Floating Origin System

This document will teach you how to use the floating origin system in Space Graphics Toolkit (SGT).

This system is fairly easy to use, but it's also easy to make mistakes and have your game behave incorrectly, so make sure you understand what each component does before using them.

Basic Camera Setup

First, you need to add the **SgtFloatingPoint** component to your camera GameObject. This component stores the high precision position of your camera, because the normal Transform.position value isn't accurate enough for a full universe.

Next, you need to add the **SgtFloatingOrigin** component to your camera GameObject. This component lets SGT know that the above **SgtFloatingPoint** is the main one, and will be used for all camera calculations.

Next, you need to add the **SgtFloatingCamera** component to your camera GameObject. This component will automatically position the camera based on any changes to the above **SgtFloatingPoint**, and will also automatically snap the camera position back to 0,0,0 if it moves too far away.

NOTE: This snapping causes the camera's Transform.position to change, which may cause unusual behaviour with certain camera control components. To fix them, you must hook into the **LeanFloatingCamera.OnPositionChanged** event, and update the required values.

NOTE: This snapping only applies to the camera. If you want normal GameObjects to snap too, then follow the next step.

Basic Object Setup

First, you need to add the **SgtFloatingPoint** component to your GameObject. This component stores the high precision position of your object, because the normal Transform.position value isn't accurate enough for a full universe.

Next, you need to add the **SgtFloatingObject** component. This component will automatically position the object based on any changes to the above **SgtFloatingPoint**, and also allows you to use all of the more advanced floating origin system component.

Your object should now snap correctly with the camera snapping.

NOTE: If your object moves (e.g. Rigidbody physics), then you should to enable the **Monitor Position** setting in the **SgtFloatingObject** component. This will automatically detect position changes to the **Transform.position**, and apply them to the **SgtFloatingPoint**.

LOD Setup

Space is very large, and it's easy to have objects millions of kilometers away and no longer visible, but even if they're so far away that you can't see them, they're still wasting CPU and GPU resources.

To fix this, the **SgtFloatingLod** component can be attached to the **SgtFloatingObject** GameObject. This component will automatically spawn and despawn the **Prefab** when it's within **DistanceMin** and **DistanceMax** of the **SgtFloatingOrigin**.

NOTE: LOD prefabs should NOT have the **SgtFloatingObject**.**Point** field set, because they will automatically be assigned/inherit the one from the **SgtFloatingLod** GameObject that spawned them.

Spawner Setup

Spawners allow you to create a hierarchy of procedurally spawning objects. For example, when you approach a star you may want to procedurally spawn planets in orbit.

To do this, I first recommend you create an **SgtFloatingLod** on your planet for the spawner, and associated prefab. This gives you to control over when your planets will spawn, and will improve performance.

Next, you need to add the **SgtFloatingSpawnerOrbit** component to the **SgtFloatingObject** you want them to spawn around (e.g. LOD prefab). This component will automatically spawn prefabs in orbit around the attached **SgtFloatingPoint** (or parent point if you spawned from **SgtFloatingLod**).

Next, you need to specify the **SgtFloatingObject** prefabs the spawner can spawn.

NOTE: These spawning prefabs should have the **SgtFloatingPoint** component attached.