HyperEngine

By CodeParade

Table of Contents

Overview

What is HyperEngine?

HyperEngine was developed to be the Backend for Hyperbolica. It is built on top of Unity and augments or completely replaces many built-in components to allow Hyperbolic and Spherical geometries to run in Unity. Specifically, HyperEngine replaces the Unity rendering, physics, and coordinate system.

Since HyperEngine is purpose-built for Hyperbolica, it is not going to be actively supported as a Unity Plugin, and this is mostly provided as a starting point and open-source package so that other non-euclidean games may be able to use pieces from it.

To familiarize yourself with the concepts explained here. I highly recommend watching the Hyperbolica devlog series from #1 to #6 to get a fuller picture of the purpose of these components.

Coordinate Systems

There are various coordinate systems used throughout the code, so they will be defined here:

- **Unit Coordinates:** These coordinates are used exclusively for tiles and objects placed in tiles. The units are the bounds of a tile from -1 to 1. Of course a point may extend past a tile but be careful not to extend too far or it will quickly get mapped to infinity. Unity's transforms are relative to Unit coordinates and so this is also the coordinate system they use.
- **Klein Coordinates:** Basically, it's the Unit coordinates but scaled to the absolute unit of curvature. So the same Unit coordinate may map to a different Klein coordinate depending on how many tiles fit around a vertex, since this will change their absolute size. Klein coordinates always have a magnitude less than 1 in hyperbolic geometry. They aren't used very much, mostly an intermediate step.
- **Poincare Coordinates:** The primary coordinate system, and the one used in GyroVectors. It is a stereographic projection of the space and the magnitude should always be less than 1 in hyperbolics space.
- **Tile Coordinates:** These are strings that represent tile locations. They get mapped into a lookup table offline with corresponding Poincare coordinates, so they generally aren't used as coordinates beyond being unique identifiers for tiles.

GyroVectors

GyroVectors are an isometry of non-euclidean space that represent an object's location and rotation in hyperbolic or spherical space. In that way, they function very similarly to Unity's Transform component. They can be composed with translations, rotations, and other GyroVectors using the "+" operator.

Scene Setup

Minimal Scene Requirements

Every scene will need a:

- Player prefab
- World Builder
- Pre-generated Tile Map
- Optionally a Map and MapCam

You will find such an example scene in: `HyperEngine/Debug/DebugWorld`
Another Unity requirement is using the new Input System, this is simply because of my own requirements, but it could be modified to use the old system if needed.

Player Prefab

This object controls the player input, movement, and camera:

- **Height:** The height of the player relative to the absolute unit.
- **Walking Speed:** The speed of the player relative to the absolute unit.
- **Jump Speed:** The speed of a jump relative to the absolute unit.
- **Start Hyper:** The starting location of the player in a level in Poincare Coordinates.
- **Map Cam:** Attach this to the MapCam object if present in the scene.
- **Main Camera:** Should attach to the Main camera of the scene.
- **Head Center:** Use when the player's camera should be positioned where the feet meet the ground instead of offset by the height. Useful when flying instead of walking.
- **Free Fly:** Ignore gravity and switch to non-gimbaled looking.
- **Ignore Colliders:** Ignore any WarpColliders in the scene.

World Builder

This is a virtual class that you will need to override for your non-euclidean world. Besides creating all the tiles for your scene, this class also sets up all the shaders and other important parameters needed to render Hyperbolic and Spherical geometry. The methods you'll need to override are:

- **MaxExpansion:** You'll need to call `HM.SetTileType(x)` here, where "x" is the number of square tiles that should fit around a vertex. For example, use 3 for spherical geometry, 4 for euclidean, and 5 or greater for hyperbolic. The value returned from MaxExpansion is the number of expansions from a seed tile, or the "radius" of your world.
- **GetTile:** Here you will map a tile's coordinate to an instantiated GameObject (don't just return a prefab). The mesh for a tile GameObject is expected to fit in coordinates from -1 to 1 in all axes. There are various helper utilities available for lookup tables and procedural placement in WorldBuilder.

In addition to these overrides, there are also some parameters that can be adjusted:

- **BoundryAO:** The brightness level to globally blend shadow maps on the tile boundary so they appear seamless.
- **Fog:** Color of global fog in the scene. Alpha channel controls the intensity of the fog.
- **FogInvDist:** Controls how far the fog should extend from the camera in inverse units.
- **Lattice3D:** Check if tiles should expand in 3D axes, otherwise only on the 2D plane.
- **DualLight:** Use cyan and magenta lighting (used only in the NEMO).

Pre-generated Tile Map

For performance reasons, tile coordinates are not computed at runtime, but are pre-generated with a lookup table offline. This allows double-precision computations and fast placement for large maps. The asset package comes with some pregenerated maps, but if you need more, you'll need to follow these steps:

1. Open `HyperEngine/Editor/TileMapper.cs` for editing.
2. Find the method `AsyncGenerateTileMaps` and comment out the maps there
3. Add `GenerateTileMap` for the tile maps you'd like to generate.
    - **Type:** The number of squares around a vertex.
    - **Lattice3D:** True of this should be a 3D lattice rather than 2D.
    - **MaxExpand:** The maximum number of expansions from the seed tile.
4. In the Unity Editor, run: HyperEngine > GenerateTileMaps
5. The new map should appear in the `HyperEngine/Resource` folder.

Map and MapCam

To use an in-game map. You'll need to add a MapCam and Map object to your scene. Link the Player component's and Map component's "mapCam" to the top-level MapCam GameObject.

Render Engine

Hyper Objects

Every GameObject in the scene that needs to exist in non-euclidean space MUST have a HyperObject component in its top-most parent. There can only be one HyperObject component per GameObject hierarchy. The HyperObject controls the root Poincare coordinate of the GameObject. Any Unity Transform to the Gameobject and its children are relative to this root.

HyperObjects have these parameters:

- **LocalGV:** The local root GyroVector of the GameObject.
- **UseTanKHeight:** Scales the object's Y-axis to be horospherical instead of flat
- **AllowCulling:** The object will disable rendering and collision detection when outside of the culling radius.
- **HighPrecision:** Use double precision coordinates instead of single precision. Only recommended for moving objects very far from the center of the map.

The HyperObject class also has some important global variables:

- **WorldGV:** The GyroVector of the main camera.
- **ShakeGV:** Set by Player component during camera shakes.
- **isShaking:** Set by Player component during camera shakes.

- **drawDistSqForward:** Radius squared in Poincare coordinates for forward render distance (what's seen from the main camera)
- **drawDistSqBackward:** Radius squared in Poincare coordinates for all other render distances (what's seen from the map camera)
- **colliderDistSqUpdate:** Radius squared in Poincare coordinates where cached collision detection with the player and nearby objects can occur.
- **camHeight:** Set by Player for the height of the camera in absolute units.

For objects that are canvases, they should use HyperCanvasObject instead of HyperObject. It functions the same way but works with CanvasRenderers.

Another exception are Forced Euclidean objects. There are objects that should appear as euclidean regardless of the rendering geometry. An example might be a VR controller visual or a menu. These objects don't require a HyperObject, but do require a EuclideObject on each renderer.

SetTextures

Every object that needs to render in non-euclidean geometry needs a SetTextures component added to each Renderer. SetTextrues is responsible for updating many shader parameters and assigning dynamic materials to maximize GPU instancing optimizations. Most objects share the same material, and it is SetTextures that uses property blocks and instanced parameters for individual objects.

The object should have one of the Materials form `HyperEngine/Materials`, usually "Object" otherwise it will not render.

SetTextrues has these parameters:

- **Texture:** The primary texture of the material (uv1).
- **AOMap:** The lightmap for the material (uv2).
- **Ambient:** A value of 1 means no diffuse shadows, 0 means fully diffuse.
- **Specular:** The intensity of specular highlights.
- **Shininess:** The shininess of specular highlights.
- **Colorize:** A color to multiply with the object's texture.
- **Noise:** Adds colored noise pattern with the intensity set by the alpha channel.
- **OverrideBoundaryAO:** If nonnegative, this value will override the value set by WorldBuilder.BoundryAO.

Physics Engine

Warp Colliders

The Unity physics engine gets disabled entirely since it is not compatible with HyperEngine's non-euclidean geometry. All Unity colliders should be removed from all GameObjects. Instead, the WarpCollider component should be used instead, simply attach it to any GameObject as long as there is a HyperObject somewhere in the object's root.

WarpColliders support various different collider types:

- **Boxes**
- **Spheres**
- **Cylinders & Capsules**
- **Triangles**
- **Infinite Planes**
- **Triangle Meshes**

These colliders should be added in the different sections of WarpCollider. They also support various handles in the Unity Editor for dragging around and placing over your objects, as long as Gizmos are turned on and the collider you want to edit is fully expanded in the inspector.

The RemoveFloorTriangles removes any triangle that is entirely at or below the Y=0 floor level. This should only be used for tiles generally, dynamic and moving objects don't usually need it.