SUPPORTS MIRROR

by: twoloop | contact: twoloopgames@gmail.com

OriginShift Documentation & Guide

Table of Contents:

Getting Started

The most important code exists in **OriginShift.cs**.

The **OriginShift** class is a singleton component that recenters the client's world when their `focus` Transform surpasses a distance threshold.

Put the **OriginShift** component on a root GameObject in your gameplay scene.

Now, assign the `focus` Transform for the world to recenter.

The `focus` Transform is usually the player, but it could also be the camera depending on how you set up your hierarchy.

Make sure your focus is a root transform (at the top of the hierarchy with no parents).

WHAT IS PRECISION MODE?

Precision mode determines the data type we use to store each client's world offset. If a client travels a great distance from world origin, they are likely to have an offset with a high value, which is what keeps track of much of their position vector. This offset data is useful for synchronizing positions over the network.

HOW TO DETERMINE PRECISION MODE:

If your game is singleplayer, set the precision mode to Float. Otherwise, use the lowest precision your that supports the distance your player can travel:

Float	1000 km
Double	0.1 ly
Decimal	1 trillion ly

OPTIMAL TICK DELAY

Ideally, OriginShift should only be checking if the distance threshold is passed when it is possible for your player to have traveled that distance. For this reason you can use this formula:

```
Optimal Tick Delay = distanceThreshold / maxSpeed (in meters/second)
```

Configuring for Multiplayer

SETUP

First, put a **NetworkOS** component on a gameobject in your scene. This stores the host's offset and is used for correcting spawn positions.

Mainly, we provide you with **OSNetRigidbody.cs**. **OSNetTransform.cs, and OSNetTransformV2.cs** (last one includes snapshot interpolation).

These components will keep your positions, velocities, and rotations in sync for all clients.

If you plan to use these components, you will need to install Mirror 46.0.4 or higher (older versions may work too).

If you aren't using Mirror, you can easily write your own using your networking library if you follow our well-commented examples.

You'll also want to place a **OSStartAdjuster.cs** component on prefabs you want to spawn in that don't need to move.

IMPORTANT CONSIDERATIONS

Keep in mind that when using this origin shifting solution, your clients' worlds are all in different places so that each of them can be near (0,0,0) on their own machine.

Why is this important?

Because now whenever you want to communicate a position over the network, you will have to transform the remote position to the local client's world space offset.

To do this, use:
`Vector3 localPosition = OriginShift.RemoteToLocal(remoteOffset, remotePosition)`

Where

- `remoteOffset` is the remote client's `OriginShift.LocalOffset`
- `remotePosition` is the remote client object's `transform.position`

GPU Instancing

By default, OriginShift is compatible with GPU Instancer.

Just make sure that you enable GPUI's origin shift bool and plug in the OriginShift as the transform on your **GPUIPrefabManager.**

You may instead want to access GPUI API to manually offset the instances.

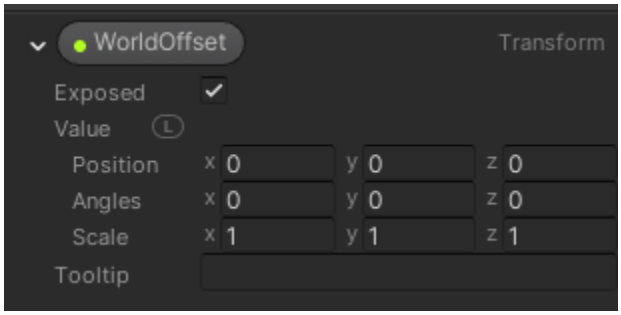
If you have special concerns or questions about GPU Instancing and OriginShift integration please feel free to reach out and we'll do our best to help you.

VFX Graph

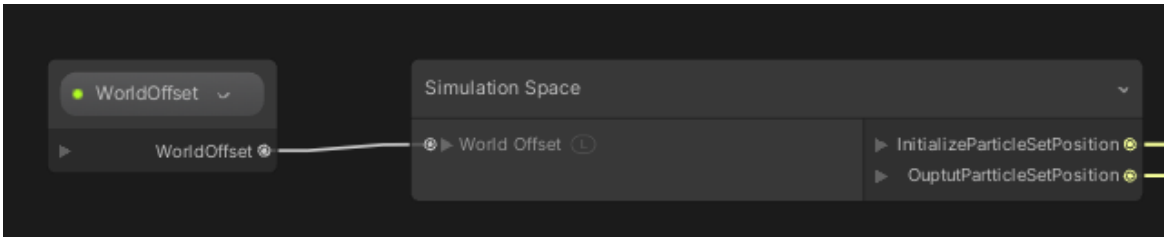
For VFX graphs, **world space** particles should use the provided SimulationSpace operator.

To set up a custom simulation space in your VFX graph you need to follow these steps:

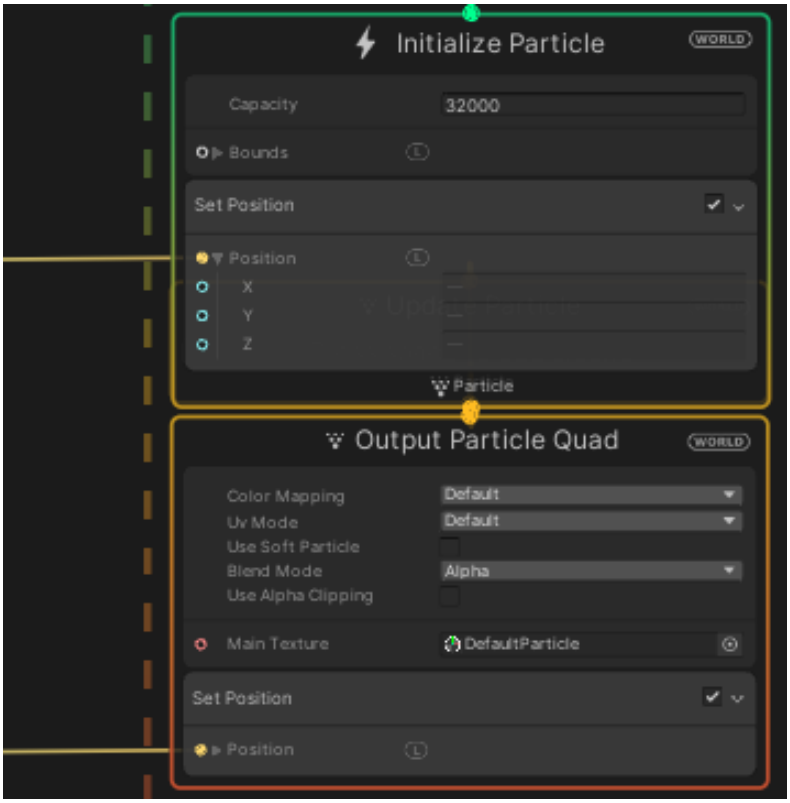
1. Create an exposed Transform parameter named “`WorldOffset`” with Value: L/Local



2. Create Node -> SimulationSpace, and plug in `WorldOffset`



3. Create SetPosition blocks in InitializeParticle and OutputParticle. Connect SimulationSpace outputs accordingly. You may need to change the “L” in the InitializeParticle SetPosition block to a “W” by clicking on it.



Disclaimers

- OriginShift currently does not support static objects. Please uncheck static on these objects
- If you have world space height fog you will likely not want to use `verticalRecentering` as the fog will not rely on the Origin Shift
- If you use shaders that depend on world coordinates (triplanar, vegetation wind etc) you may want to use local/object space coordinates in that shader instead because when the client recenters you may notice the shader values change for a very short moment
- This is not a solution for single meshes that are insanely large. (ex. a 100000 x 100000 plane or a giant single planet mesh).
- If you use the function Transform.Move() to move your player, OriginShift will not work unless you use continuous mode