Thank you for purchasing the Motion Warping!

Here is what you need to do now:

1. Join our Discord and get a verified role [LINK HERE].
2. Pull the demo project from the GitHub repository [LINK HERE].
3. Import the asset via a Package Manager.
4. Visit the official documentation GitBook page [LINK HERE].

This document contains a quick step-by-step guide on how to use the **Motion Warping plugin** in your project.

How this asset works

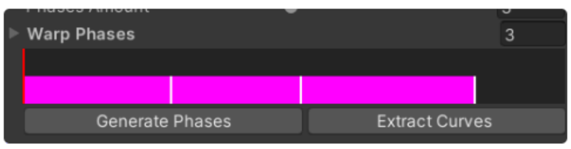
How does it warp the motion?

Our plugin uses 4 concepts in order to properly adjust character animation in realtime:

1. #warping-phases
2. #play-rate-scale
3. #total-root-motion
4. #post-animation-update

Warping Phases

Warping Phases are segments of the animation, where we want to perfrom warping. In our plugin, these are **purple** draggable windows found in the Motion Warping Asset:



Vaulting animation example.

The example above is a vaulting animation that has 3 warping windows. **But why 3?** The amount of warping windows depends on the number of the target points we are going to provide. To vault over an object, we must know:

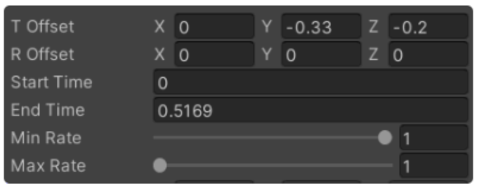
- Front obstacle edge point
- Rear obstacle edge point
- Landing point

This makes it 3 warping points, which gives us 3 animation segments.

Another example: climbing animation - we only need a single warp phase, because we want to go from the current character position to the obstacle top point.

Warping Phases also contain useful information about the animation segment, such as:

- Target point position and rotation offset
- Segment playrate controls
- Segment start and end time



T Offset X 0 Y -0.33 Z -0.2
R Offset X 0 Y 0 Z 0
Start Time 0
End Time 0.5169
Min Rate 1
Max Rate 1

T and R Offsets are used to offset the target point of the segment. This is super useful, since the system will always align the character root with the target point. You can also use this to fix awkward poses, caused by the original animation.

Min and Max playrate define the play rate limitations. And you will find out more about play rate in the next chapter.

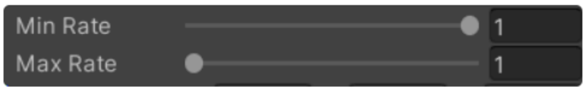
Play Rate Scale

When warping animations, we might run into a situation when the obstacle is simply too long. If we keep the play rate the same, it will look strange, as if the character has an insane momentum.

To fix this, our plugin scales the playrate for the current animation phase. This is done by computing the vector length difference.

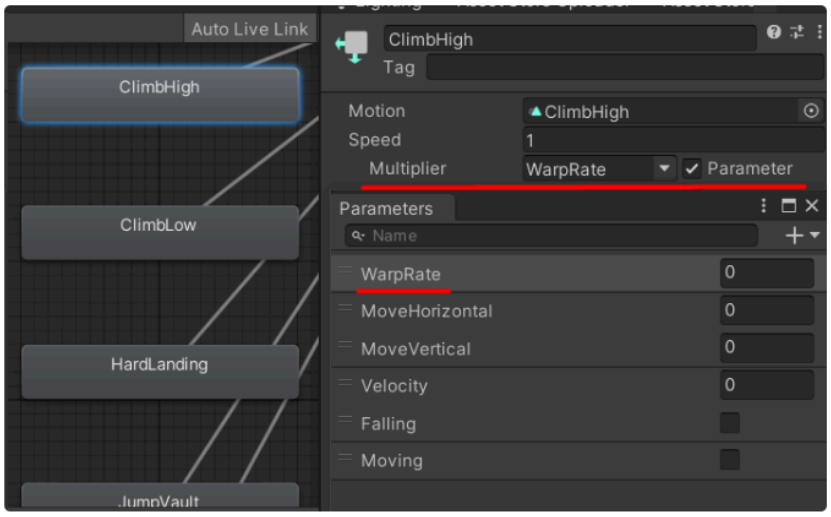
Example: if the obstacle is 1m away, and our animation was made for a 0.5m obstacle, the play rate will be scaled down twice: from 1 to 0.5. If it was 0.25m away, the play rate will be increased from 1 to 2.

You can adjust the play rate for any segment manually by editing the respective values:



Min Rate 1
Max Rate 1

The play rate is finally applied in the **Animator Controller** via a float parameter:



ClimbHigh
Tag

Motion ClimbHigh
Speed 1
Multiplier WarpRate ✓ Parameter

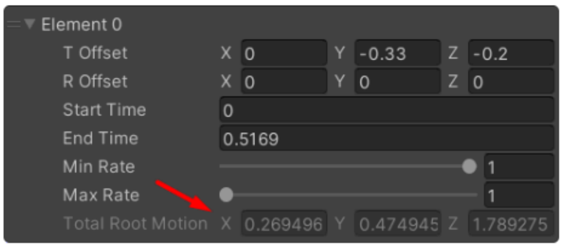
Parameters

WarpRate	0
MoveHorizontal	0
MoveVertical	0
Velocity	0
Falling	☐
Moving	☐

If you have a custom animation system, you will need to find a way to manually plug in the adjusted play rate value.

Total Root Motion

Every **Warping Phase** contains accumulated root motion for each translation axis.



Element 0

T Offset	X 0	Y -0.33	Z -0.2
R Offset	X 0	Y 0	Z 0
Start Time	0		
End Time	0.5169		
Min Rate	1		
Max Rate	1		
Total Root Motion	X 0.269496	Y 0.474945	Z 1.789275

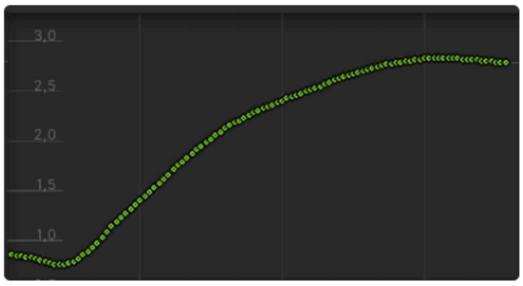
✓ **Tip:** the total root motion is refreshed when you re-open the motion warping asset.

But why is it important? This feature is used to preserve the original motion. This is crucial for proper warping, and you will find out more in the next chapter.

Post-Animation Update

Animations are warped in the `LateUpdate()`, right after the Animator update. This allows to completely take over the character root motion, and preserve the previously applied IK and constraints.

In order to warp the motion, we must know the difference between original animation and the desired target point. Let's use a climbing animation curve as an example:



Climbing animation curve

This curve offsets the character vertical position by ~3 meters. The Total Root Motion will be ~3, as it's the final offset.

In runtime, when our animation is playing, the system will acumulate animation translation deltas, and divide them by the **Total Root Motion.**

In runtime, when our animation is playing, the system will acumulate animation translation deltas, and divide them by the **Total Root Motion.**

✓ **Example:** let's say our animation was made for climbing a 3 meter obstacle. The object we want to climb is 6 meters, and we desire to warp. The height delta is 3 meters, so we need to add it to the current root motion, using total root motion:

Formula: offset = desiredHeightDelta * accumulatedRootMotion / totalRootMotion,

where **(accumulatedRootMotion / totalRootMotion)** is in range [0;1] and acts as an alpha or weight.

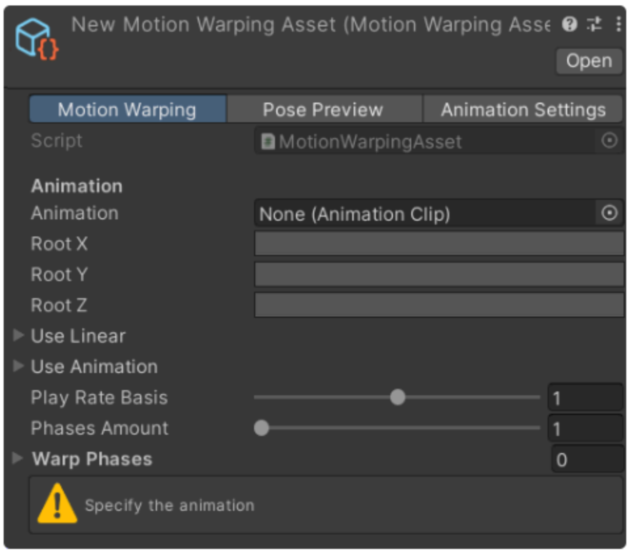
After that, **Motion Warping** will apply the offset to the current position.

Note: rotation warping is slightly different. Instead of using Total Root Motion, rotations are simply SLERP'ed based on the current Warp Phase normalized time:

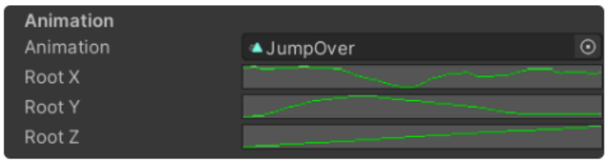
rotation = Quaternion.Slerp(startRotation, targetRotation, normalizedTime).

Step 1: Create a Motion Warping Asset

First, we need to create a data asset, which will configure the warping for our animation. To create a new asset: **Right click → Create → MotionWarping → MotionWarpingAsset.**


New Motion Warping Asset (Motion Warping Asse
Open
Motion Warping | Pose Preview | Animation Settings
Script MotionWarpingAsset
Animation
Animation None (Animation Clip)
Root X
Root Y
Root Z
Use Linear
Use Animation
Play Rate Basis 1
Phases Amount 1
Warp Phases 0
Specify the animation

Now you need to select your animation. Let's use **JumpOver** animation as an example. Once you've chosen your clip, hit the **Extract Curves** button.

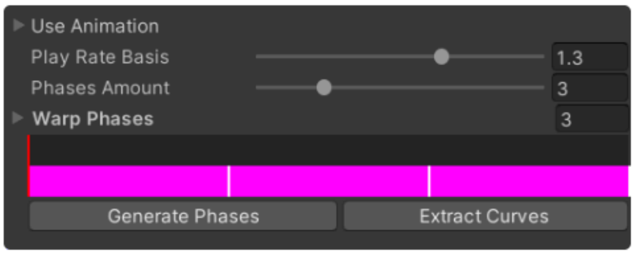

Animation
Animation JumpOver
Root X
Root Y
Root Z

Curves are properly extrcated!

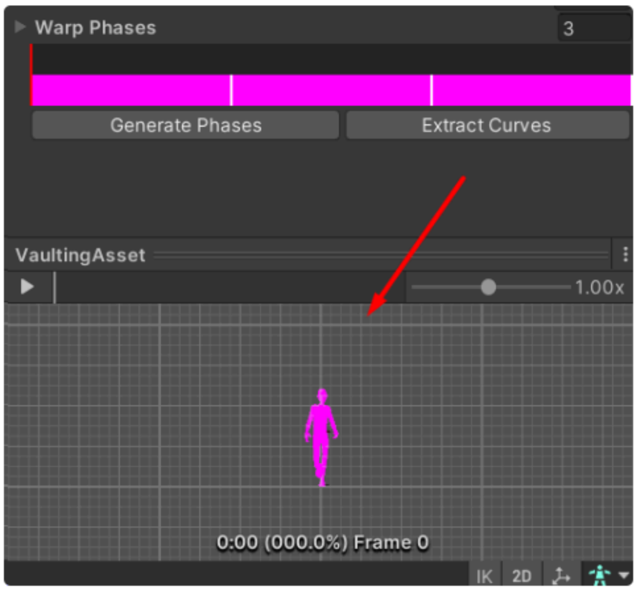
Now it's time to mark parts of the animation we want to warp. Specify the amount Phases Amount - this value depends on desired target points.

> **Example: JumpOver** animation this value is 3, because we have 3 target points: close edge, far edge and landing point.

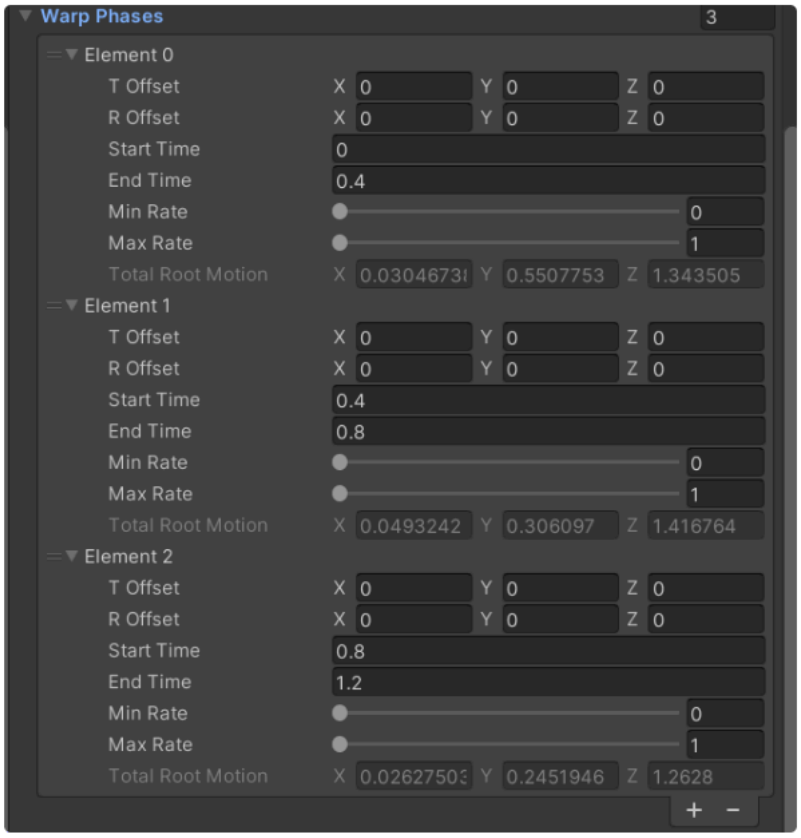
After that, hit the **Generate Phases** button:


Use Animation
Play Rate Basis 1.3
Phases Amount 3
Warp Phases 3

These 3 purple segments mark parts of the animation, which will be used for warping. You can resize and drag these areas. Also, you can preview the animation right in the editor:


Warp Phases 3
VaultingAsset
1.00x
0:00 (000.0%) Frame 0

If you open up the Warp Phases list you will see the details for each phase:



Warp Phases 3

Element 0

T Offset	X	0	Y	0	Z	0
R Offset	X	0	Y	0	Z	0
Start Time		0				
End Time		0.4				
Min Rate		0				
Max Rate		1				
Total Root Motion	X	0.0304673	Y	0.5507753	Z	1.343505

Element 1

T Offset	X	0	Y	0	Z	0
R Offset	X	0	Y	0	Z	0
Start Time		0.4				
End Time		0.8				
Min Rate		0				
Max Rate		1				
Total Root Motion	X	0.0493242	Y	0.306097	Z	1.416764

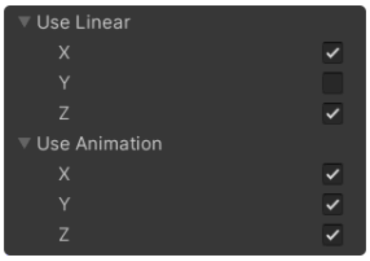
Element 2

T Offset	X	0	Y	0	Z	0
R Offset	X	0	Y	0	Z	0
Start Time		0.8				
End Time		1.2				
Min Rate		0				
Max Rate		1				
Total Root Motion	X	0.0262750	Y	0.2451946	Z	1.2628

- T Offset: translation offset for the target point.
- R Offset: rotation offset for the target point.
- Start Time: animation start time for this segment.
- End Time: animation end time for this segment.
- Min Rate: minimum allowed playrate.
- Max Rate: maximum allowed playrate.
- Total Root Motion: defines the distance along each axis.

> **Tip:** make sure to close and open up the asset again to generate the total root motion!

The last properties we have left are:



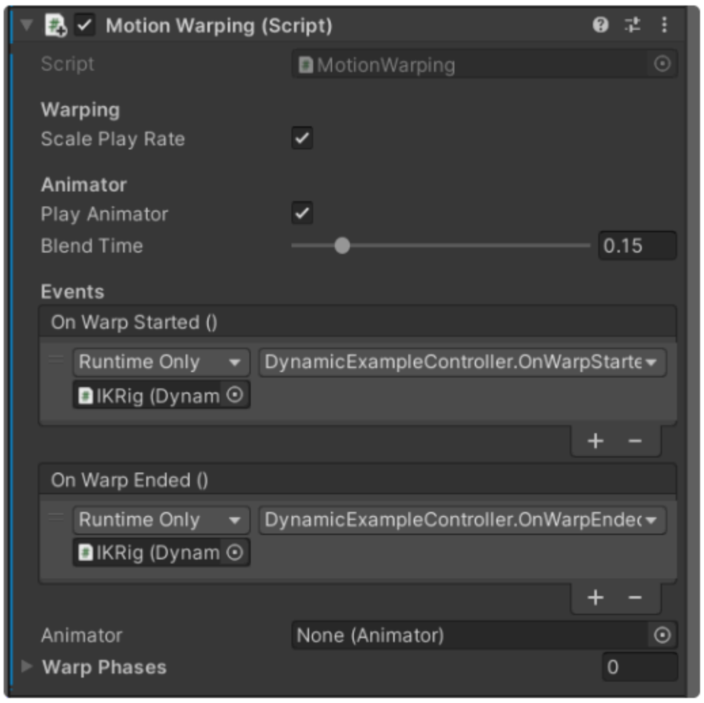
Use Linear	
X	✓
Y	
Z	✓
Use Animation	
X	✓
Y	✓
Z	✓

- **Use Linear:** defines if the will use the linear approach. Set it to true for X.
- **Use Animation:** defines if the system will use original root motion. Useful for Baked Into Pose animations.

At this point, our Motion Warping Asset is ready. Now let's move on to our character.

Step 2: Add Motion Warping Component

Add **Motion Warping** to your character:



Motion Warping (Script)

Script: MotionWarping

Warping

Scale Play Rate: ✓

Animator

Play Animator: ✓

Blend Time: 0.15

Events

On Warp Started ()

Runtime Only | DynamicExampleController.OnWarpStarte

IKRig (Dynam

On Warp Ended ()

Runtime Only | DynamicExampleController.OnWarpEndec

IKRig (Dynam

Animator: None (Animator)

Warp Phases: 0

- Scale Play Rate: whether we should modify original play rate.
- Play Animator: whether we should automatically play the animation.
- Blend Time: cross-fade blend in time for our animation.
- OnWarpStarted: called right before the interaction. Use it to disable collisions and other systems.
- OnWarpEnded: called right after the interaction. Use it to enable the collision or movement back.

Make sure to add a reference to the component in your code:

```
public class YourController : MonoBehaviour
{
        //...
        private MotionWarping _warpingComponent;
        //...

        private void Start()
        {
                //...
                _warpingComponent = GetComponent<MotionWarping>();
                //...
        }
}
```

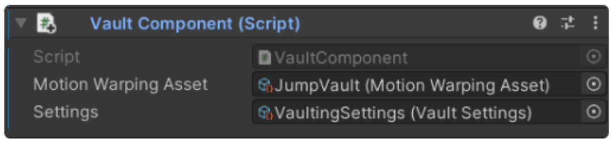
We will use it to check if it's possible to perform an interaction. Now let's move to the **Warp Provider** section.

Step 3: Add a Warp Provider

Warp Providers are components, which implement **IWarpProvider** interface. Their role is quite important in the system: they analyze the environment by runtime tracing, and then provide desired target points for our character.

For example, **Vault Component** will trace forward to find an obstacle, and if it's vaultable, we can perform our vaulting action.

Let's add a **Vault Component** to our character:



Vault Component (Script)

Script	VaultComponent
Motion Warping Asset	JumpVault (Motion Warping Asset)
Settings	VaultingSettings (Vault Settings)

Make sure to specify the asset we created in **Motion Warping Asset** and vaulting settings. You can use the ones right from the demo project.

Now we need to actually interact with the **Vault Component** in the code.

```
public class YourController : MonoBehaviour
{
        //...
        private MotionWarping _warpingComponent;
        //...

        //...
        private VaultComponent _vaultComponent;
        //...

        private void Start()
        {
                //...
                _warpingComponent = GetComponent<MotionWarping>();
                //...
        }

        private void Update()
        {
                if (Input.GetKeyDown(KeyCode.F))
                {
                        _warpingComponent.Interact(_vaultComponent);
                }
        }
}
```

`Interact` method returns `WarpInteractionResult` struct:

```
public struct WarpInteractionResult
{
        public WarpPoint[] Points; // target points in world space.
        public MotionWarpingAsset Asset; // animation data asset.
        public bool Success; // whether attempt was a success.
}
```

If you want to implement a custom Warp Provider, simlpy implement **IWarpPointProvider** interface in your MonoBehaviour.