi Torus Effectors (1)
▶ Show Box Effectors (5)
▶ Show Multi Effectors (5)

Figure 14: This is the inspector window when the effector tool is selected. It comes with useful settings at the top such as visualizing the displacement or a preview camera, which shows the effector its instantaneous displacement. It also gives a foldable overview of the effectors added in the game object, with the amount added shown between parenthesis. It shows the name of the effector followed by a selection button to select the effectors easily. It is also possible to enable or disable the effector.

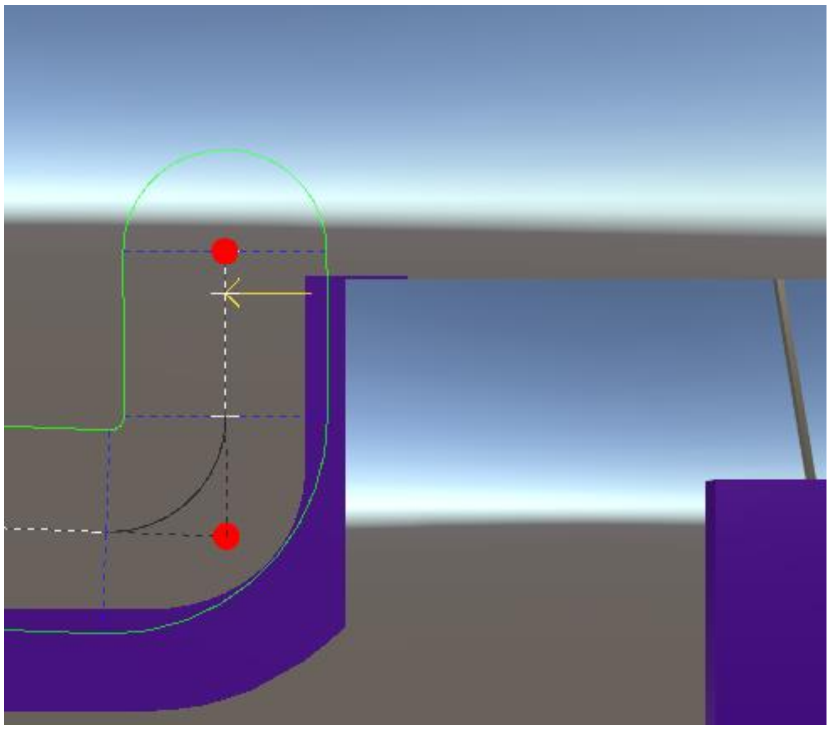


Figure 15: The camera preview is enabled. The mouse cursor is at the start of the yellow arrow and the camera is at the arrow head. This is shown in a preview window in the bottom right corner.

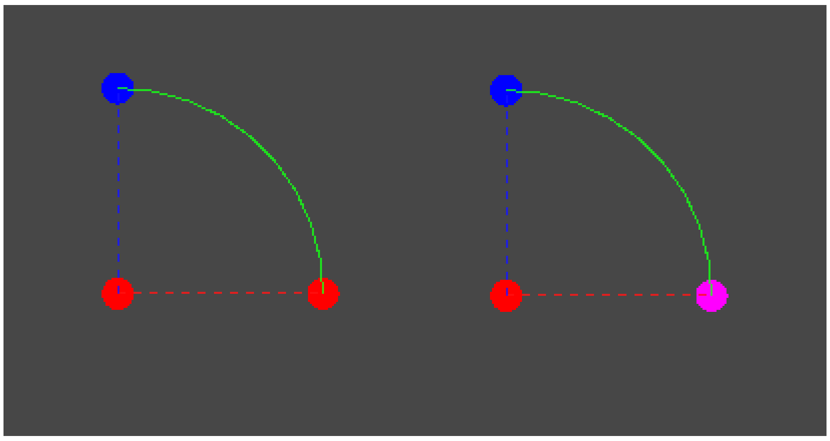
5.3 Tutorial

This section is a short tutorial on how to create and manipulate effectors. See 5.1 on how to add the tool to the hierarchy.

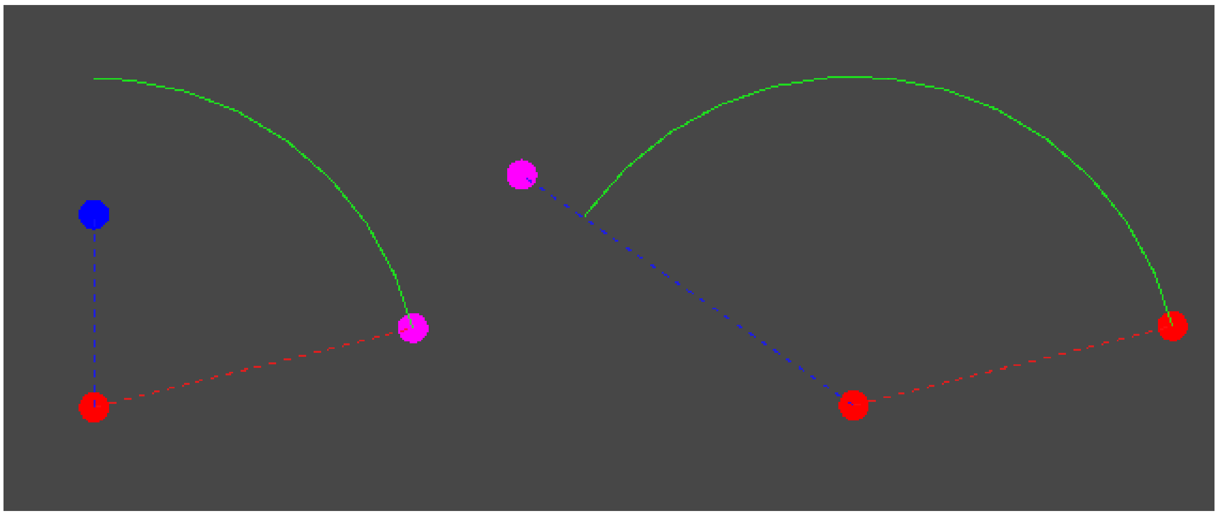
5.3.1 Create an effector

When the tool is selected and the focus is on the scene window, we can create a semi circle effector by pressing “2”.

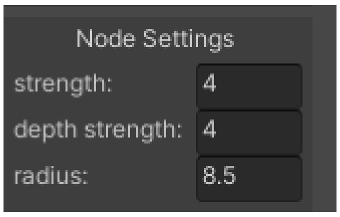
You get a default effector added in the screen at the mouse location. The red disc shapes handles are the main handles of an effector, for circle/torus effectors these determine the center and the radius. The blue handles have secondary functionality, for the semi circle/torus it determines the arc the effector makes. For box effectors these determine the distance outward.



When moused over a handle it appears white. When left moused clicked while over them it selects the handle. A selected handle appears magenta in color.



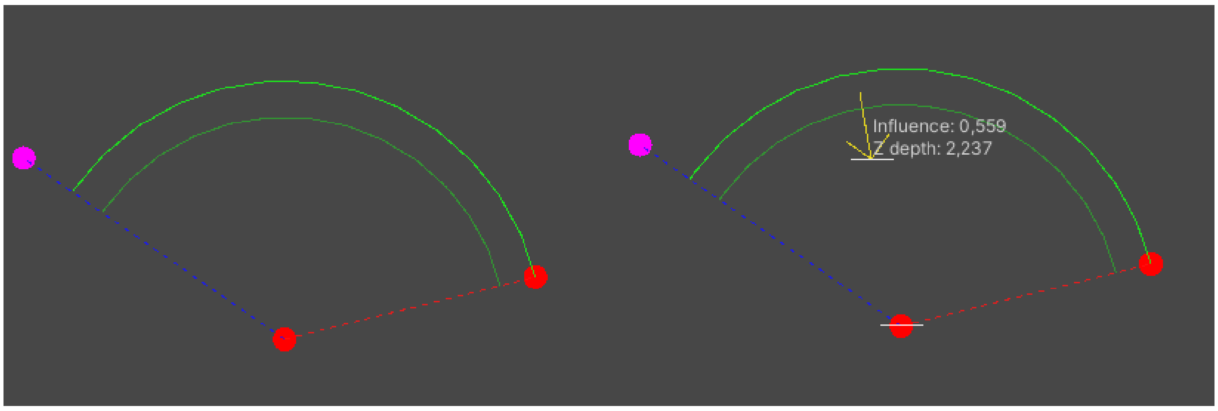
By dragging the selected handles we can shape the effector.



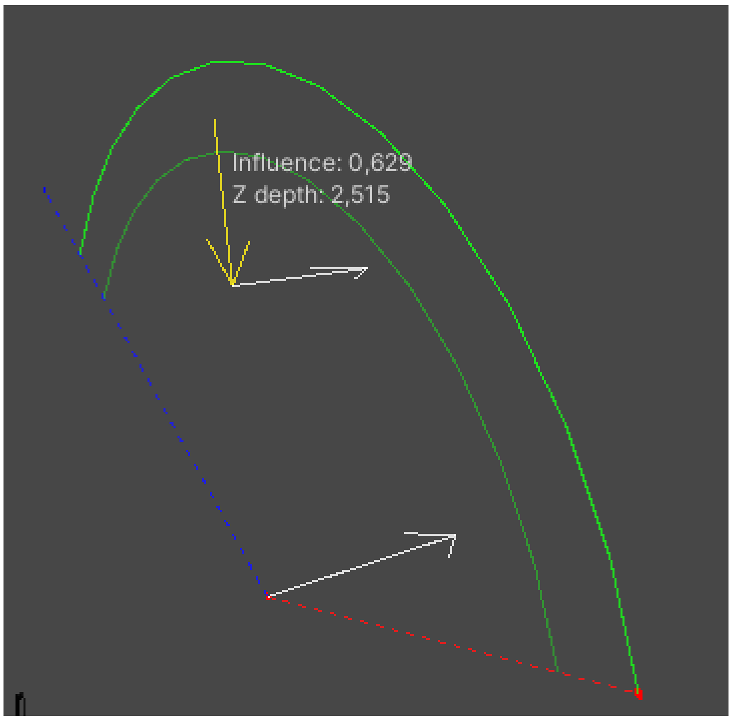
Node Settings

strength:	4
depth strength:	4
radius:	8.5

We can set some arbitrary strengths to the effector.



A darker green line is shown when setting the feather amount to a value between 0 and 1. By setting "Show Influence" to true in the inspector we can mouse over the effector and see how the feather changes the influence. The depth is also shown in text as this is more convenient in 2D view then to rotate the camera in 3D to see how much depth is added.

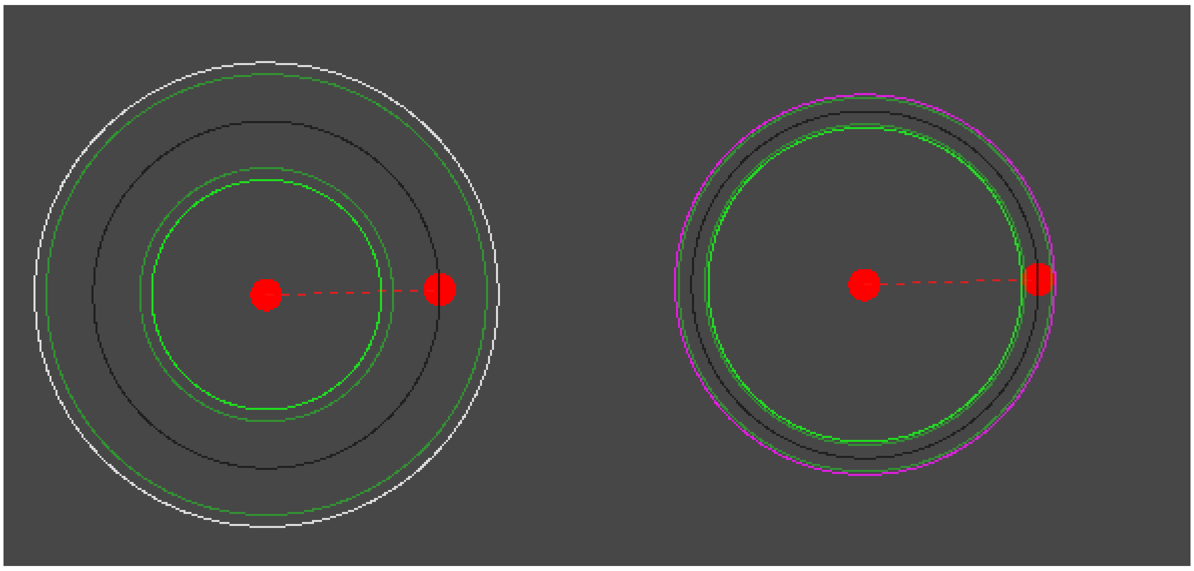


The same view but now in 3D. A positive depth is a zoom in effect. The white arrow indicates the depth strength. The white arrow connected at the yellow arrow show the depth displacement at that current position.

You can also enable the preview camera to see what effect the displacement has on the camera position.

This is already a working effector. This is the most basic functionality for effectors which is roughly the same for all other types of effectors.

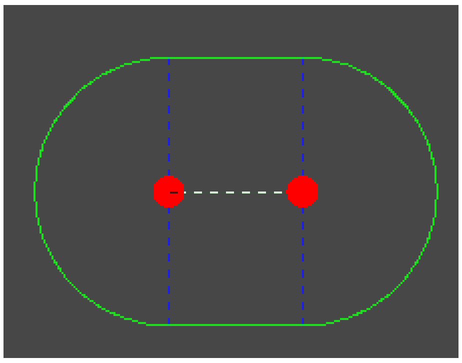
The torus effector has a special handle. The outer ring of the torus effector can be clicked and dragged to set the minor radius of the torus (also for the semi torus).



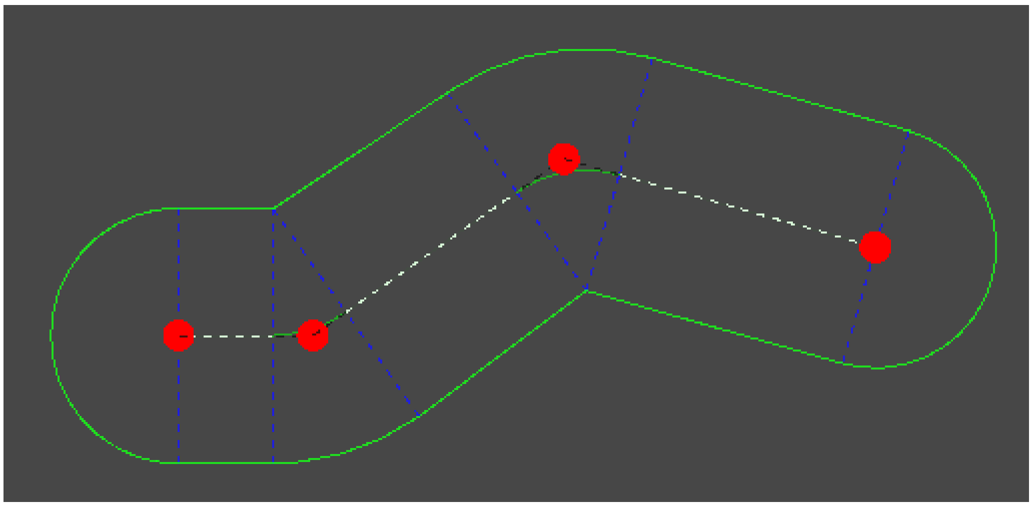
5.3.2 Create a multi effector

The multi effector is different from the other effectors because we can set strengths for each individual node. This is a short tutorial on how to create a multi effector and what the icons mean!

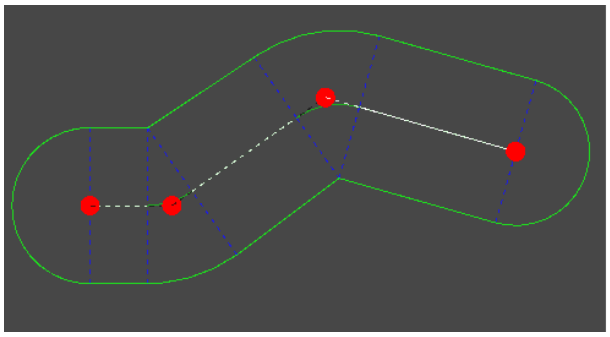
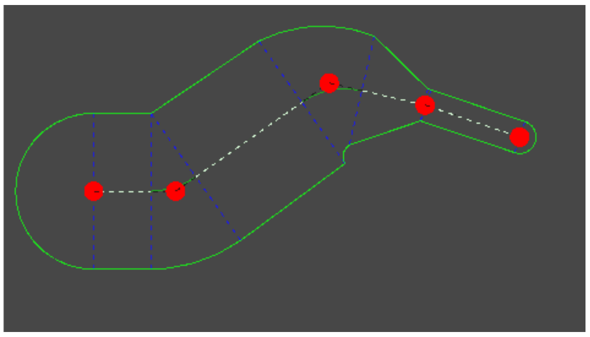
Press 6 to add a multi effector



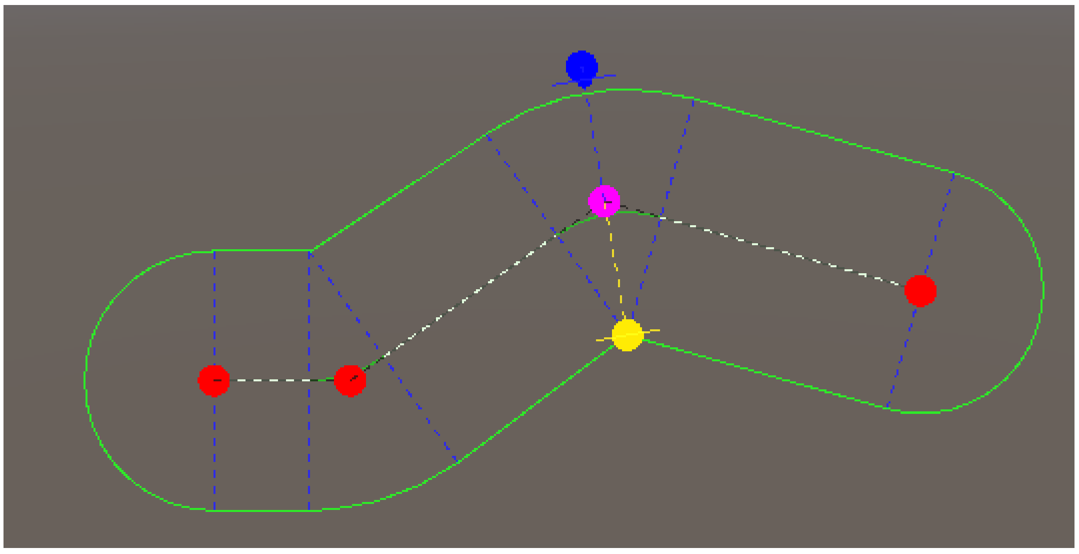
This is the default multi effector, it has circle caps and a box effector in the middle. **To add nodes you must not have selected a handle (press A if you have selected a handle)!!!** Press the left mouse button anywhere to add new nodes.



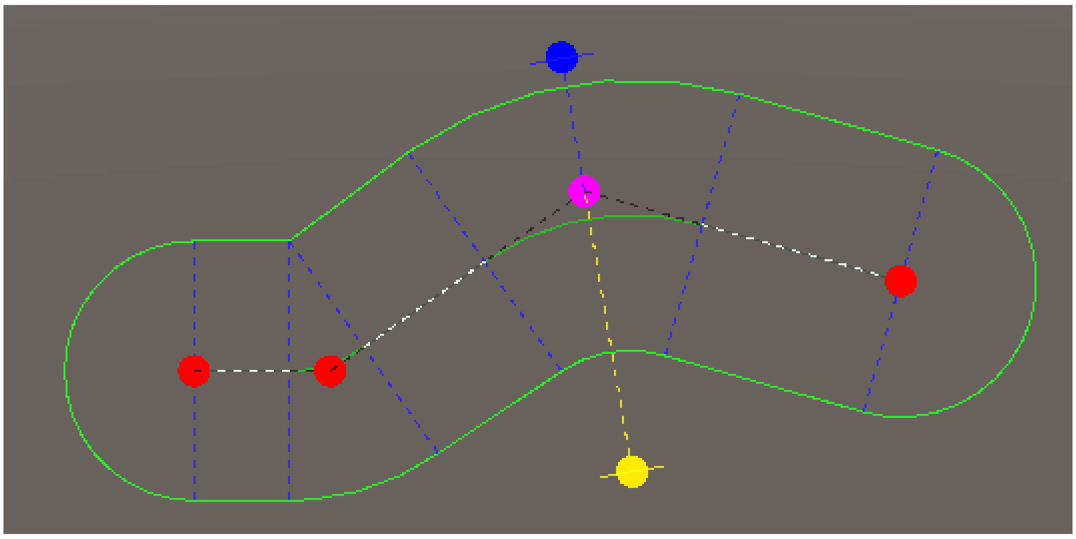
In this case we added 2 extra nodes to the end. You can also insert nodes by hovering the mouse cursor over the line between two points. When it is fully white you can insert a point by clicking the left mouse button. You can remove points by right clicking while over them. Remember you can also undo any operation.

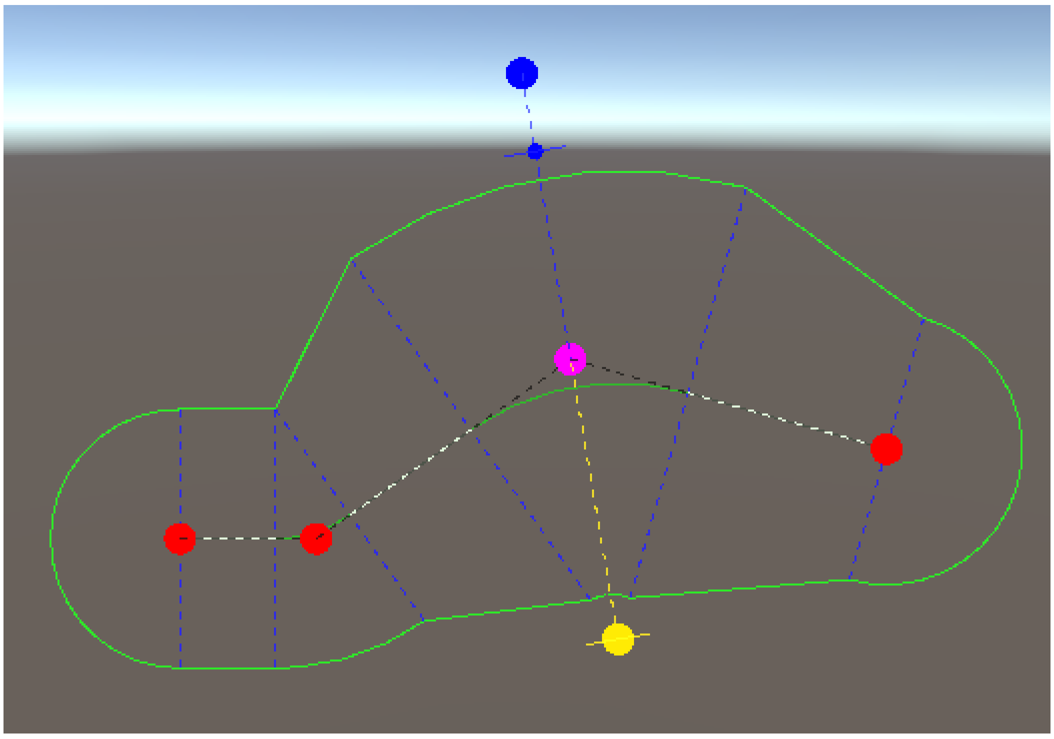
The multi effector has a special handle. When selected there are two more handles shown. The blue node determines the desired outward distance of the multi effector. The yellow one determines the desired pivot location of the (specialized) semi torus effector. This is the leading factor for outward distances. The outward distance and strengths are interpolated from node to node over the effectors.



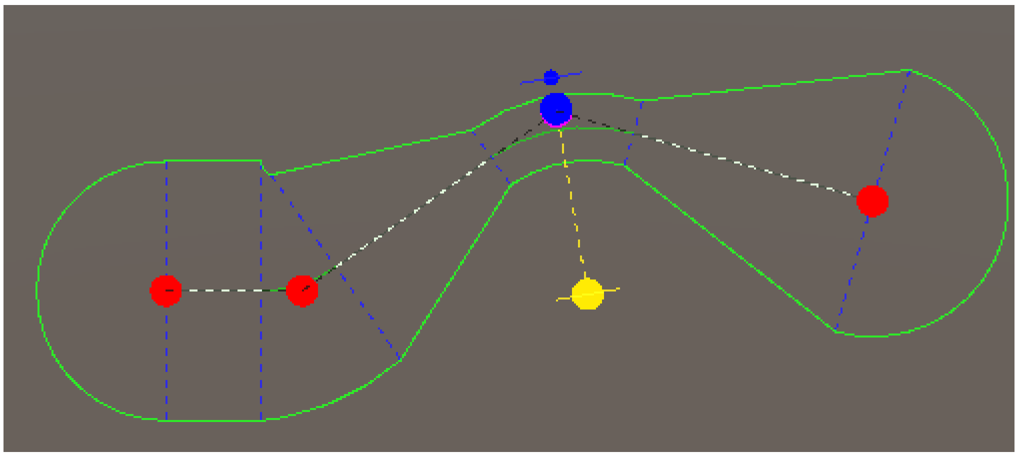
The blue bar with tiny disc, that is perpendicular to the direction of the blue node, is the current outward distance. The current outward distance is measured from the magenta node to the blue node. In the image above you see that it does not matches the blue node. By dragging the desired pivot (yellow) outwards we can get the blue bar match the blue node. The real outward distance and real pivot positions may not line up to the desired ones. This happens, because the effector is calculated in such a way that it always has a valid solution.



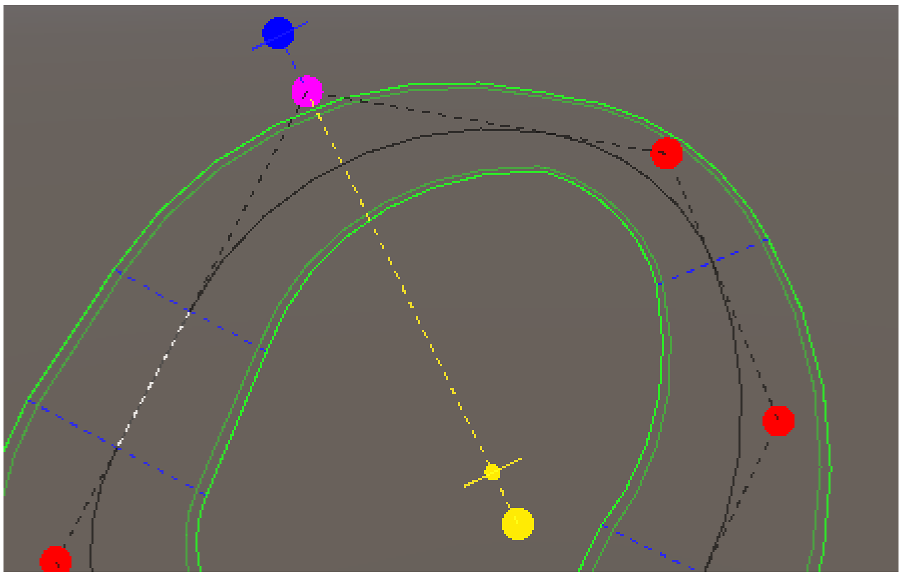
We can increase the desired distance again, but it caps at some point. This happens because of several factors. **The main reason for limiting the distance and pivot is that it makes interpolation and proper connecting of the effectors impossible if we exceed this distance.**



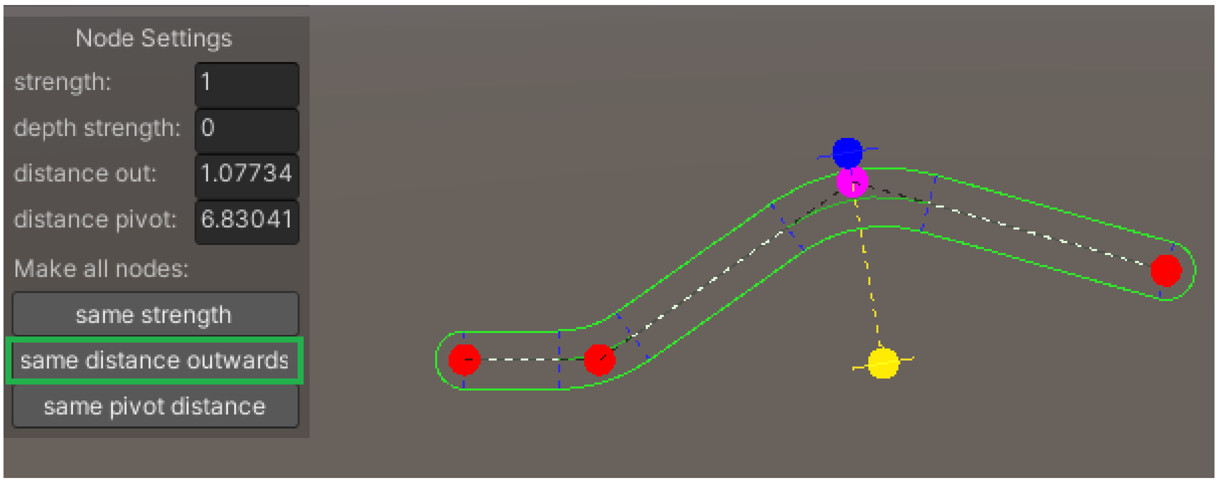
The same clamping happens if we would want to make the outward distance very small at this node. You can get a good view of the blue bar and tiny disc here.



The desired pivot distance can also not match the true pivot distance as seen from the image below. The yellow bar with tiny disc is closer to the magenta node than the large yellow handle. This means that the center of rotation of this interpolated semi torus is at the yellow bar with tiny disc!



We can set all outward distances across all nodes equal by pressing the button "same distance outwards".



Node Settings

strength: 1

depth strength: 0

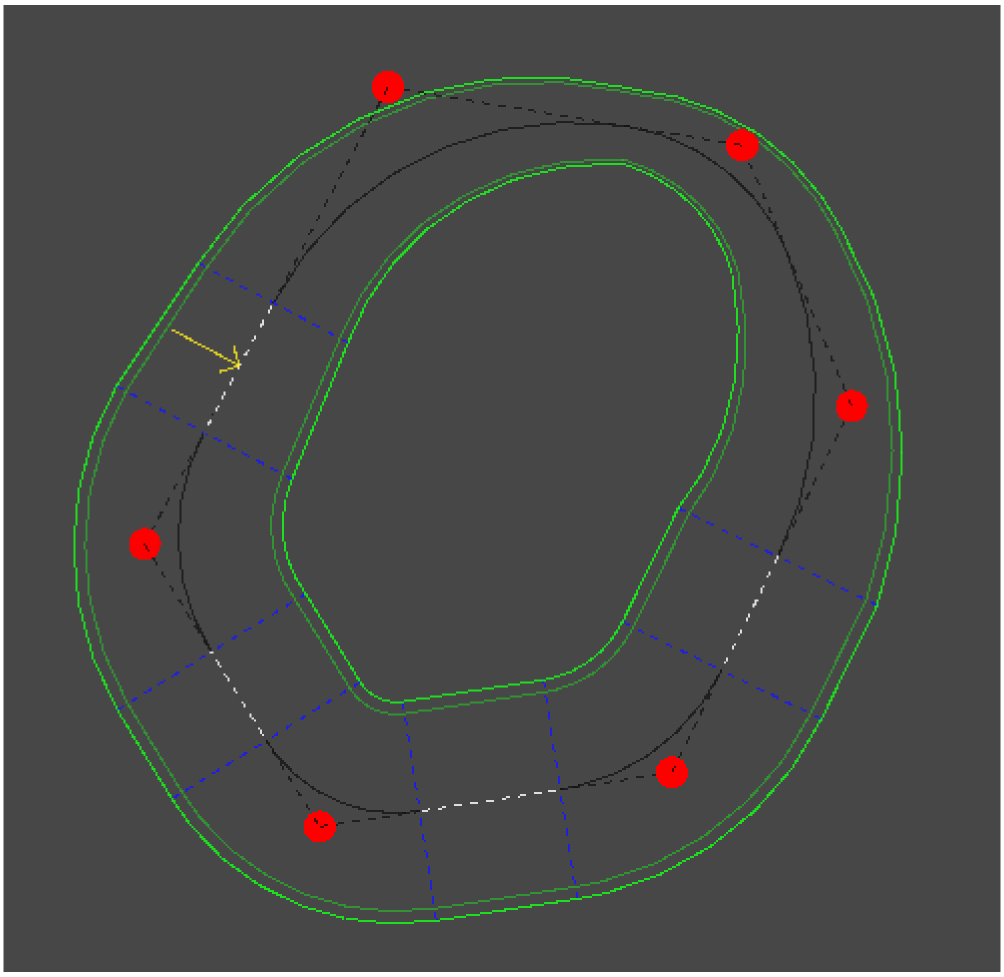
distance out: 1.07734

distance pivot: 6.83041

Make all nodes:

This can also be done for the pivot distances. It is possible to set strength values for all nodes individually or you can copy the selected node value to all other nodes.

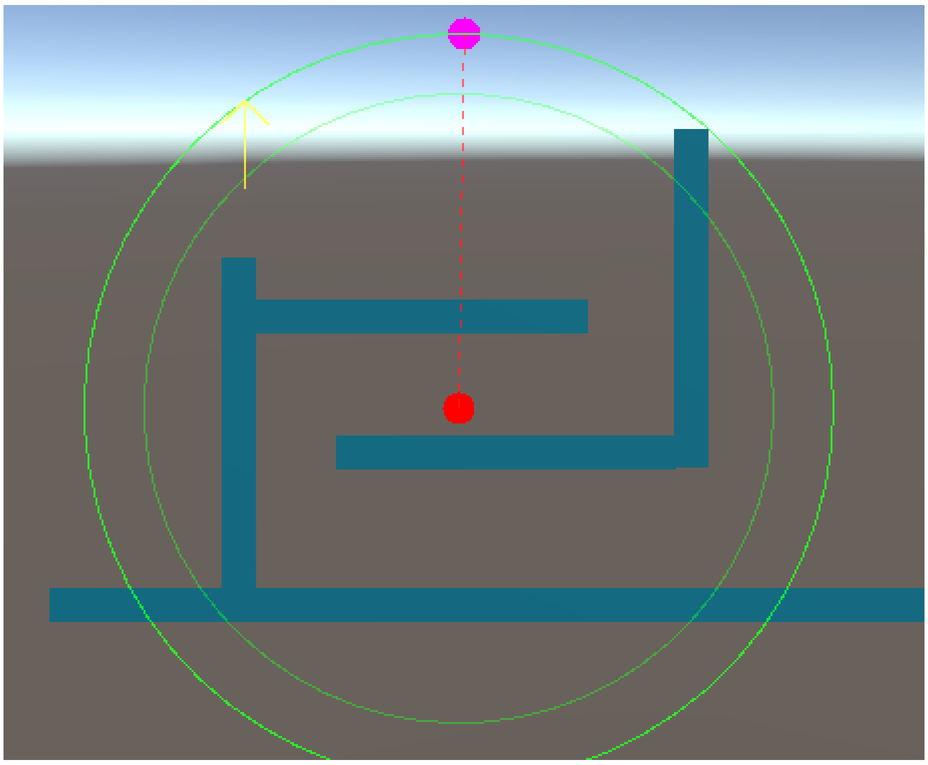
You can create all sort of weird shapes with the multi effector. Remark that the semi torus has a variable minor radius which is interpolated between segments. The one below has loop enabled.



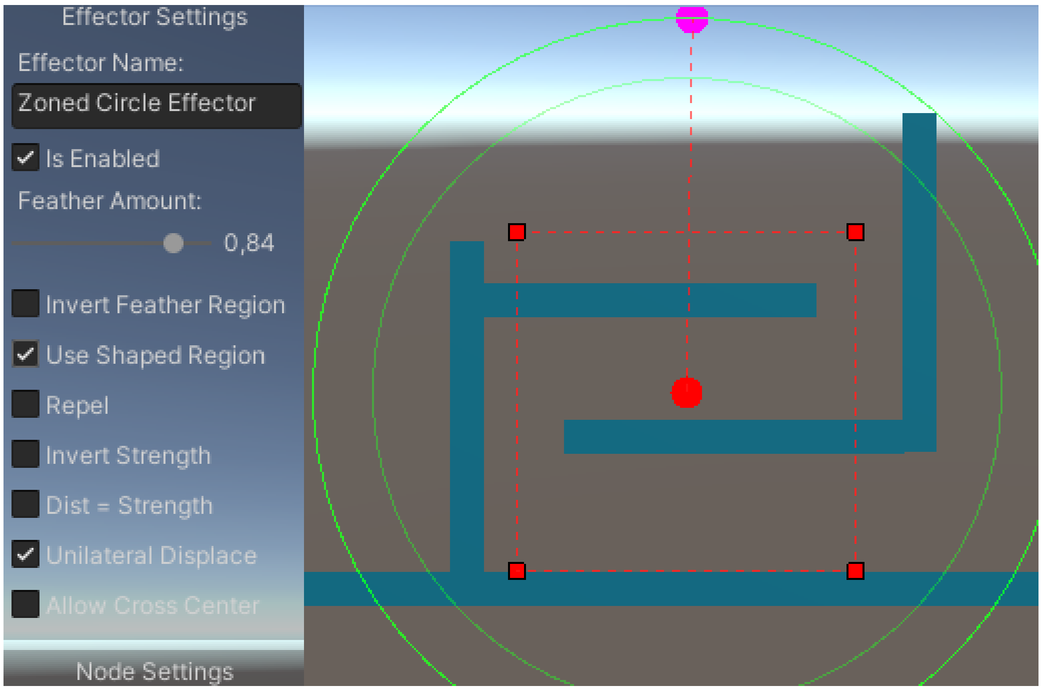
The multi effector is effective at creating rail effects for the camera! Set **dist = strength** to true and have a high feather amount. That is all there to it!

5.3.3 Shaping a shaped region

The shaped region is very easy to create. We start with an area in the game world we want to have affected by an effector. In this example we have some platforming room with a zig zag path through it. We place a circle effector over the region that we want to have affected.



We want only to have the path inside the zig zag affected. The circle exceeds the area we want to have affected by a great amount. The shaped region will is going to define the region where the effector is active. To enable the shaped region, tick the “Use Shape Region” box in the effector settings. This shows a square outline with square nodes as default.



Effector Settings

Effector Name: Zoned Circle Effector

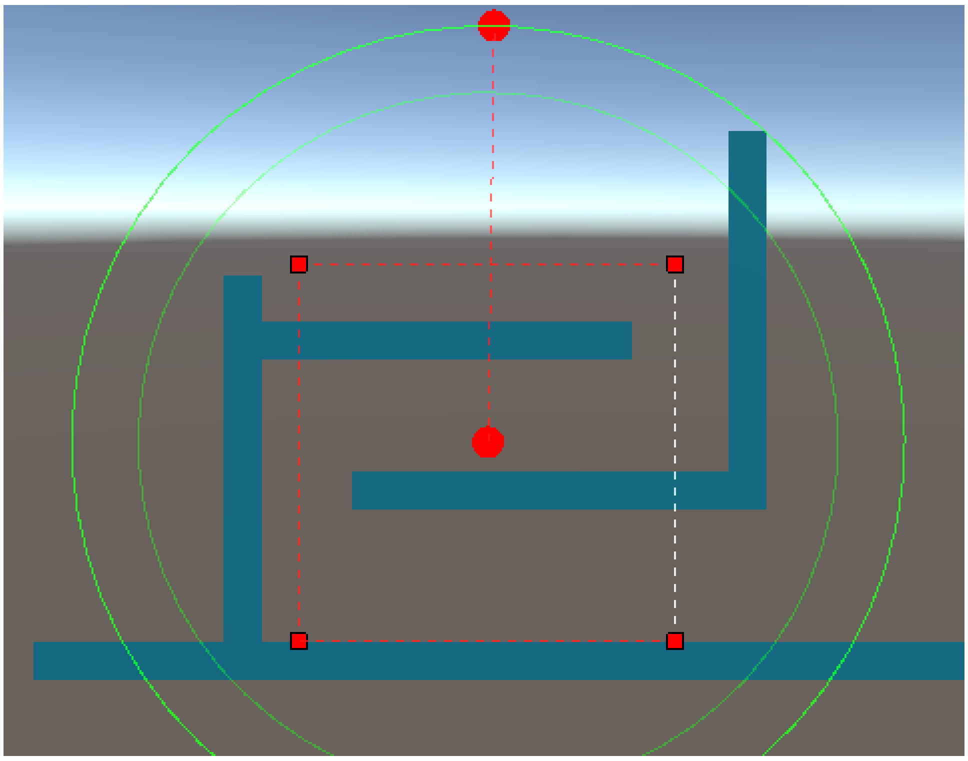
- [x] Is Enabled

Feather Amount: 0,84

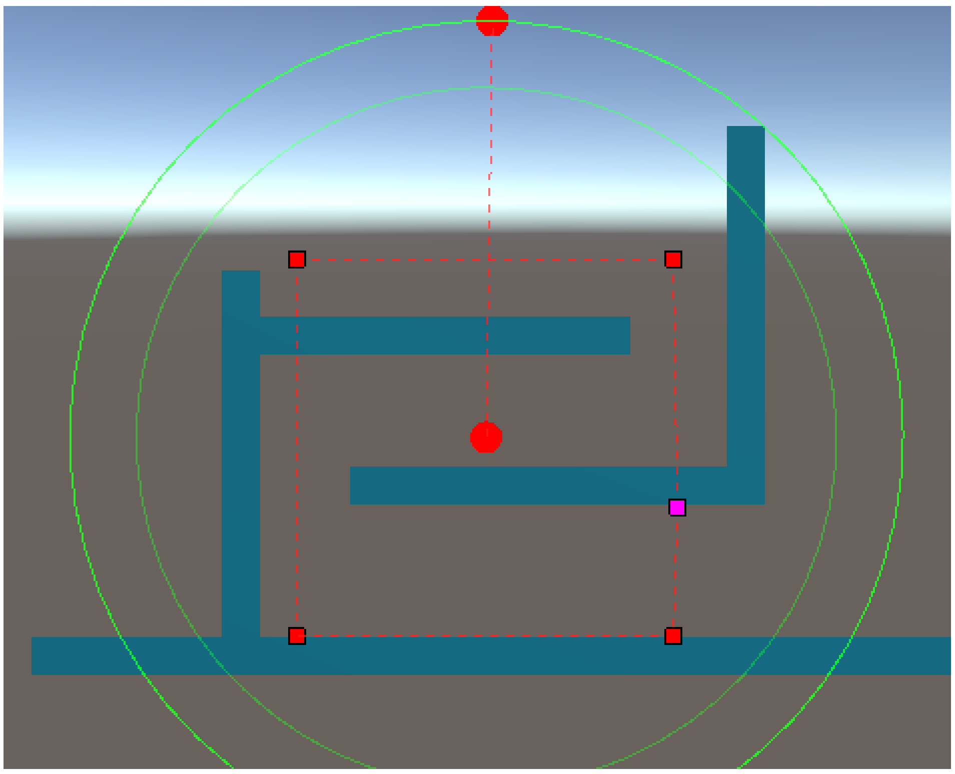
- [ ] Invert Feather Region
- [x] Use Shaped Region
- [ ] Repel
- [ ] Invert Strength
- [ ] Dist = Strength
- [x] Unilateral Displace
- [ ] Allow Cross Center

Node Settings

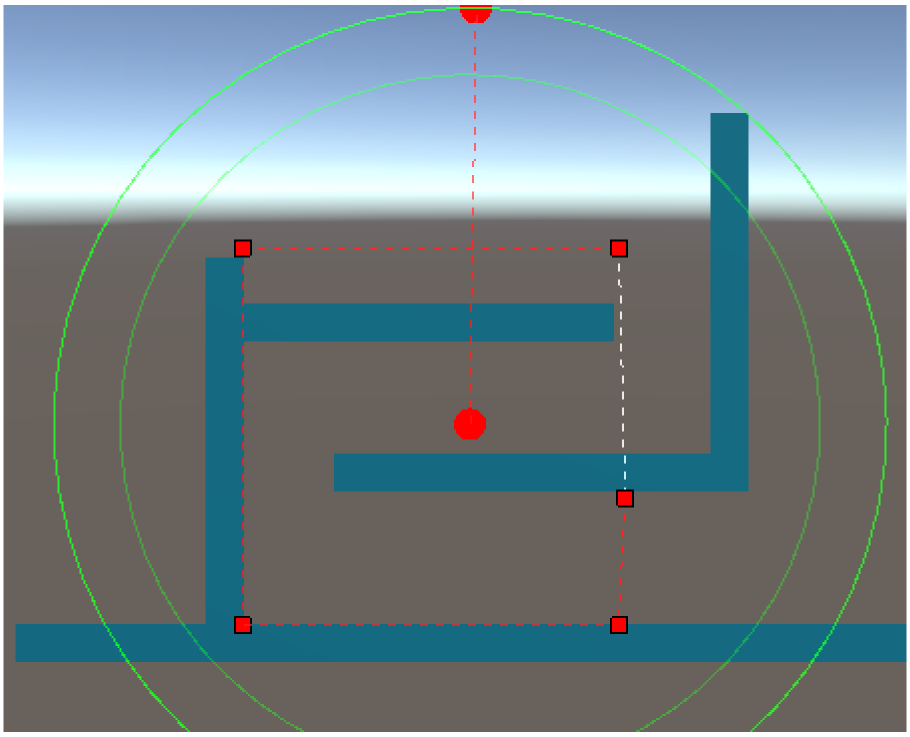
It’s a coincide that the box lines up nicely at the lower part. To insert points mouse over the red dashed lines connected between two square nodes.



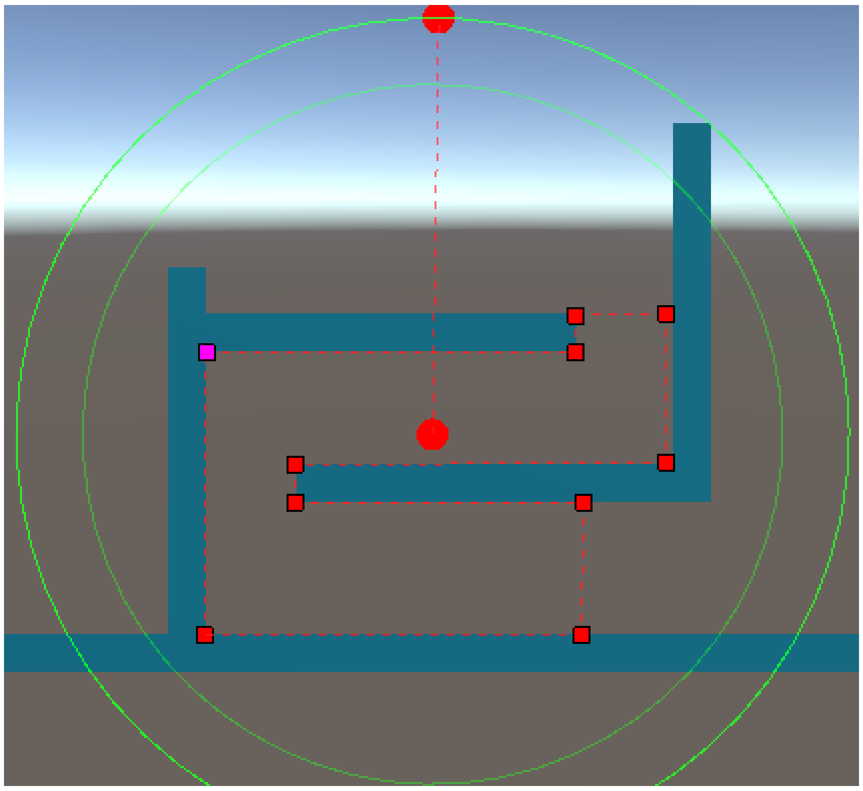
Left click once to get the node inserted. You can drag it around right away if you hold the left mouse button.



We can drag the whole shaped region by pressing “V”. Drag the mouse to find a good position and click left to confirm.



Now we can add more points to line out the area we want to have affected.



That is all there is to shaped regions!

A final remark on creating effectors is that you should not start with an effector and build something around it! Start with the layout first and add effectors later!

6 Dynamic Camera Controller

This section covers how to set up your own camera controller by taking the DynamicCameraControllerMinimal.cs as example. It is slightly more advanced than the version mentioned in 3.6.

6.1 Minimal setup

The camera controller is the main drive to have the camera work with tracking and have it also get affected by the effectors.

A camera controller only works if it inherits the MonoBehaviour class. We need the following fields:

```
public class DynamicCameraControllerMinimal : MonoBehaviour
{
	// takes care of smoothing out the montion of the camera towards the target
	public DynamicCameraTracker cameraTracker;
	// takes care of smoothing the orthograpic size to a new value
	public MassSpringDamperTracker1D orthoSizeSmoother;
	// contains specialized functions
	public DynamicCameraFunctions dynamicCameraFunctions;
	// rigidbody2D as target we want to track
	public Rigidbody2D targetRigidbody;

	// transform that is the camera or is the parent of the camera. It is better to
	// have the camera parented to the rig as this allows for the
	// camera shake effect.
	public Transform cameraRig;
	// camera attached to the rig
	public Camera currentCamera;
	// offset added to the final position to shift the camera
	public Vector2 cameraTargetoffset;
	// z position of the camera at initialization
	public float cameraRigPositionOffsetZ = 0;

	// We need to calculate the acceleration as deltaV / deltaT
	private Vector2 targetAcceleration = Vector2.zero;
	// We also need to keep track of the previous velocity to calculate
	// the acceleration
	private Vector2 previousTargetVelocity = Vector2.zero;
```

For initialization we need:

```
// Initialize camera and camera offsets
void Start()
{
	cameraRigPositionOffsetZ = cameraRig.position.z;
	cameraTracker.SetInitialConditions(cameraRig.position, Vector3.zero);
}
```

We need to calculate the acceleration of the target in the FixedUpdate() as:

```
private void FixedUpdate()
{
	// calculate acceleration like this (must be in fixed update):
	targetAcceleration = (targetRigidbody.velocity - previousTargetVelocity) /
	                            Time.fixedDeltaTime;
	previousTargetVelocity = targetRigidbody.velocity;
}
```

Then in the LateUpdate() we determine the target velocity that is compensated for collisions, the effector displacement, lead/lag offsetting and finally updating the tracker. This looks like:

```
void LateUpdate()
{
	Vector3 targetPosition = targetRigidbody.transform.position;
	// dont use Rigidbody2D.position to get the position!
	Vector2 targetVelocity =
dynamicCameraFunctions.LookAheadTrackingVelocityCompensation(targetRigidbody);
	// this has to be called before the effector displacement

// Lead/Lag (type 1) by compensating the velocity. This version of calculating lead
lag is insensitive to the target position so this can be done before the effector dis-
placement. However it can cause jitter at smoothTimes < 0.1.
targetVelocity = dynamicCameraFunctions.LeadLagTargetByVelocityCompensation(
targetVelocity, cameraTracker);


DCEffectorOutputData displacementOutput;

// displace the target position by the effectors, the order of calls is important as
we can targetPosition to be a lead/lag extrapolated
	targetPosition = dynamicCameraFunctions.DisplaceByEffectors(targetPosition, ref
targetVelocity, ref targetAcceleration, out displacementOutput);  // we need the
displacement for orthographic cameras as the depth is stored in the Z component

// Lead/Lag (type 2) by displacing the target by the current velocity and scale the
influence of it by the effector influence.

// targetPosition = dynamicCameraFunctions.LeadLagTargetByPositionCompensation(target-
Position, targetVelocity, (1 - displacementOutput.influence));

// the order is important: if we would offset by lead/lag by position first and then
do effector displacement, we displace from the perspective of the extrapolated posi-
tion.
// This can give unwanted behaviour!


	// displace by effector first then displace by lead/lag scaled by
	// (1 - influence factor) so that lead/lag is less with higher influences
	targetPosition =
dynamicCameraFunctions.OffsetTargetPositionLeadLag(targetPosition, targetVelocity * (1
- displacementOutput.influence));
	// the order is important: if we would offset by lead/lag first and then do
	// effector displacement, we displace from the perspective of the
	// extrapolated position. This can give unwanted behaviour!

	Vector3 cameraTargetPosition = new Vector3(targetPosition.x, targetPosition.y,
targetPosition.z + cameraRigPositionOffsetZ);

	// apply a final offset to the target
	cameraTargetPosition = cameraTargetPosition + (Vector3)cameraTargetoffset;

	// Update the camera rig with critical damped step
	cameraRig.position = cameraTracker.CriticalDampedStep(cameraTargetPosition,
targetVelocity, targetAcceleration, Time.deltaTime,
MassSpringDamperFunctions.ClampType.Circle);

	// NOTE: use cameraTracker.CriticalDampedStableClampStep(…) if you want to
	// using clamping for the positional tracker.
}
```

That is all to it! Read the source code carefully if you like to know more!

6.2 Setup for smooth zoom in orthographic mode

Although zooming works in both perspective mode and orthographic mode, the latter mode is instantaneous. It requires some extra lines of code to make smooth zooming in orthographic mode possible.

Below is a snippet of the DynamicCameraController.cs where smoothing is applied to the orthographic size change.

```
// Smooths the orthograpic size to a new value
public MassSpringDamperTracker1D orthoSizeSmoother;
// Simple initialization of the orthographic size.
float intialCameraOrthoSize = Camera.main.orthographicSize;

LateUpdate()
{

       // displacement is the displacement from effectors
       Vector3 cameraTargetPosition;
       // no displacement in the z direction for orthographic cameras
       cameraTargetPosition = new Vector3(targetPosition.x, targetPosition.y,
                                          cameraRigPositionOffsetZ);
       // the target orthographic size is determined based on the displacement in the
       // z direction, the – sign is required if we want to have
       // a negative value to be zooming out
       float targetOrthoSize = Mathf.Max(intialCameraOrthoSize -
displacementOutput.displacement.z * DCEffector.depthToOrthgrapicSizeFactor, 0.1f);

       // set the new ortho size value by updating it as a critical damped system
       currentCamera.orthographicSize = orthoSizeSmoother.CriticalDampedStep(
                                          targetOrthoSize, 0, Time.deltaTime, false);


}
```

Note that initialCameraOrthoSize must be initialized in the Start() method. In the DynamicCameraController.cs it is initialized as the value of the camera that is attached to the script. There is another way of the orthographic size. One could also use:

```
// OrthographicSize smoothing type 2:
// This is another way to create a ortho graphic zoom effect.
// The (cameraRig.position.z - cameraRigPositionOffsetZ) is taken here as
// the already smoothed factor for setting the orthographic size.

// The disadvantage of this is that it also displaces the camera position on
// the Z axis, which could be something that is unwanted when you are using
// orthographic settings.
if (currentCamera.orthographic)
{
       // Set OrthographicSize by the difference of the Z positions of the camera
       // note cameraRig.position.z is already smoothed!
       float targetOrthoSize = Mathf.Max(intialCameraOrthoSize –
                                          (cameraRig.position.z - cameraRigPositionOffsetZ) *
                                           DCEffector.depthToOrthgrapicSizeFactor, 0.1f);

       // apply orthographicSize
       currentCamera.orthographicSize = targetOrthoSize;
}
```

This way the smoothTime of the Z axis of the tracker is used, which could be more convenient if you switch dynamically from ortho to perspective.

7 Effector Manager

A final important part of the package is the effector manager script: DCEffectorManager.cs. It was shortly mentioned before, but we dive into more details here.

The DCEffectorManager.cs script stores the created effectors that are created in the editor. They get pushed to a global list during run time.

7.1 Enabling / disabling

The local lists of any manager in the hierarchy are pushed to the global lists when OnEnable() is called. When OnDisable() is called, the local stored effectors are removed from the global lists, while keeping the local lists! This means that you can disable the game object with the DCEffectormanager.cs script to “disable” all the effectors without setting the IsEnabled field to false for every effector. Enabling the game object pushes the effectors back to the global list (and new entries in the local lists are added too). OnDestroy() does remove the effectors from both the local list and global list.

Note that while in play mode, you can add effectors by using the effector tool. However they are not pushed to the global lists. You can still push them to the global list by disabling and then enabling the DynamicCameraEffectorTool game object.

7.2 Manually adding effectors

You can add effectors manually programmatically by using

```
DCEffectorManager.AddEffectorToGlobalList(effector);
```

The global lists are persistent throughout scenes so you must not forget to remove the effector when you don’t need it anymore. Remove it from the list by:

```
DCEffectorManager.RemoveEffectorFromGlobalList(effector);
```

You can also add an effector to a local list and disable and re-enable the DynamicCameraEffectorTool game object. It is not recommend to add new effectors like this during run time!

7.3 Other useful functions

You can get an effector by name by using:

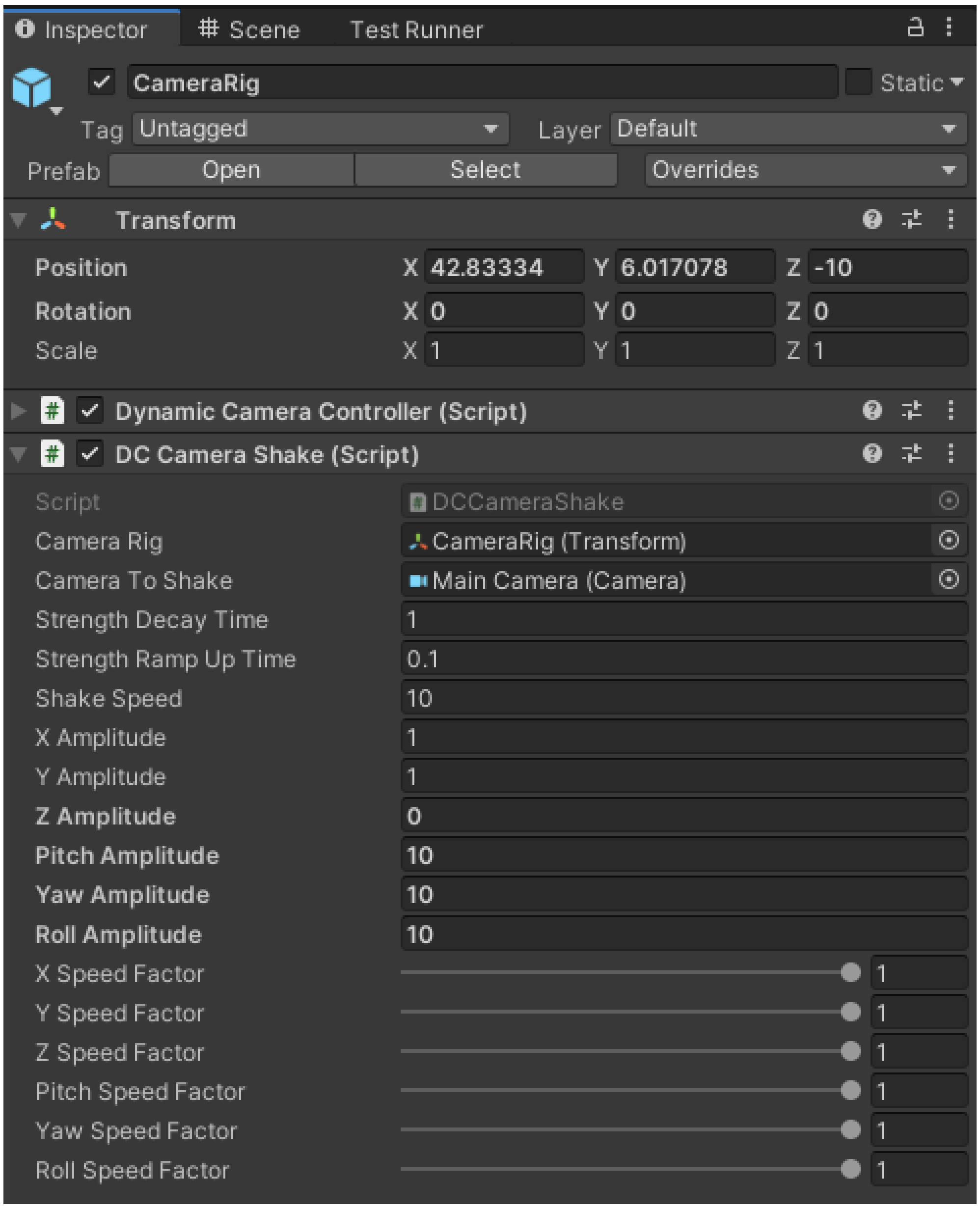
```
// get the effector by name
Effector = DCEffectorManager.GetEffectorByName(name);

// get the displacement stored in the output data
Bool = DCEffectorManager.GetDisplacementAt(Vector2, out DCEffectorOutputData);

// get the displacement and an array of the effectors where the point was inside
DCEffectors[] = DCEffectorManager.GetDisplacementAndEffectorsAt(Vector2,
                                                           out DCEffectorOutputData);
```

8 Bonus: Dynamic Camera Screen Shaker

This screen shake effect script is added as bonus. It is a frame independent camera shake effect that can translate and rotate. It is an additive screen shaker with exponential decay. This means that if you apply a strength multiple times to the shaker, it stacks and amplifies the shake. The exponential decay makes it feel more intense at high strength and more soft at low values. This screen shaker can also be used for 3D projects.

8.1 Parameters

The CameraRig.prefab has the DCCameraShake.cs script attached to it. In this section we got over the parameters of the camera shaker and what they do.



CameraRig

Tag Untagged | Layer Default

Transform

	X	Y	Z
Position	42.83334	6.017078	-10
Rotation	0	0	0
Scale	1	1	1

Dynamic Camera Controller (Script)

DC Camera Shake (Script)

Script	DCCameraShake
Camera Rig	CameraRig (Transform)
Camera To Shake	Main Camera (Camera)
Strength Decay Time	1
Strength Ramp Up Time	0.1
Shake Speed	10
X Amplitude	1
Y Amplitude	1
Z Amplitude	0
Pitch Amplitude	10
Yaw Amplitude	10
Roll Amplitude	10
X Speed Factor	1
Y Speed Factor	1
Z Speed Factor	1
Pitch Speed Factor	1
Yaw Speed Factor	1
Roll Speed Factor	1

Figure 16: The camera shake script in the inspect view when the CameraRig.prefab is selected.

The parameters are:

- **Camera Rig**: this is the parent of the camera that is going to be shaken. This is used as the reference point for offsetting the camera.
- **Camera To Shake:** the camera that will be shaken.
- **Strength Decay Time:** The time it takes until we go from max shake to no shake.
- **Strength Ramp Up Time:** is the time it takes to go from 0 to max strength.
- **Shake Speed:** The speed at which the camera shakes, the value 10 is nice default shake
- **Amplitude (XYZ):** The amplitude in meters of the camera.
- **Amplitude (pitch/yaw/roll):** Represents the amount of rotation in degrees the camera can have to one side. An amplitude of 10 degrees means that we can rotate from -10 to 10 degrees.
- **Speed factors (6DOF):** scales the appropriate parameters by a factor of the shake speed making it shake slower on that axis.

8.2 How to use

To use the camera shaker we first must get the DCCameraShake component. This can be done in two ways. The first is to make a public reference to the object that has the DCCameraShake component attached. The second way is to get this component from the parent of the active camera, which can be accessed at any time from `Camera.main` or from `Camera.allCameras`. However you are free to attach the DCCameraShake script to any other game object you like. The following code snippet is taken from the demo script CameraShakeUsage.cs.

```
public GameObject shakerObject;
[Range(0,1)]
public float shakeStrength = 0.5f;

private DCCameraShake cameraShaker;

// Start is called before the first frame update
void Start()
{
	cameraShaker = shakerObject.GetComponent<DCCameraShake>();
}

// Update is called once per frame
void Update()
{
	if (Input.GetKeyDown(KeyCode.Q))
	{
		cameraShaker.AddShake(shakeStrength);
	}
}
```

In this example we used the Q key to add shake. You can change this yourself to add shake on damage taken or when the ground is rumbling for example. Note that the strength is normalized for the shaker. This means that the minimum strength is zero and the maximum is one! So setting AddShake(100.0f) corresponds to AddShake(1.0f). Adding a strength of 0.1 multiple times before the shake is over, will result in a higher shake strength.

You can test the screen shake in the demo scenes by pressing Q. However you have to adjust the parameters in the inspector itself.

9 Demos

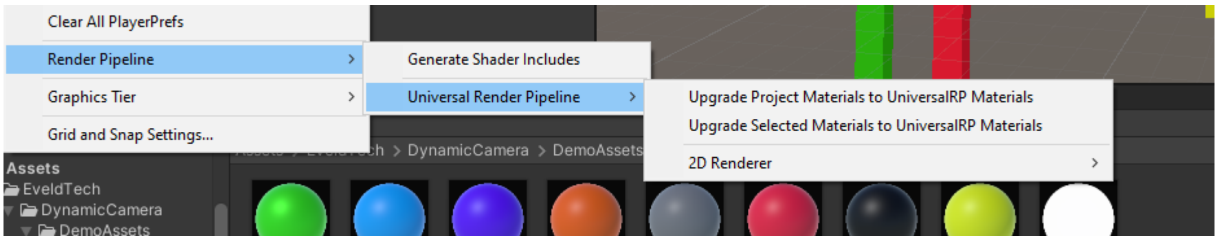
9.1 Demo scenes

There are two demo scenes in the package:

- SpeedScene.unity
- SlowScene.unity

The demo scenes in the package are used with the built-in renderer. If the package is opened in a project with the Universal RP, make sure to upgrade the materials in the folder to Universal RP materials. This is done as follows.

Select the materials BasicMat up to BasicMat8 in the materials folder. Then go to: Edit > Render Pipeline > Universal Render Pipeline > Upgraded Selected Materials to UniversalRP Materials.


Clear All PlayerPrefs
Render Pipeline
Graphics Tier
Grid and Snap Settings...
Generate Shader Includes
Universal Render Pipeline
Upgrade Project Materials to UniversalRP Materials
Upgrade Selected Materials to UniversalRP Materials
2D Renderer
Assets
EveldTech
DynamicCamera
DemoAssets
DynamicCamera > DemoAssets

The speed scene is for testing the tracker at high speeds of the player. The slow scene is for the slower paced rigid body tracking.

You can play the scenes and walk / bounce around and test a wide variety of camera settings from a small UI while playing. It also shows the game controls and allows you to switch between the scenes during run time. Note that the scenes must be inside the build else you can’t switch between them.

Note that the camera shaker parameters must be adjusted from the inspector.

You can test that the tracker is frame rate independent by pressing F1 to enable the low frame rate and press F1 again to restore it to default settings. Or you can find the TargetFrameRate game object in the hierarchy of both scenes and enable it in the inspector with a set target frame rate.

9.2 Demo scripts

The demo scripts that are relevant to make the dynamic camera tracker work:

- DynamicCameraControllerMinimal.cs
- DynamicCameraController.cs

Additional demo scripts are located in the DemoScripts folder.

10 Miscellaneous

A scripting reference is added in the documentation folder.

More information will be on the Unity Asset Store Page, such as tutorial videos.

A current known issue with the preview window of the effector tool is that the preview is darkish with the universal render pipeline for both preview camera types.

11 Contact Information

For contact, feedback and help please send an email to:

eveldtech@gmail.com