Overview and Intent

The orbit and gravity plugin provides an easy way to create gravity sandboxes and do orbital mechanics calculations in Unreal Engine. The plugin lets you quickly give actors “gravity” (through GravityMass components) that apply forces to other actors (through OrbitMovement components). Gravity is calculated through Newton’s Law of Gravitation, with a configurable constant that allows you to set the size of the gravity effect. The plugin then calculates a realistic orbit for actors, lets you obtain orbital parameters, and provides functions that do orbital mechanical calculations you may need.

Video Overview: https://www.youtube.com/watch?v=jzugf55ldrw

System Requirements

Orbit and Gravity Plugin will run on any Windows PC capable of running Unreal Engine 4. Other platforms have not been tested.

User Guide

BP_GravityMass component

Add this component to an actor to give that actor gravity. Commonly used configuration options are:

- Will Enable Gravity - Enable to have the component have an effect.
- Standard Gravitational Parameter - Increase to have the GravityMass have a larger gravitational field. Gravitational fields follow the inverse square law.
- Min Safe Distance - Used for “IsOrbitSafe”, the minimum safe distance from the center of the actor.
- Use Closest Point - If enabled, the distance used in the gravitational computation is the closest point from this actor to the other actor. If disabled, the center point of the GravityMass actor is used.
- Anti-Gravity - Reverses the direction of gravity. If used, all orbital parameters will be invalid.

BP_OrbitMovement component

The BP_OrbitMovement component will apply the force generated by GravityMass components. **Make sure your actor is set to Moveable!** Commonly used configuration options are:

- Real Time - if enabled, the timestep used to update orbit positions will be the DeltaTime, or framerate. If disabled, the “Simulated Time” is used. You can set a low simulated time to increase the accuracy of kinematic physics mode actors.

- Additional World Accel - Additional acceleration to apply to actors, for example, if you also want world gravity. (Use <0,0,-980> for typical earth gravity)
- Sub Step Kinematic Physics - Use multiple physics calculations per frame (for kinematic physics only)
- Start circularized - Start the actor with a circularized orbit based on the BP_GravityMass component that is having the largest effect on the actor
- Direction to start - A vector describing the direction the actor should start its orbit
- Draw Debug trials - Debugging trails for actors
- Start Velocity - An initial velocity for the orbit. This, along with the position, define the orbit. Specify in world space unless “Start Velocity In Local Space” is true
- GravityMassArray - An array that specifies which GravityMass components the Movement component should use. If left blank, will default to every GravityMass in the scene. Only used with kinematic physics mode.
- GravityMassToOrbit - Specifies which GravityMass to Orbit. Only used in kepler mode. If blank, will default to the largest influence.
- Use Orbit Time For Start & Orbit Time for Start - If this is set, kepler physics mode will start the actor X seconds into the orbit, instead of starting where the actor is currently placed.
- Draw Orbit On Begin Play - An example to show how to draw a spline around the orbit.

Orbital parameters and functions

Orbital Parameters are available in the “BP_Orbit” class inside the BP_OrbitMovement component. For example, to access an orbiting actors true anomaly:

1. Get a reference to the BP_OrbitMovement component for the actor.
2. Drag a pin off the reference from step one and select “Get Orbit”
3. Drag a pin off the orbit reference from step 2 and select the “Get True Anomaly” function.

Technical Details

Accuracy

The orbit and gravity plugin uses “float” variable types for calculations, so accuracy will be aligned to floating point precision. The physics type used to calculate orbits will also affect accuracy:

- Kinematic: This method loops each BP_OrbitMovement component through each GravityMass component, and updates the position and velocity of each BP_OrbitMovement based on the DeltaTime of the last update. Therefore, the faster the DeltaTime (or framerate), the more precise that orbit position will be. Substepping can be used to improve accuracy by doing multiple physics iterations for each frame, or, if realtime simulation is not required, you can set an artificial low time interval for each calculation.

- Kepler: This method generates a mathematically precise orbit location based on where the orbiting body ‘should’ be at a given time. However, because Kepler’s equations are probably ‘unsolvable,’ numerical approximation is required. Therefore, each frame may have slight errors, but these errors will not compound over time like kinematically calculated orbits.

- Blended: Uses kinematic physics for an estimated position, then kepler physics for a precise distance for that location. Slightly more efficient than kepler mode.

Computational Efficiency
Many factors will affect performance, such as how many GravityMass and OrbitComponents are in the scene, and which physics mode you select.

To help with computational efficiency, the orbit plugin will only calculate orbital parameters once per frame. For example, if you use an OrbitMovement component’s GetTrueAnomaly() function, there will be a calculation required. If then, during the same frame, you call GetTrueAnomaly() again, the plugin will remember the value has already been calculated, and provide that with very little performance cost.

Collision
To implement actions that occur after a collision, implement the HandleAHit() class in BP_OrbitMovement.

Wrap Up

Thank you for using Orbit and Gravity plugin. Please ask on the unreal engine forum or contact dingtechnology@gmail.com for questions or further technical support.