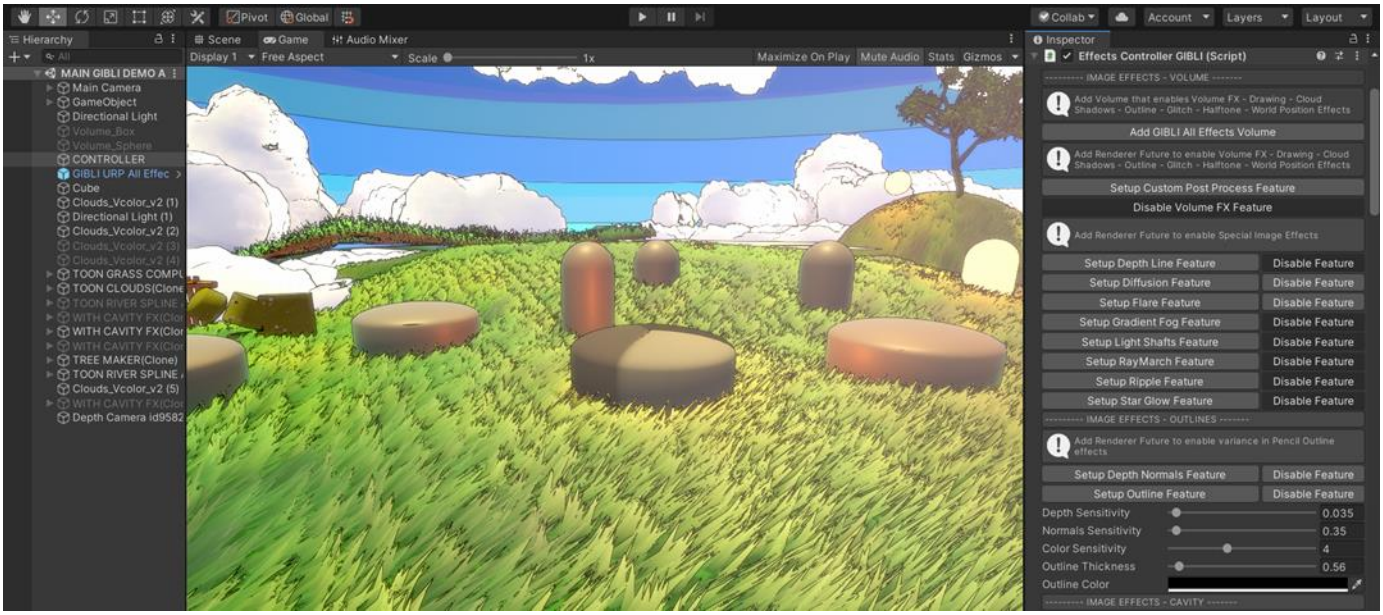
GIBLI Anime Scene Generation Framework Setup Guide (v1.2)

Overview

The GIBLI system is a general Anime – Cartoon – Non Photorealistic Rendering (RPR) scene generation framework. The system is made for the URP pipeline. For help with using the system and using it with other **ARTnGAME** assets, including **Sky Master ULTIMATE** Weather suit that Is planned to affect the shaders based on weather, please visit **ARTnGAME** **Discord channel**, **Tutorial videos** and the **GIBLI Forum** thread. **The system can be upgraded to Sky Master ULTIMATE Suit**. The system works with all URP versions and has been tested up to Unity 2021.3 LTS. **If upgrading from Unity 2019.4 to 2021.3, there is a number of patches that need to be applied, found inside the PATCHES folder**, as shown in last section of this guide, or re-download the 2021.3 version. The system requires the **Mathematics** and **Post Processing Stack** Unity Packages to be installed.

Setup steps

To setup the system, first step is to add the **"Effects Controller GIBLI"** component to an empty gameobject in the scene, and optionally name the object as **"GIBLI CONTROLLER"**. This will add the global manager that can enable the various image effects and instantiate the various sub systems like the river and grass. Note that the image effects will be applied to the first default Forward Renderer of the pipeline, so in order to edit the renderer must be setup as Default and in the first slot in the pipeline. After the setup can move to any slot in the pipeline setup in Forward Renders list, if this is desired.



Add GIBLI All Effects Volume

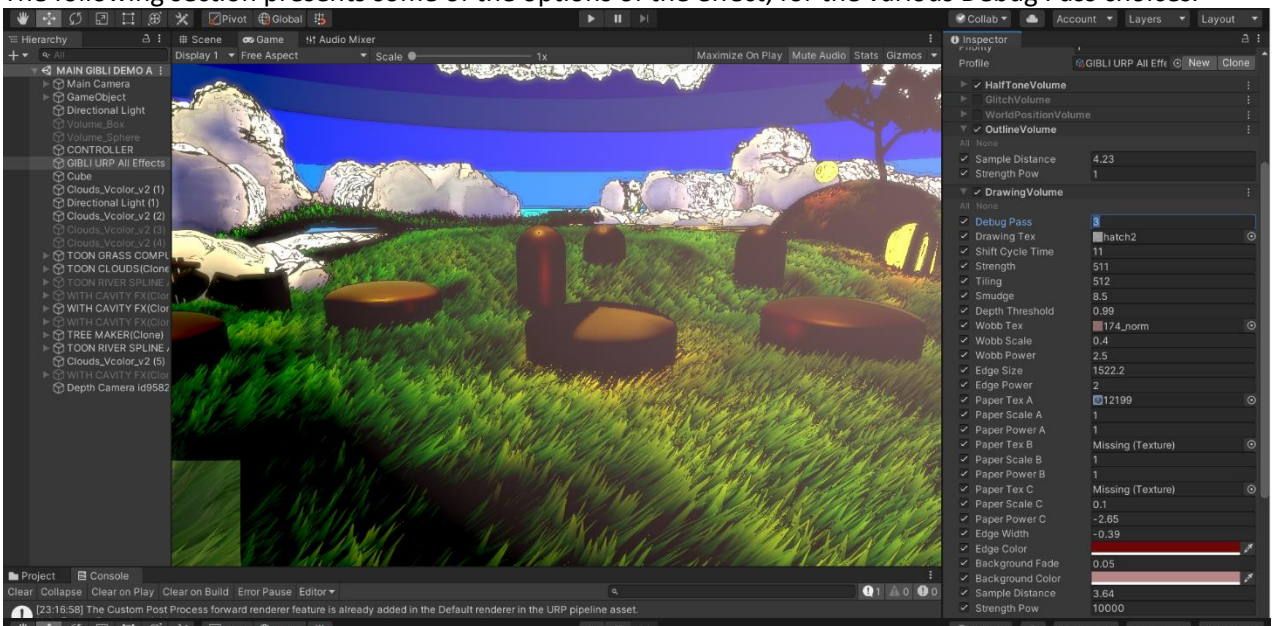
This button will instantiate a sample Volume with the volume enabled and tweaked effects in the scene, for use with the below options.

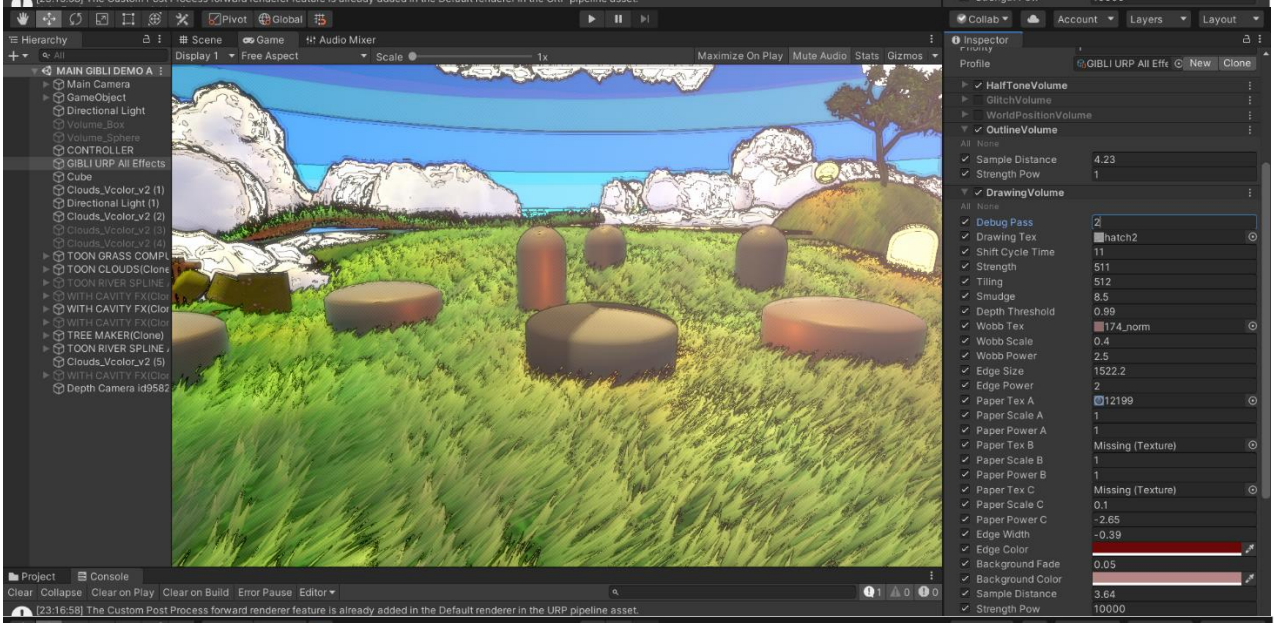
Setup Custom Post Process Feature

This adds the Custom Post Process image effect in the Forward Renderer, which is a **Volume effect** that allows various effects to be drawn as image effect on the scene, ranging from water color to outlines. The relevant script in the Volume is the **"Drawing Volume"** which has several controls and draw types (one to four, set in Debug Pass variable in the Volume).

Drawing Volume Effect (Custom Post Process Feature)

The following section presents some of the options of the effect, for the various Debug Pass choices.





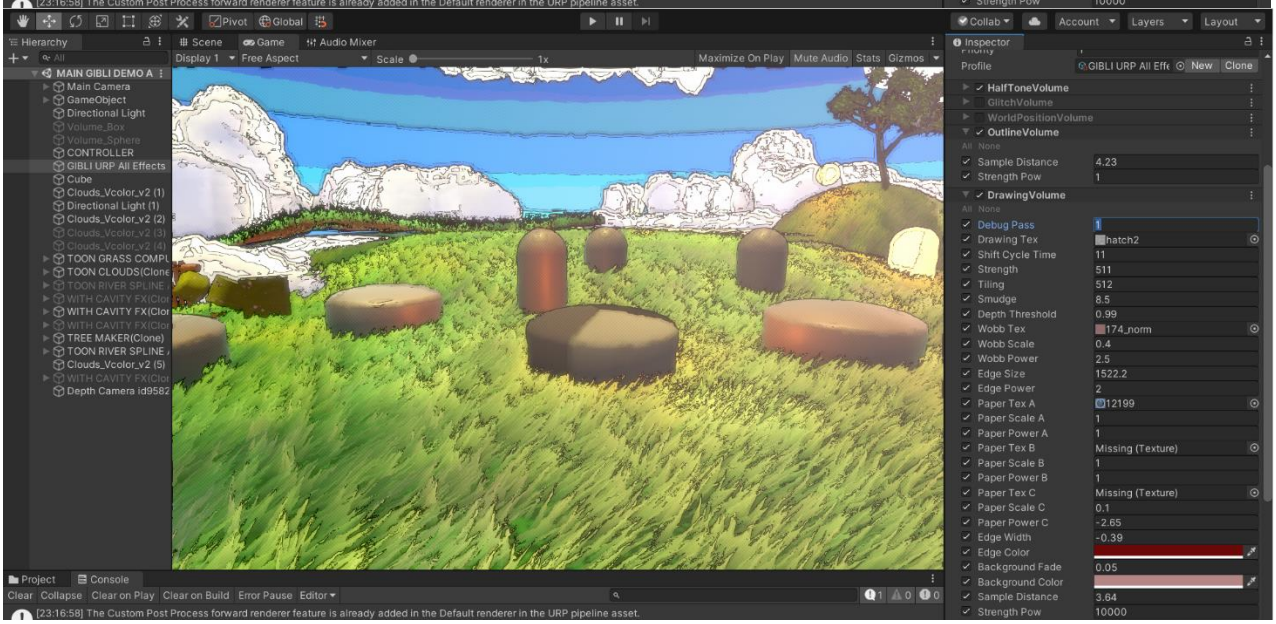
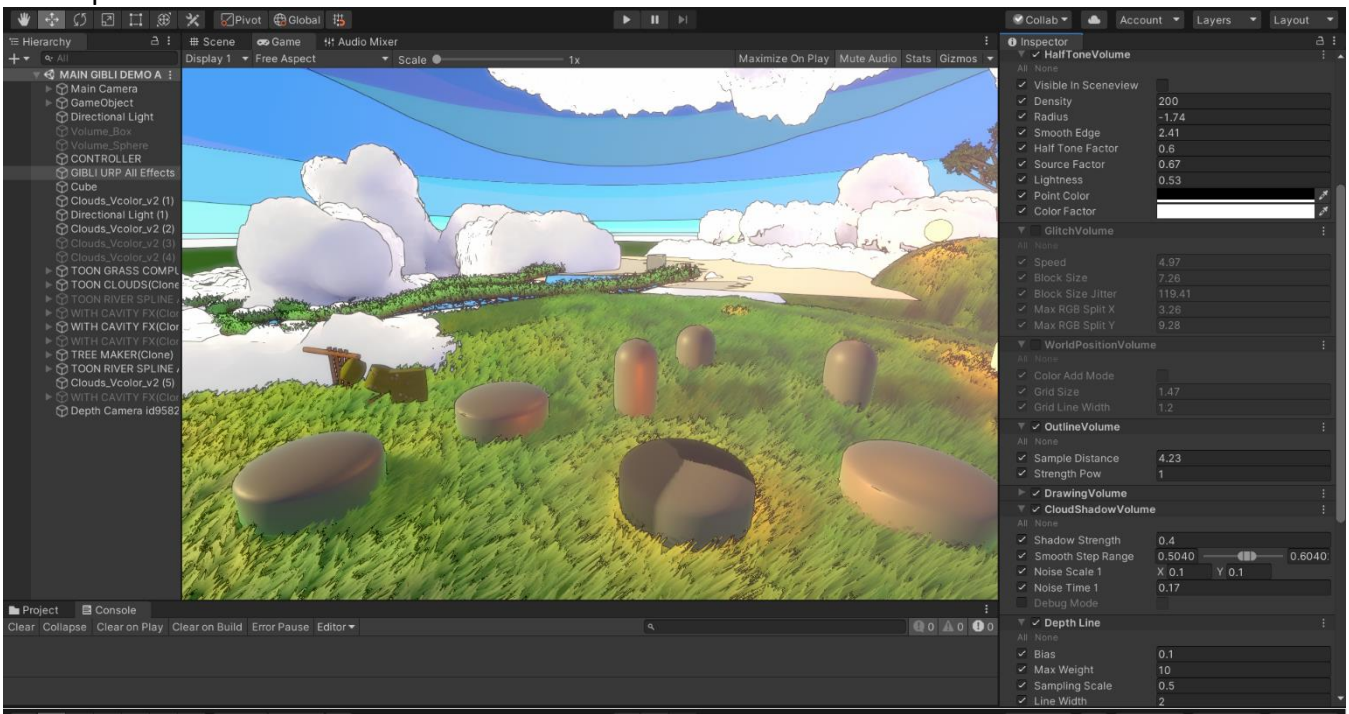


Image Effects

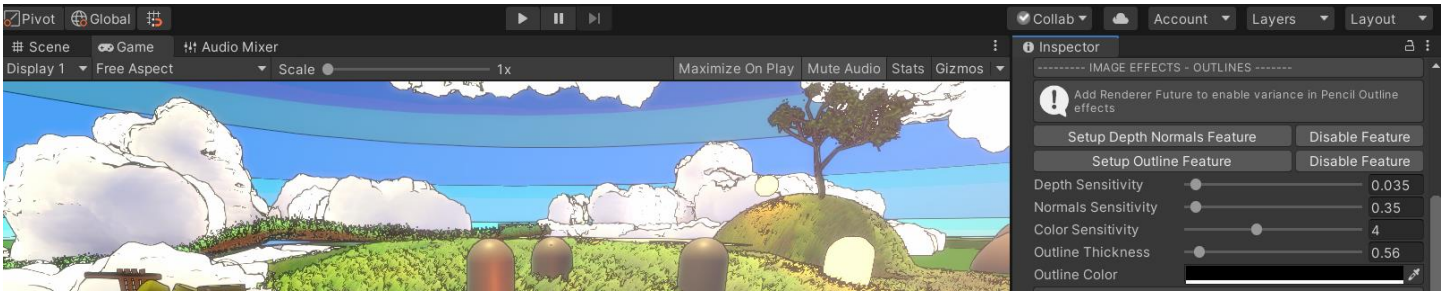
The system offers a range of image effects, enabled with the buttons from “Setup Depth Line Feature” to “Setup Star Glow Feature”. Those effects are controlled by the relevant to each Volume entry, as shown in the following section.

Volume Effects corresponding to “Setup Depth Line Feature” to “Setup Star Glow Feature” renderer features. The effects are directly tweaked by the Volume entries and saved in the volume properties asset. Some of the sample Volume effects, like the Halftone and Cloud Shadows are directly implemented in the volume and do not require a forward renderer feature.



Artistic outline Image Effect

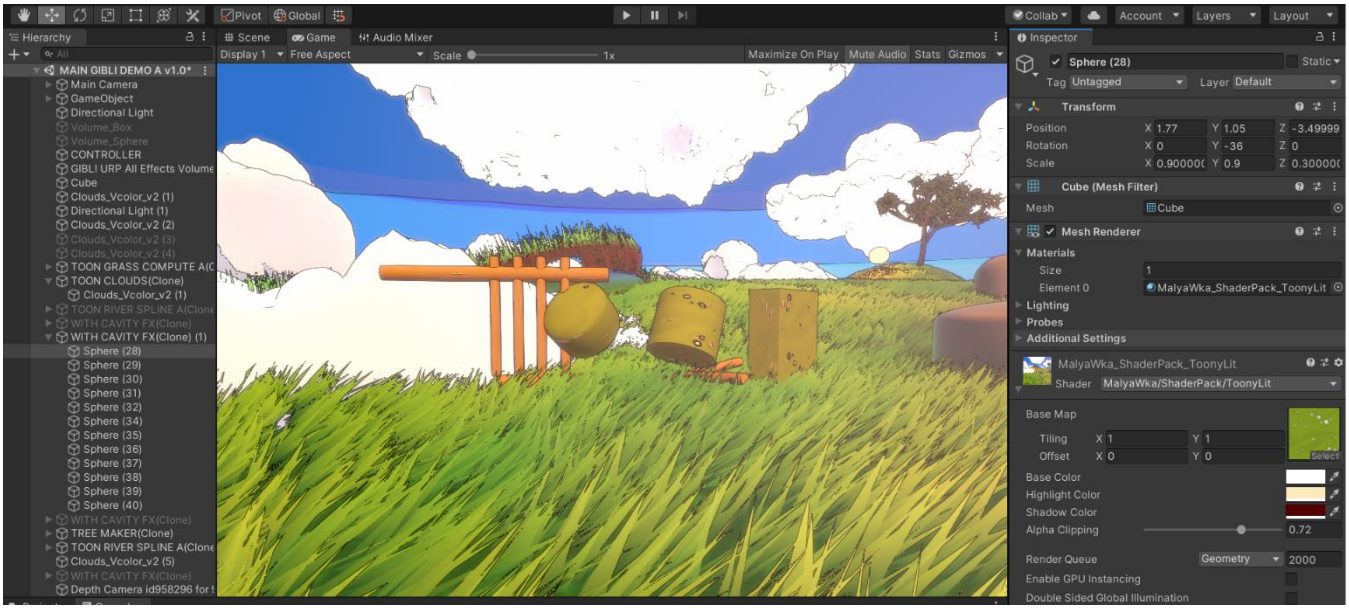
The artistic outline image effect is one of the major system features, allowing for various artistic out lining options. This effect is tweaked directly in its Outline material and the material properties can be set by the Global Manager as shown below, so each scene can apply its own settings to the material. The Outline material is also referenced in the Advanced section of the Global Manager, that shows all script variables, directly below the buttons and tweaking section.



The **Depth sensitivity** increases the effect based on scene depth, the **Normals sensitivity** based on scene objects normal and **Color sensitivity** based on object colors. The Normals option also require that the **"Setup Depth Normals Feature"** option is enabled. **Note that the Normals mode is not compatible with the Toon Clouds** because will grab the state of the cloud nesh before the Vertex animations are applied, thus is to be used for fixed geometry. The **Outline Thickness** changes the thickness of the outline.

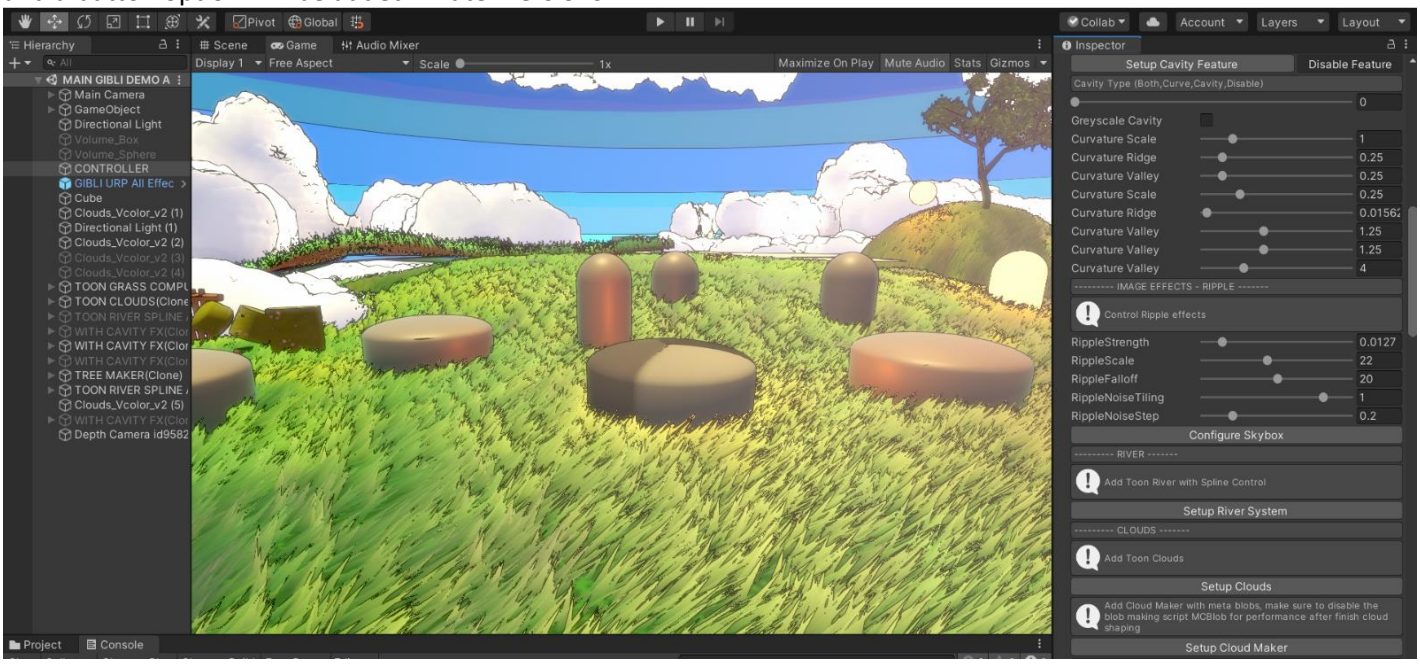
Cavity shader

The cavity shader is emulating faded outlining together with occlusion and requires the cavity shader shown below applied to the objects, plus the Cavity Forward Renderer feature assigned in the renderer in the pipeline.



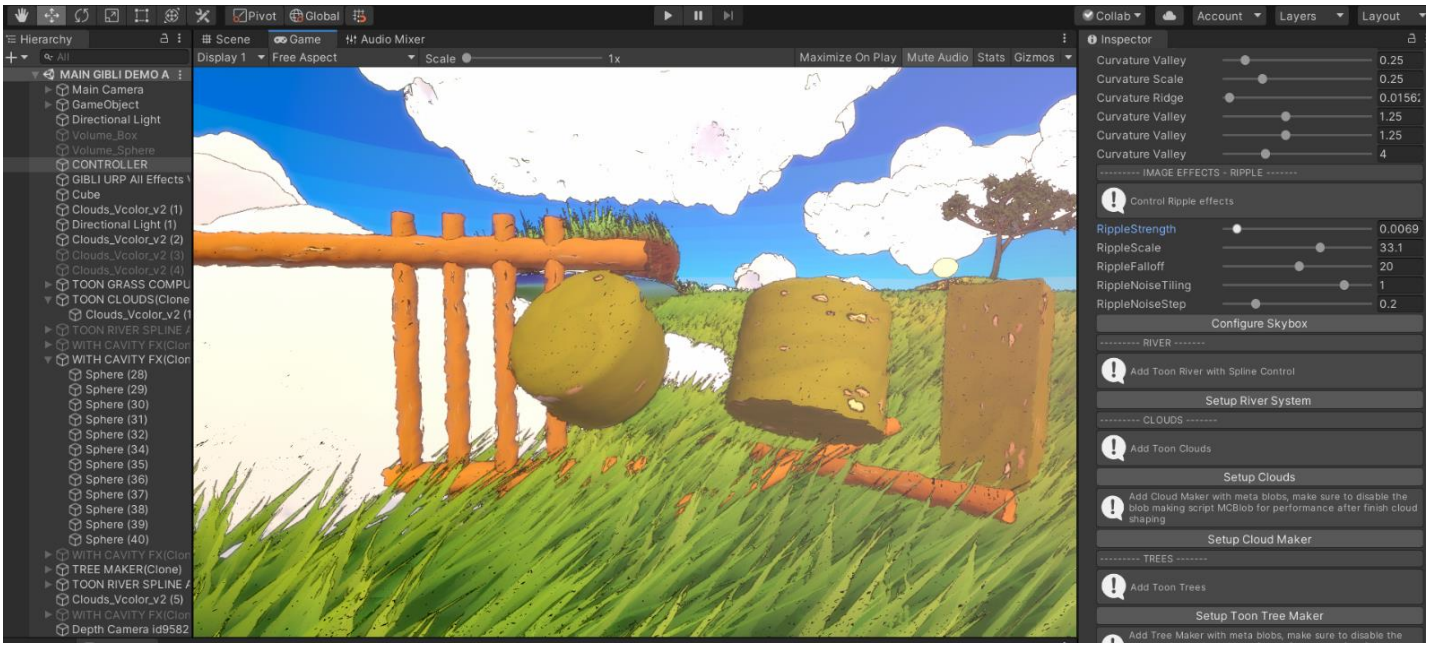
In Unity 2019 the Depth Normals feature described above must be activated, to provide the Depth Normals texture to the effect. In Unity 2021, alternatively to enabling the **Depth Normals feature**, a Unity 2021 specific version of the Cavity Forward Renderer is available, with **a version of the forward feature that enables the grab of the Depth Normals from Unity 2021 system directly.** Note **that this mode is not compatible with the Outline renderer feature described above**, as it provides a non compatible Depth Normals texture with the effect, which creates a black border in the background, so those effects are to be used separately.

The Cavity Forward Renderer feature is enabled by the "Setup Cavity Feature" button, this will activate the Unity 2019 version of the effect in the Forward Renderer. The Unity 2021 variant must be manually added to the Forward Renderer and a button option will be added in later versions.



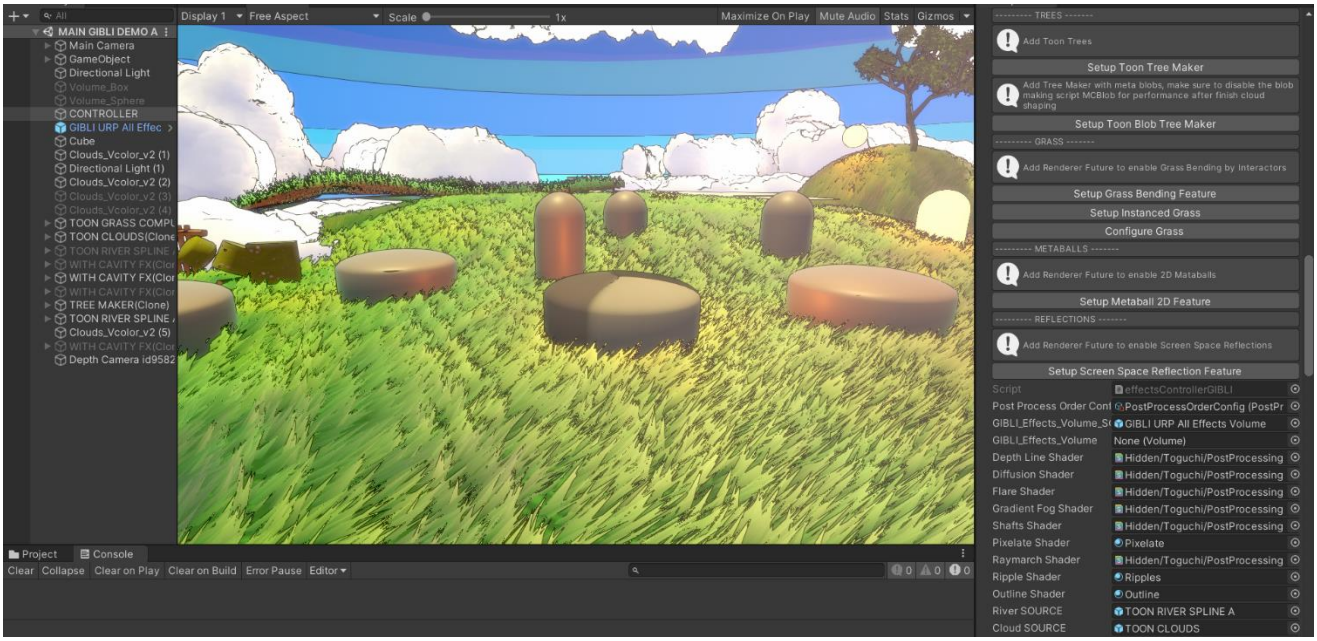
Ripple effect

The ripple effect can be applied in the screen to give a rippling affect to all objects. The system is controlled directly in the Ripple material and the tweaking of the variables can be saved through the Global Manager as shown below.



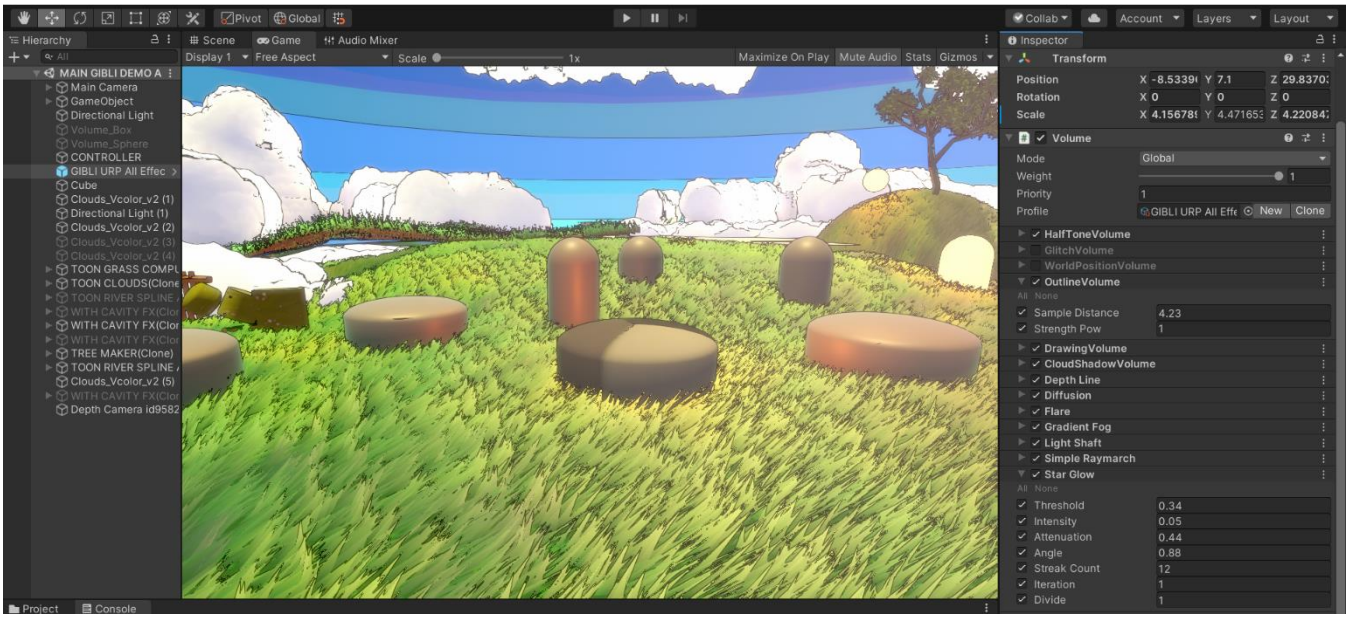
GIBLI Sub Module systems

The asset provides a wide range of sub systems, most of which can be instantiated through the Global Manager.
The **Skybox, River, Clouds, Grass and Trees sub modules can be instantiated in the scene using the relevant buttons**.
Those modules will be described in detail in the following sections.



Volume FX Setup

Several of the image effects described above require the use of a Volume in the scene, with the equivalent effect enabled. These effects are tweaked in the volume directly and there is no control variables in the global manager. A sample volume with all the effects can be directly instantiated in the scene using the “Add GIBLI all effects volume” button in the Global Manager. The Profile used in the volume is shown in the following image.

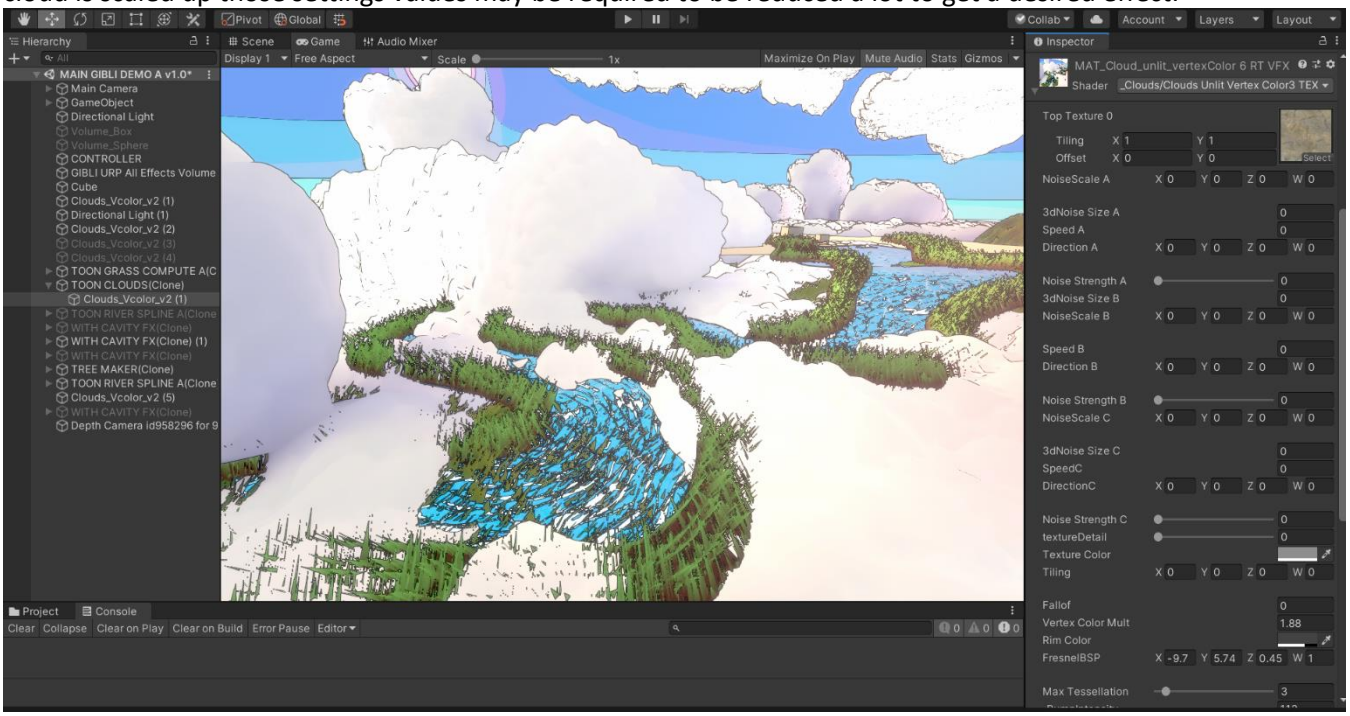


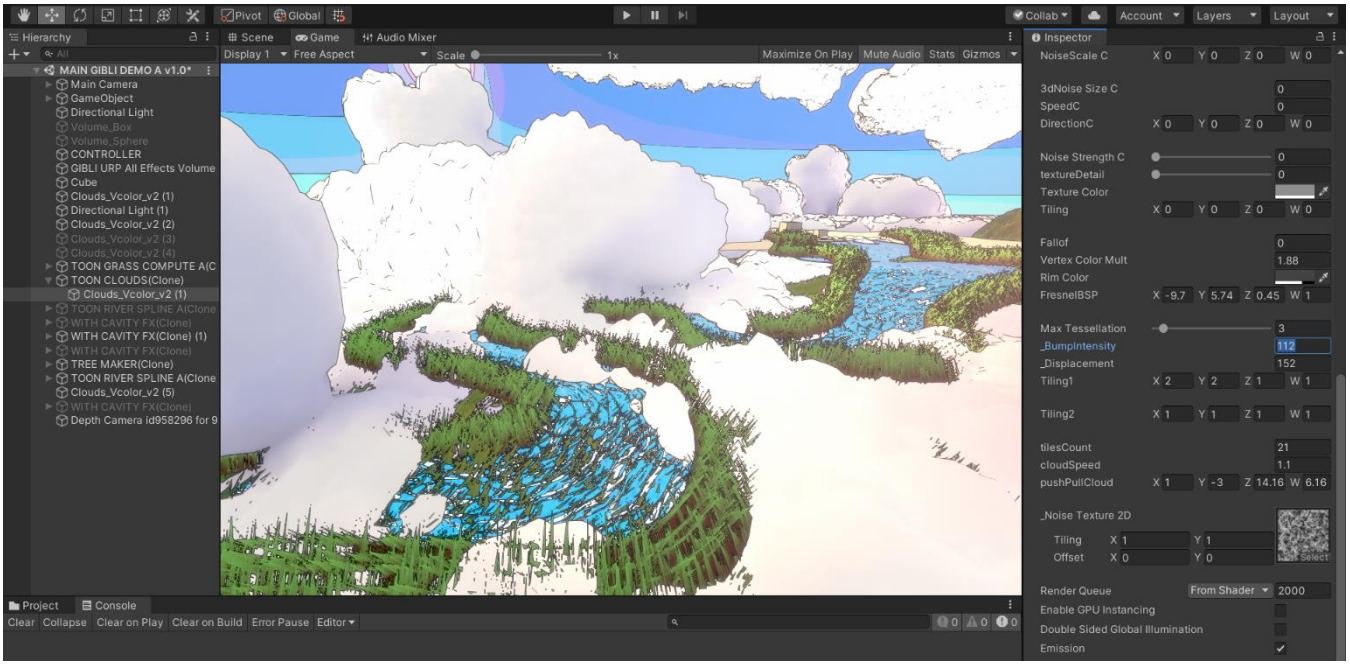
Toon Clouds Sub Module

This system contains a standard cloud mesh with a special cloud shader and a cloud mesh maker module for making custom cloud shapes. Note the clouds are not meant for fly through.

The Cloud Material is shown in the image below.

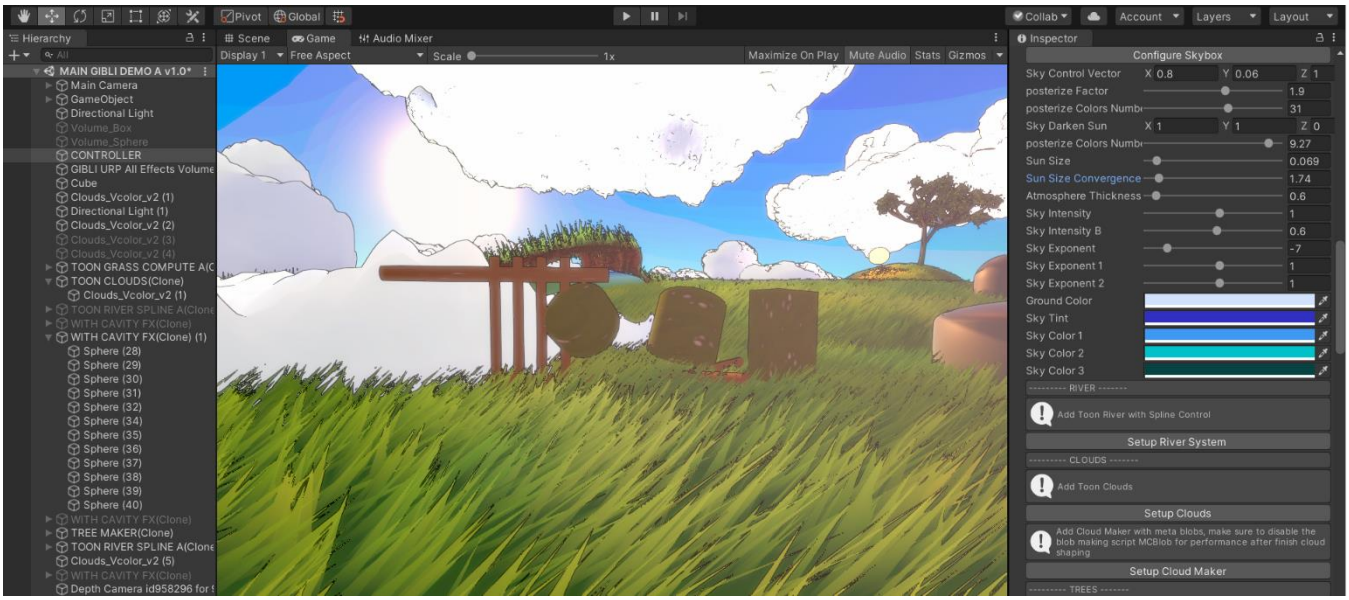
The material has various noise and wind options that can be tweaked to get the desired dynamic effect. Note that as the cloud is scaled up those settings values may be required to be reduced a lot to get a desired effect.





Toon Skybox Sub Module

The toon skybox is applied with a special shader to the skybox material of the scene and can be configured through the global manager. The various options can be accessed through the “Configure Skybox” button as shown below.



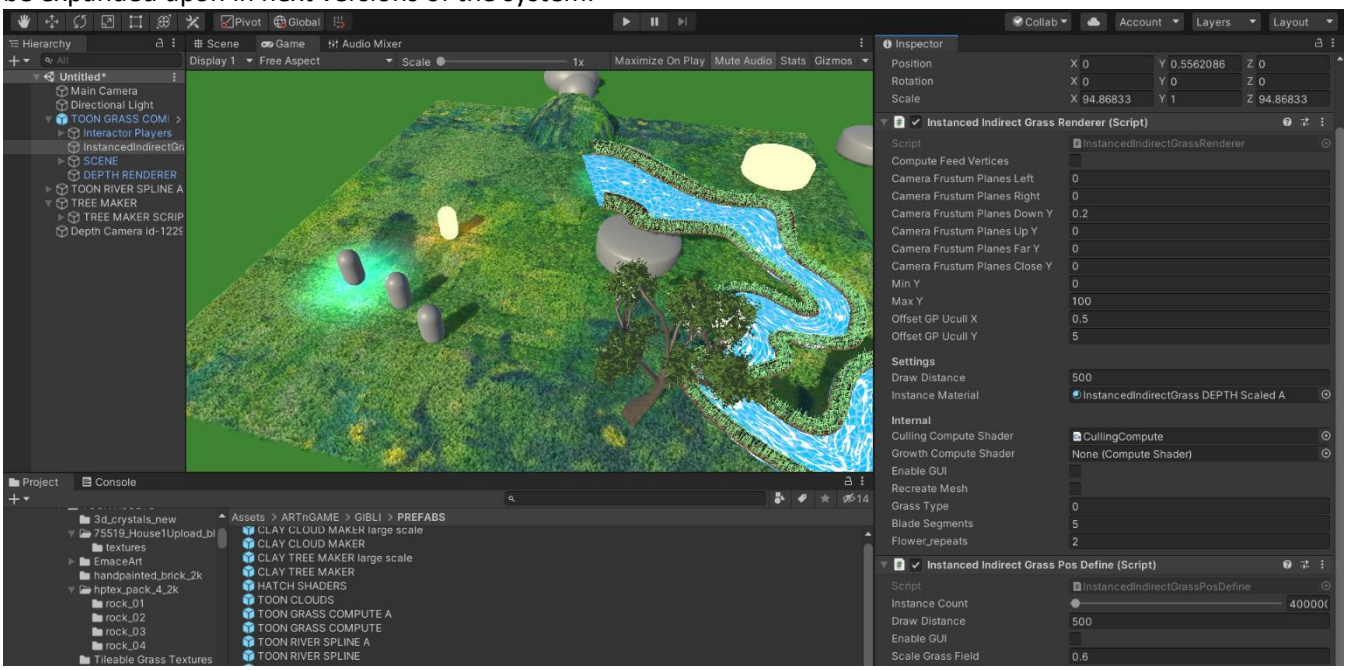
The skybox options include the Sun Size to make the sun larger or smaller, as well as a **posterize effect function** to reduce colors to the “Posterize Color number” amount for a more artistic effect.

There is also tweaking of **the intensity, separation and colors of the various sections of atmosphere** and overall sky tinting.

Toon Grass Instanced Sub Module

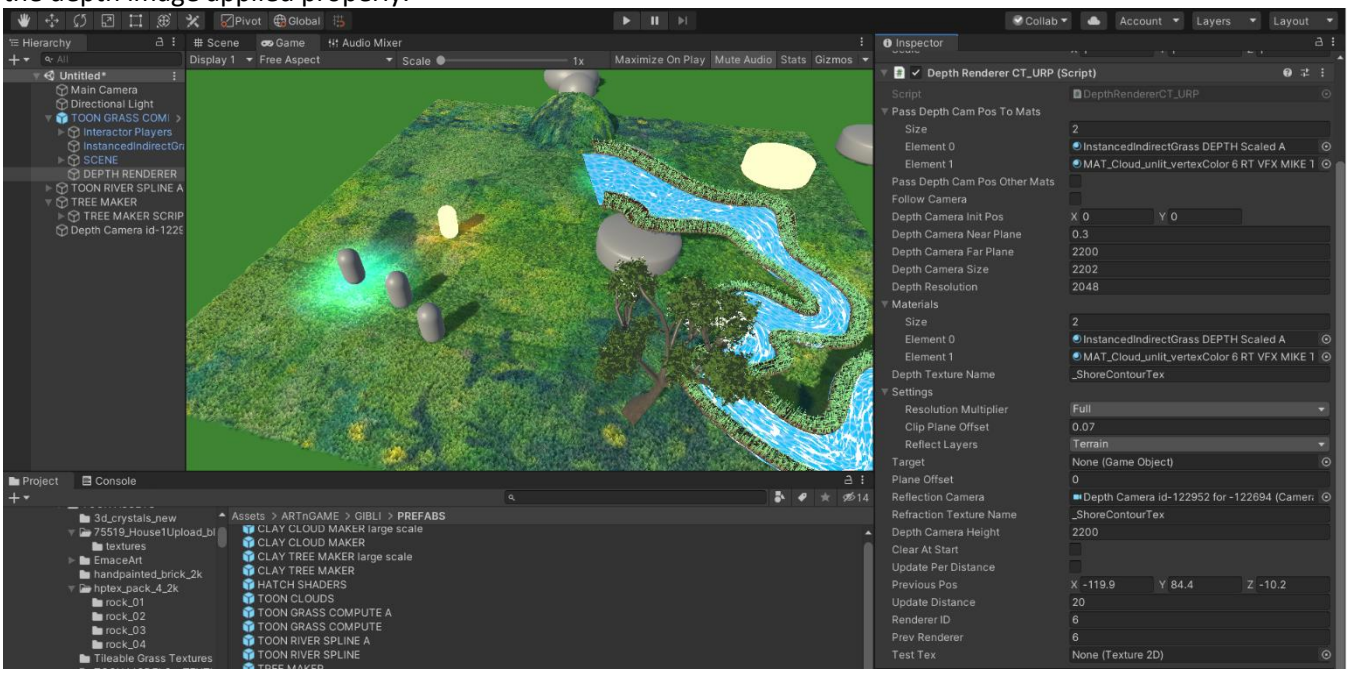
The grass module uses compute shaders and direct draw instantiation to create efficient grass that allows agent interaction using the Grass Bending renderer feature.

The basic Grass Prefab Setup is shown below, the Toon Grass module can be instantiated directly from the Global Manager using the “Setup Instanced Grass” button. This will create the module shown in the image below, with the main script controlling the effect “**Instanced Indirect Grass Renderer**”. The “Compute Feed Vertices” is an experimental mode, where the shader gets the vertices from a compute shader, this is implemented in the Long Grass demo and will be expanded upon in next versions of the system.

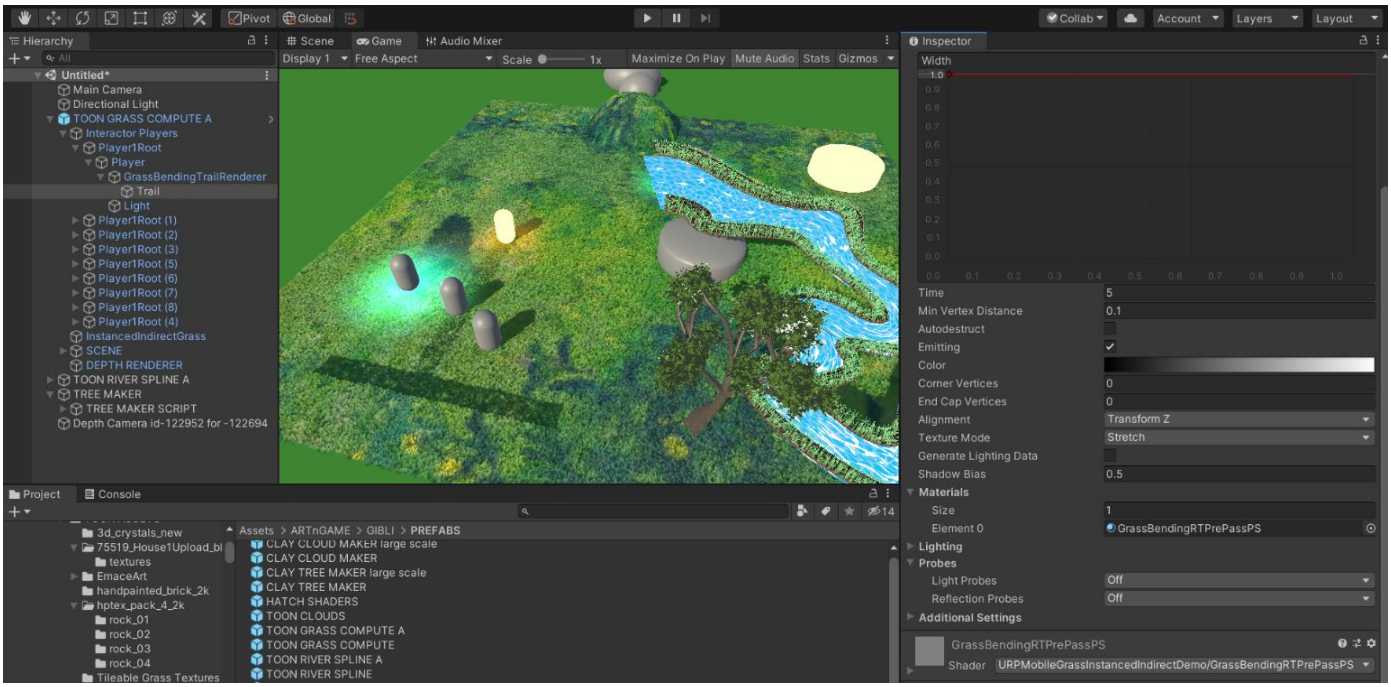


The system uses a top –down depth rendering camera to get the terrain and object height, and can adapt the grass on the objects in the depth rendering camera layer.

The Depth renderer is shown in the following image. The grass material must be referenced in the materials list to have the depth image applied properly.

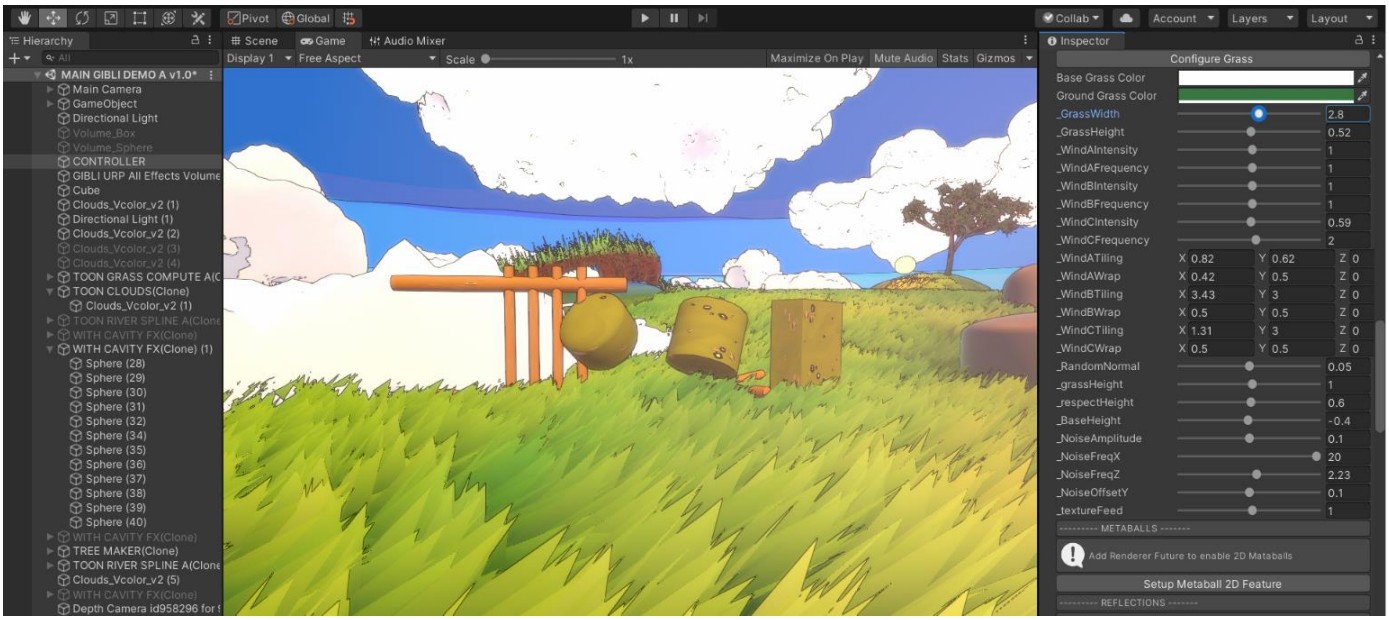


The system also allows interaction of agents with the grass, using a rendering of a trail and passing it to the shader. The Interactors setup is shown in the image below.



The grass can be tweaked for size and wind directly in its material and also though the Global manager so can have the various properties saved within the scene.

The grass configuration parameters are shown in the image below and are available after press the “Configure grass” button.



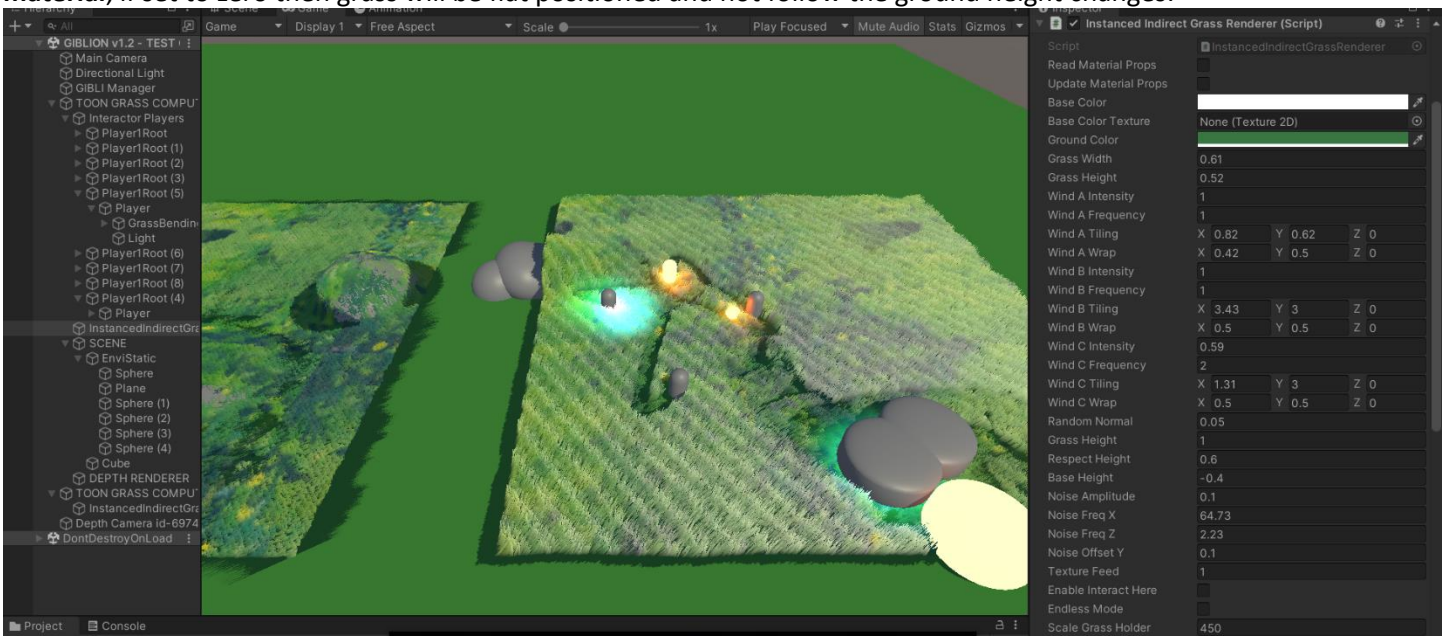
Use a Texture Feed other than one to make the grass flat and not follow the terrain contour as given by the top down depth rendering camera.

Note that the renderer ID variable of the Depth Renderer script must be set in a renderer ideally without any image effects applied.

Update for GIBLION (v1.1 & v1.2) – The system in v1.2 is now having **multiple new options**, an option to **use multiple grass fields** in the same scene (in "**InstancedIndirectGrassPosDefine**" script, "**Use Local Instance**" checkbox in all grass field generators in the scene), to recreate the grass positions on demand and to shape – eliminate grass based on a user defined and paintable mask (this feature is under development, a first version with interactor erase grass is available for testing its possibilities).

Also the **grass properties can now be read from the material assigned to the grass field and saved in the script**, using the "**Read Materials Props**" checkbox and also can be updated by the script using the "**Update Materials Props**" checkbox. This allows to use the same material across scenes, with parameters reloaded to the material from the script.

The "**Texture feed**" variable is on by default and **applies the depth texture rendered from the Depth Renderer to the material**, if set to zero then grass will be flat positioned and not follow the ground height changes.

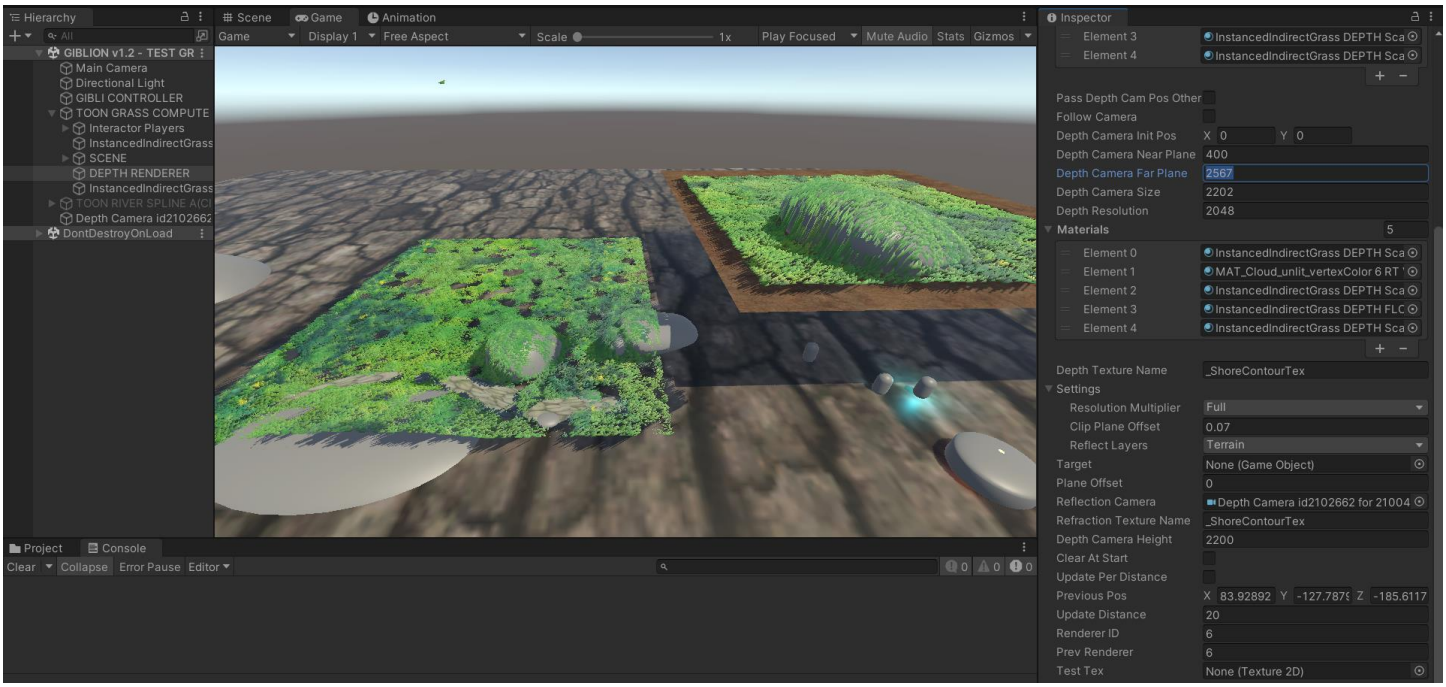


The **"Enable Interact Here"** variable shows to the system which field will be used to draw the interactor's trails.
The **"Endless mode"** can be used to center the grass on the player and follow the player as moves around the map.
In this mode is best to extend the field using the **"Scale Grass Field"** variable in the "**InstancedIndirectGrassPosDefine**" script and increase the blade count in **"Instance Count"** variable to have bigger grass reach in the distance.

Using multiple Grass Fields and fields under zero scene height.
Using two grass fields at heights below zero process, [Video Companion Tutorial] companion.

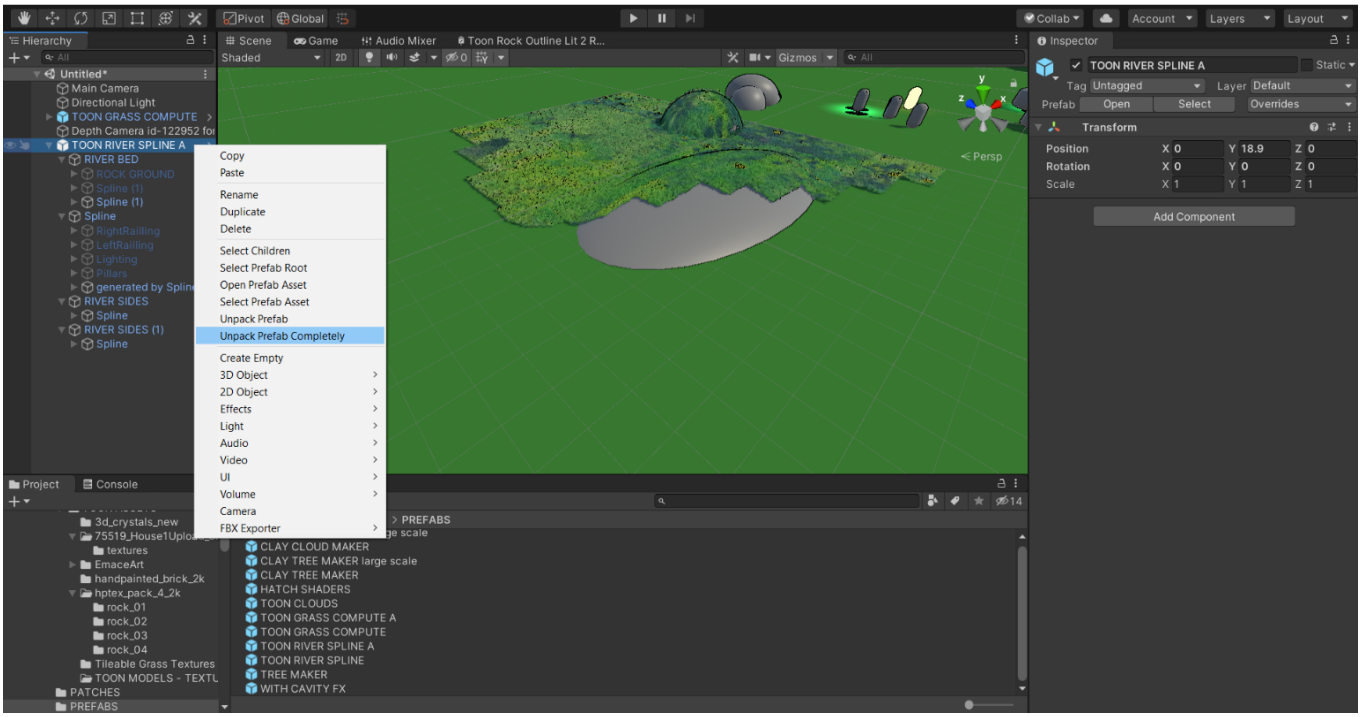
1. Define a grass field by adding them from the GIBLI Manager with **"Setup Instanced Grass" button.**
2. **Copy it to get a second one** and move them around the map
3. Copy the **"InstancedIndirectGrass DEPTH Scaled A WIDE"** material
4. Assign the copied material to the 2ond grass field in **"Instanced Material"** slot in the "**InstancedIndirectGrassRenderer**" script.
5. In the "**CullingCompute** Shader" slot in the 2ond field also assign a copy of the "**CullingCompute**" shader (a copy is already available, named as "**CullingCompute Second Grass Field**")
6. If more than 2 fields are added, **use same process and use extra copies of the material and culling compute shader**.
7. If the fields are above zero height the setup is ready for use, can also enable the endless mode in the scripts to get the grasses appear around the camera in an endless way and can mix endless and specifically placed fields as well.
8. **If the fields are below zero then need few extra steps to define properly the depth camera** that applies the grass to the terrain height.
9. **Note that the Depth Camera must use a Forward Renderer that has no image effects!** Also in the "Depth Renderer CT_URP" script **must assign the Terrain layer to the "Reflect Layers" and also assign that layer to your terrain and any objects** the grass will render on.

9a. **To assign renderer, set the ID in the "Depth Renderer CT_URP" script,** in Renderer ID slot, this corresponds to the placement of the renderer in the pipeline Renderers list.
10. Add the "**InstancedIndirectGrass DEPTH Scaled A WIDE**" and "**InstancedIndirectGrass DEPTH Scaled A WIDE 2**" for the 2ond field materials to both the "**PassDepthCamPosToMats**" and "**Materials**" slots in the "**Depth Renderer CT_URP**" script. This allows the depth rendering camera to pass the real time rendered height map rendertexture to the materials and move the grass on top of the surfaces assigned to the “Terrain” layer.
11. **Note: Only the first field grass is managed and properties are saved in the GIBLI manager**, for each other grass field make sure to assign a different material to the grass field so other scenes not affect it and changes must be done in the material directly or use the new read and update material option in each of the grass field creation scripts.
12. For more than one grass fields **must also check the "Use Local Instance" checkbox** in the "**InstancedIndirectGrassPosDefine**" script.
13. If the grass does not appear or not move with the transform, **enter play mode to reset grass states** or use "**Recreate grass**" in "**InstancedIndirectGrassPosDefine**" script and then disable and re-enable the "**InstancedIndirectGrass**" script.
14. Assume we need to lower the two fields from zero height to minus 300m height as a sample case.
15. In the GIBLI manager (“**Effects controller GIBLI**” script) press "**Configure Grass**" button to alter the 1st field. Put "**_BaseHeight**" to -299. For 2ond field select its material e.g. **"InstancedIndirectGrass DEPTH Scaled A WIDE 2**" and set "**BaseHeight**" variable to -299 as well.
16. Enter play mode and set **"-300" to "MinY"** in both fields in the "**InstancedIndirectGrass**" script.
17. Start playing with "**DepthCameraFarPlane**" variable in the "**Depth Renderer CT_URP**" script, until grass is visible. Play also with "**DepthCameraNearPlane**" variable, this will scale the grass vertically, so it correctly attaches to the surfaces. This can also be used with different number if need grass to not appear on top of surfaces but only on lower sides. For the -300m case, use far plane 2685 and near 400.
18. If the 2ond grass field is not perfectly aligned to ground, **also play with "baseheight" more in the material** to get it to ground. e.g. in the demo used -332.15.
19. Use the "**Interact Here**" option to tell the system at which of the grass fields the interactors will apply to the grass. The other fields must have the checkbox off.
20. **The 2ond field material may also be edited and its properties saved** by enable the "**Read Material props**" and then the "**UpdateMaterialProps**" variables in the "**InstancedIndirectGrass**" script.

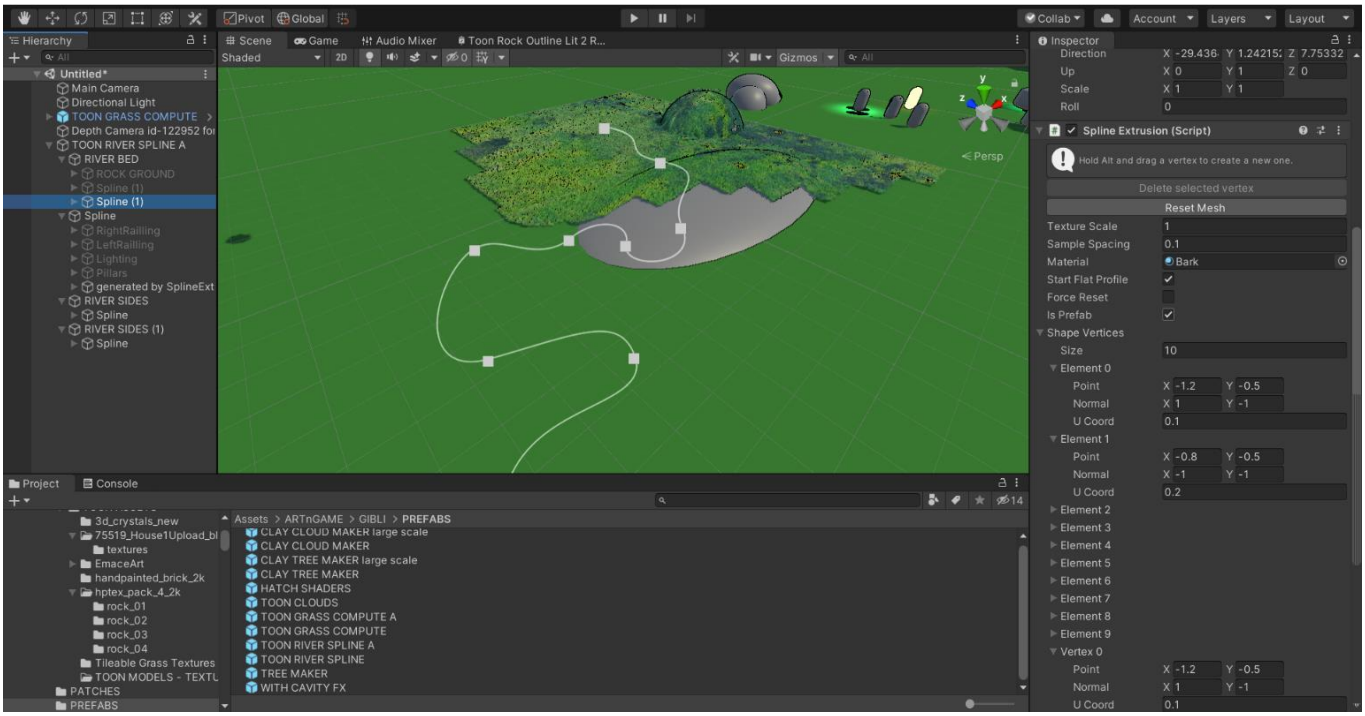


Toon River sub Module

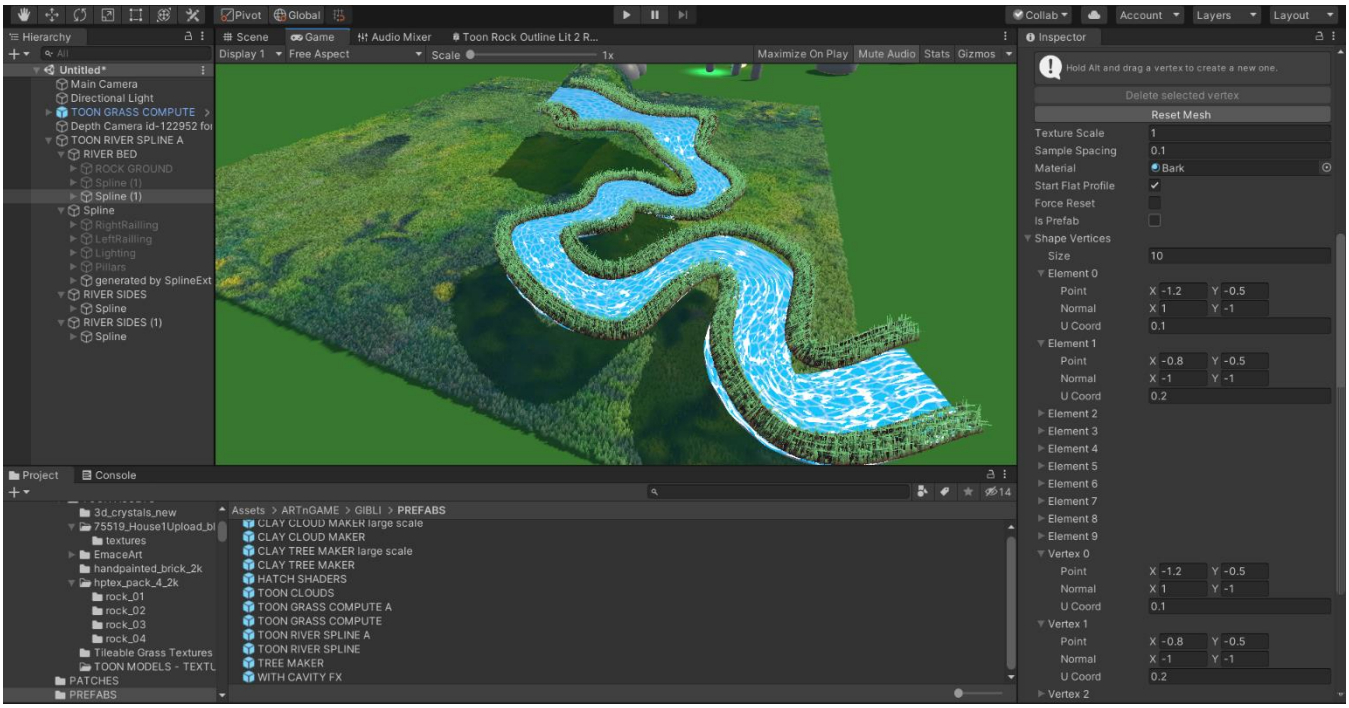
The River prefab with spline setup can be seen below. The system can be instantiated directly from the Global Manager, which handles the breaking of the prefab and setup of the splines to non-prefab mode. The steps if use the prefab directly are described below. First insert the prefab and unpack it.



Then **set in each spline the "isPrefab" to false**, to create the river meshes. Use the same technique if there is a need to prefab the spline meshes, first activate "isPrefab" and then create the prefab, to avoid any issues.

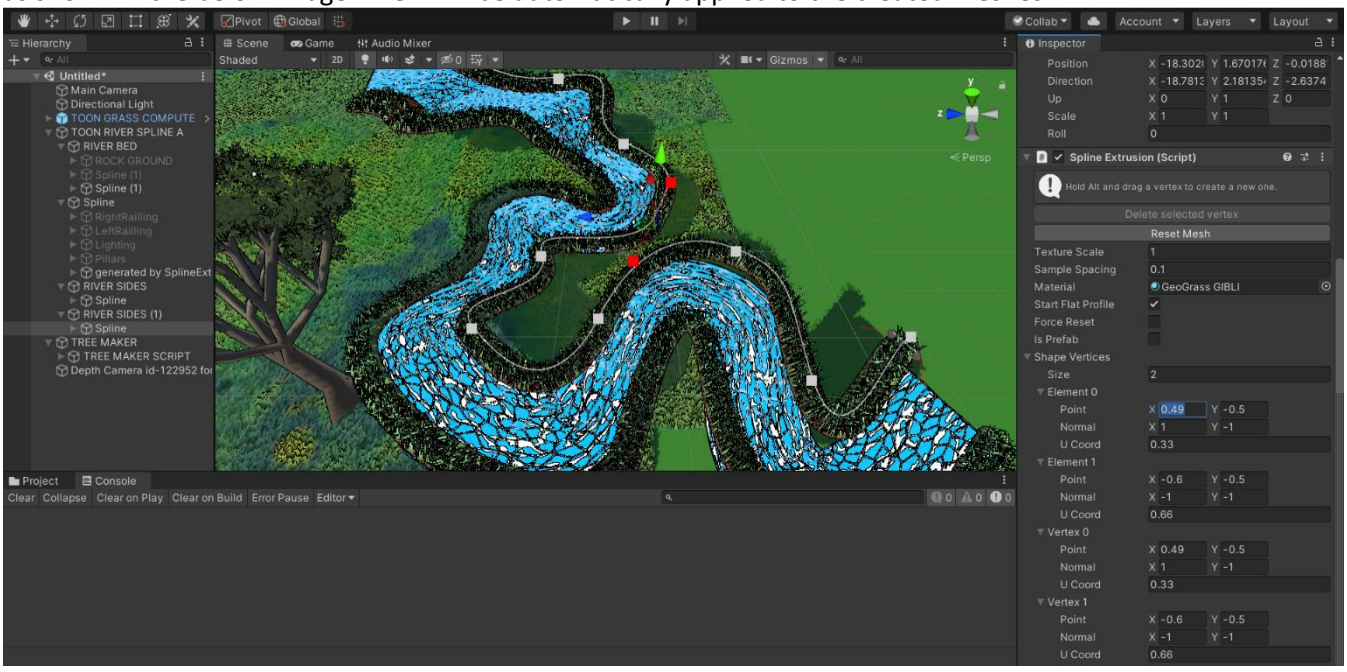


Then can tweak the river using the spline controls and the width and positions in Shape Vertices section. The mesh profile lines can also be edited in the scene view with the available handles near the start of the river.



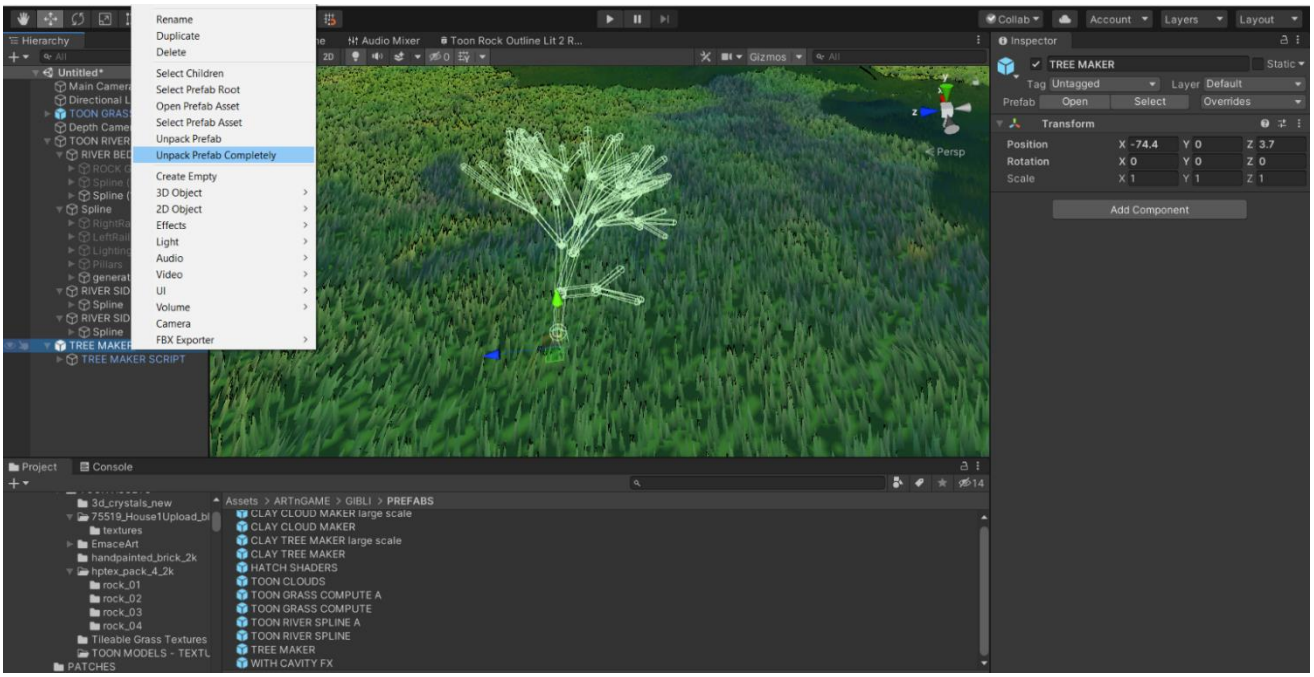
The example prefab uses 4 spline based meshes, one for river bed, one for water and two for the river sides, with a material applied that adds a Geometry Shader based grass to the mesh. This grass does not support Mac devices and is suitable mostly for Windows.

The material assigned to the generated meshes must be declared in the “Material” slot in the “**Spline Extrusion**” script as shown in the below image. Then will be automatically applied to the created meshes.

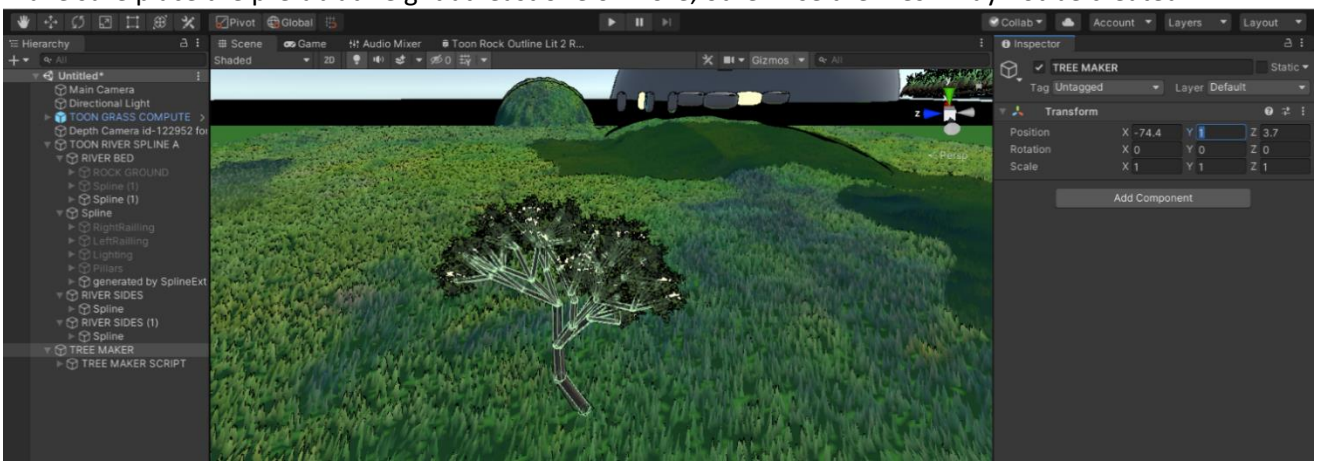


Tree maker Sub Module

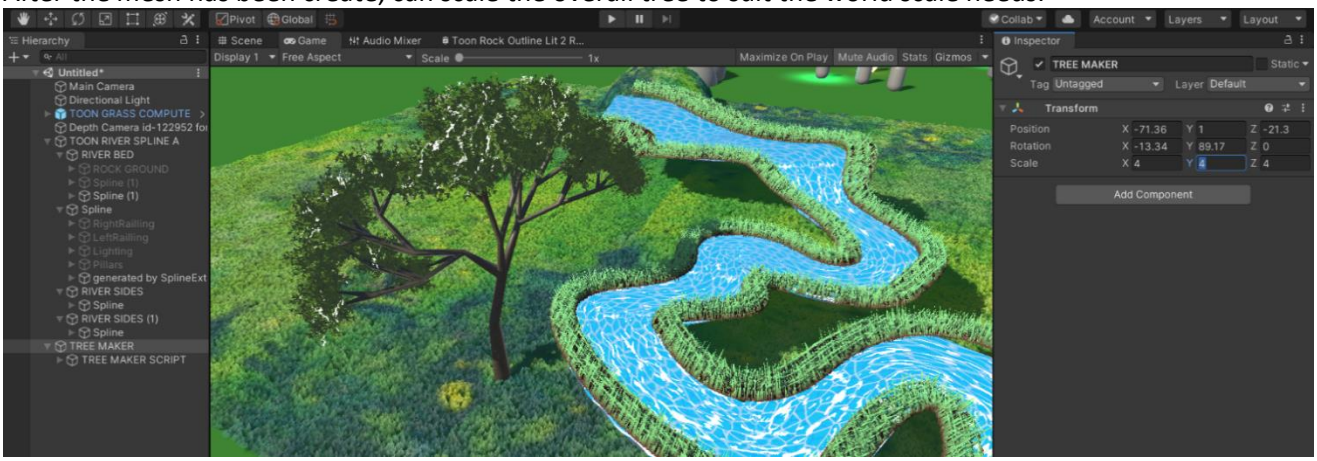
The Tree maker setup from prefab is shown below. These steps are automated if the Tree maker in instantiated through the Global manager. First unpack the prefab after it is inserted in the scene.



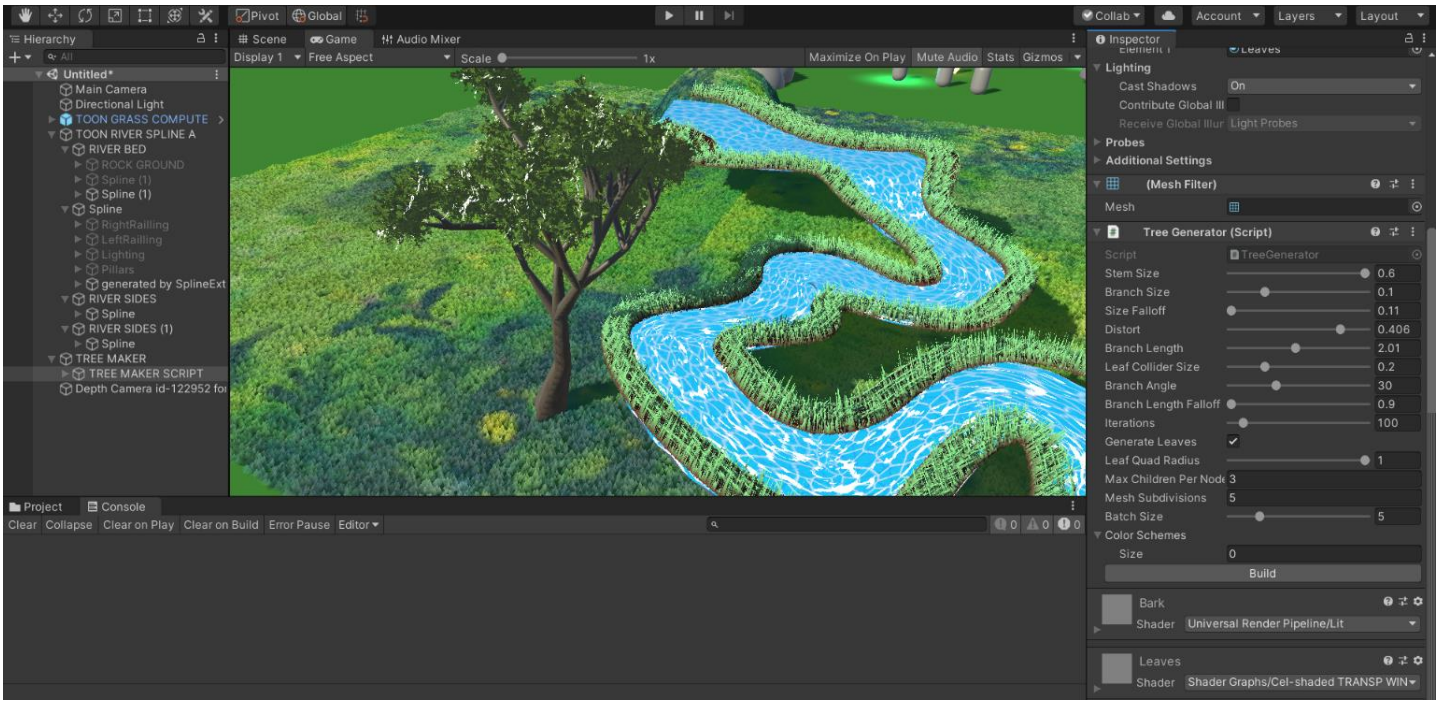
Make sure place the prefab at height at least one of more, otherwise the mesh may not be created.



After the mesh has been create, can scale the overall tree to suit the world scale needs.



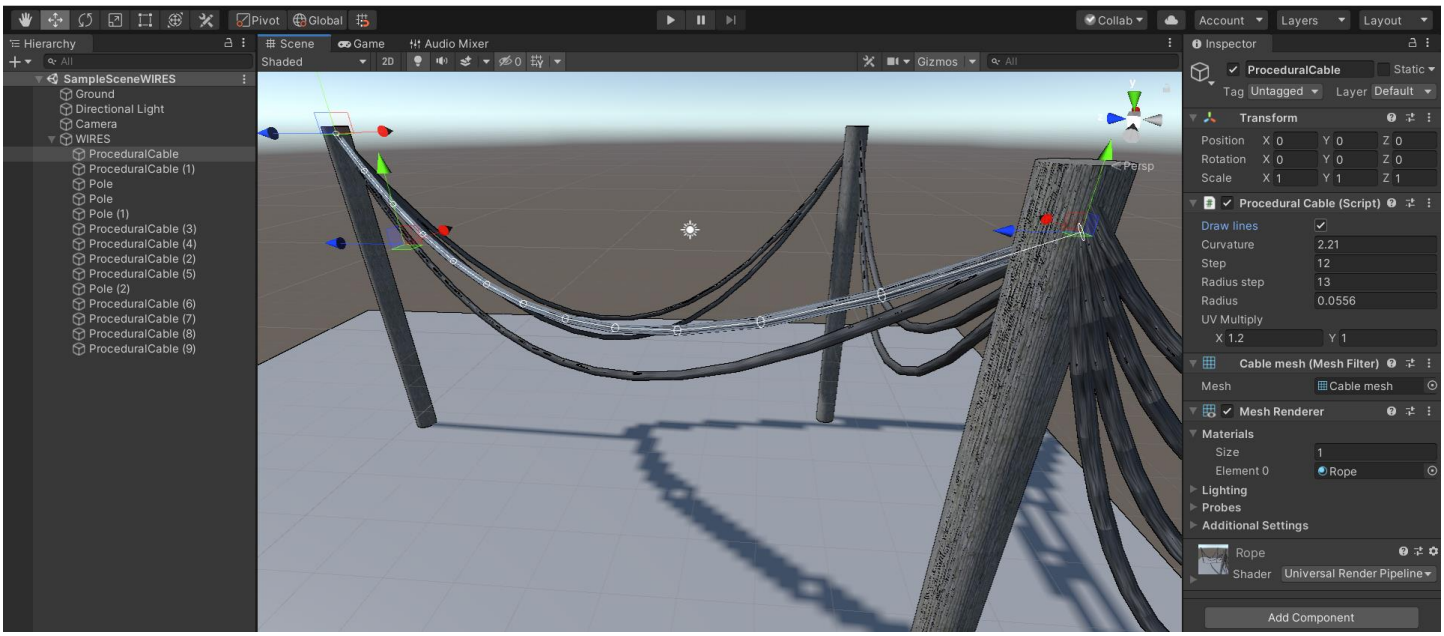
Then the tree can be further randomized and customized in the Tree Generator script, change the variables as needed and then press “Build” button to rebuild the tree with the new properties.



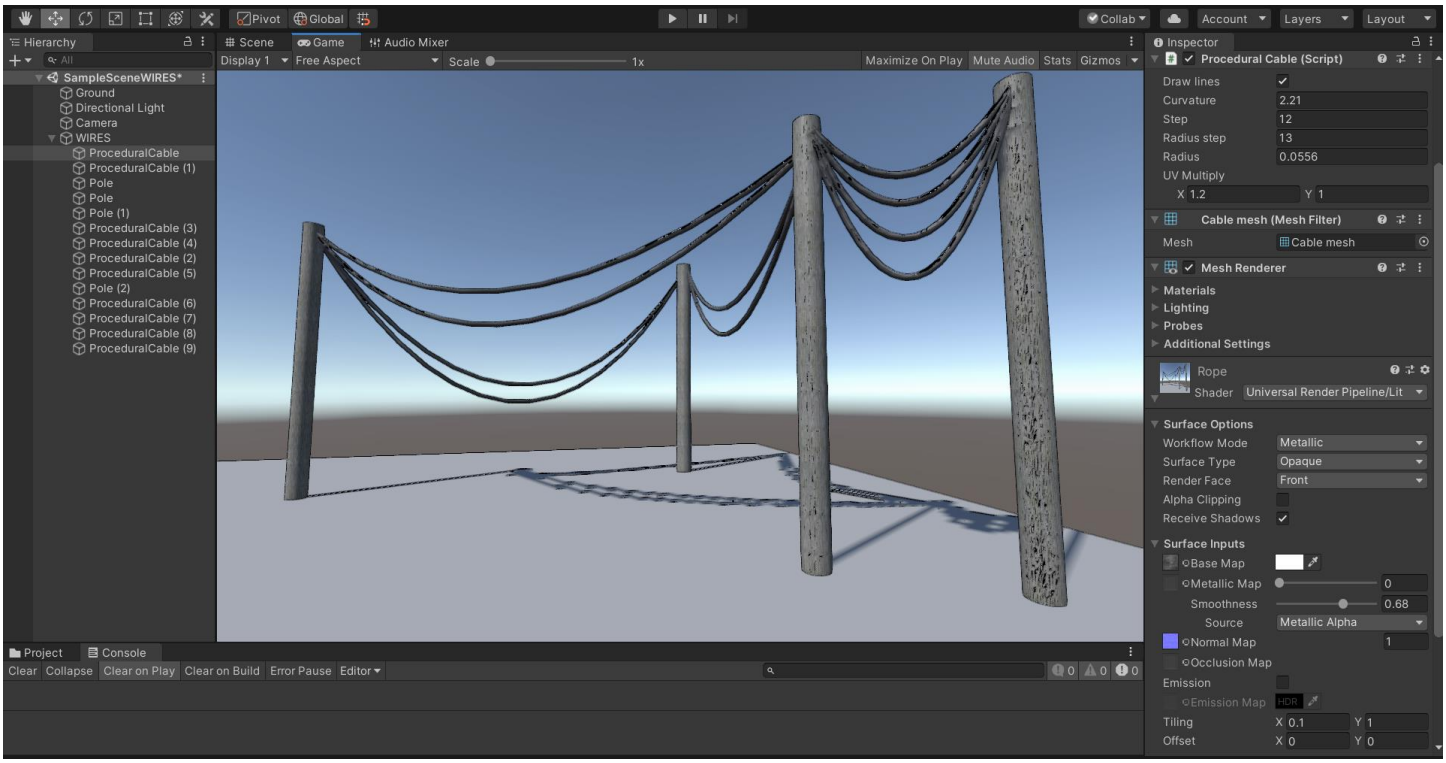
Wires Sub Modules

The system allows for creation of wires in two ways. One is a fixed mesh tweaked directly without dynamics and another a dynamic wire module that can be affected with physics in run time. The second is generally heavier in performance and should be used in moderation.

The fixed mesh wire generator is shown below, the wire can be edited by using the start and end points in the editor, as shown in the below image. If the prefab is used, make sure to first unpack the prefab after insertion in the scene.

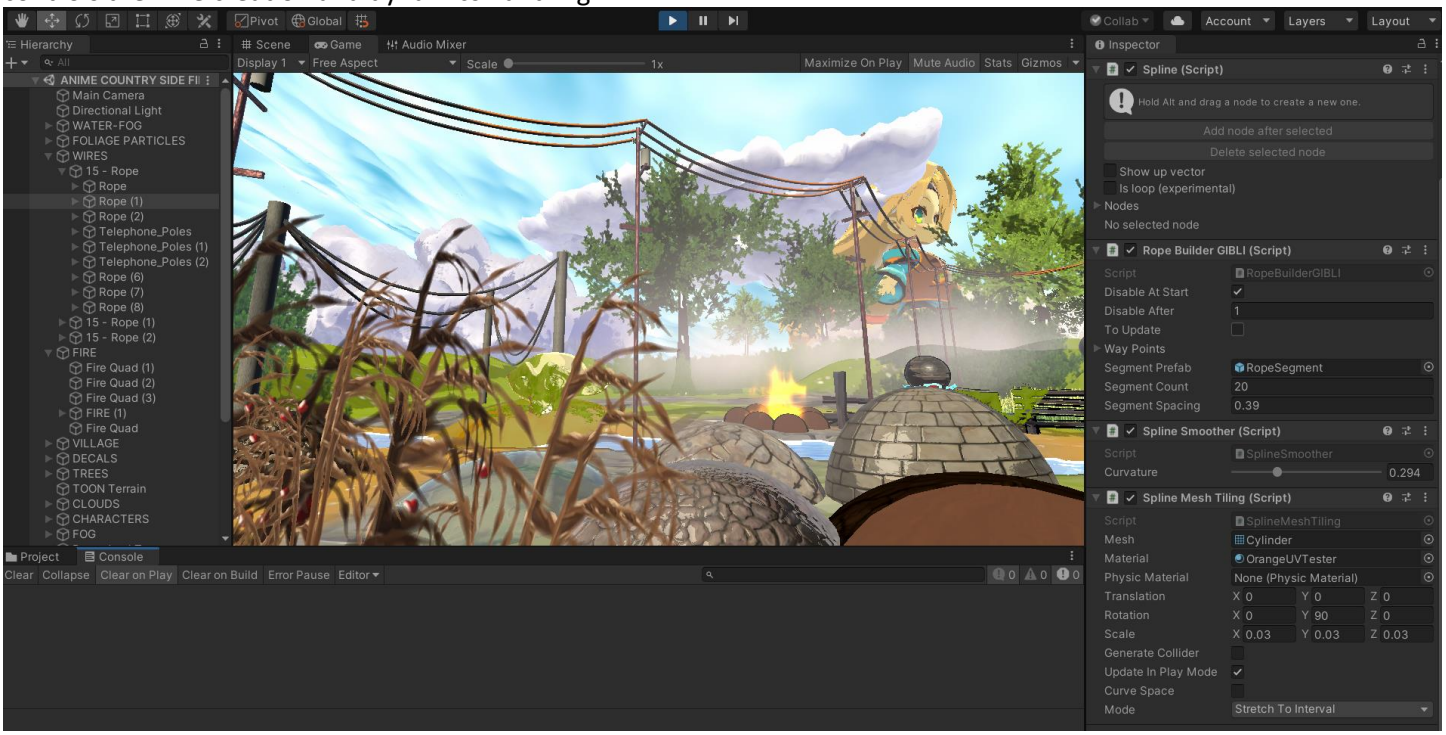


For the rendering of the wires, a normal URP lit or a toon material can be applied in the wire in Mesh Renderer directly.

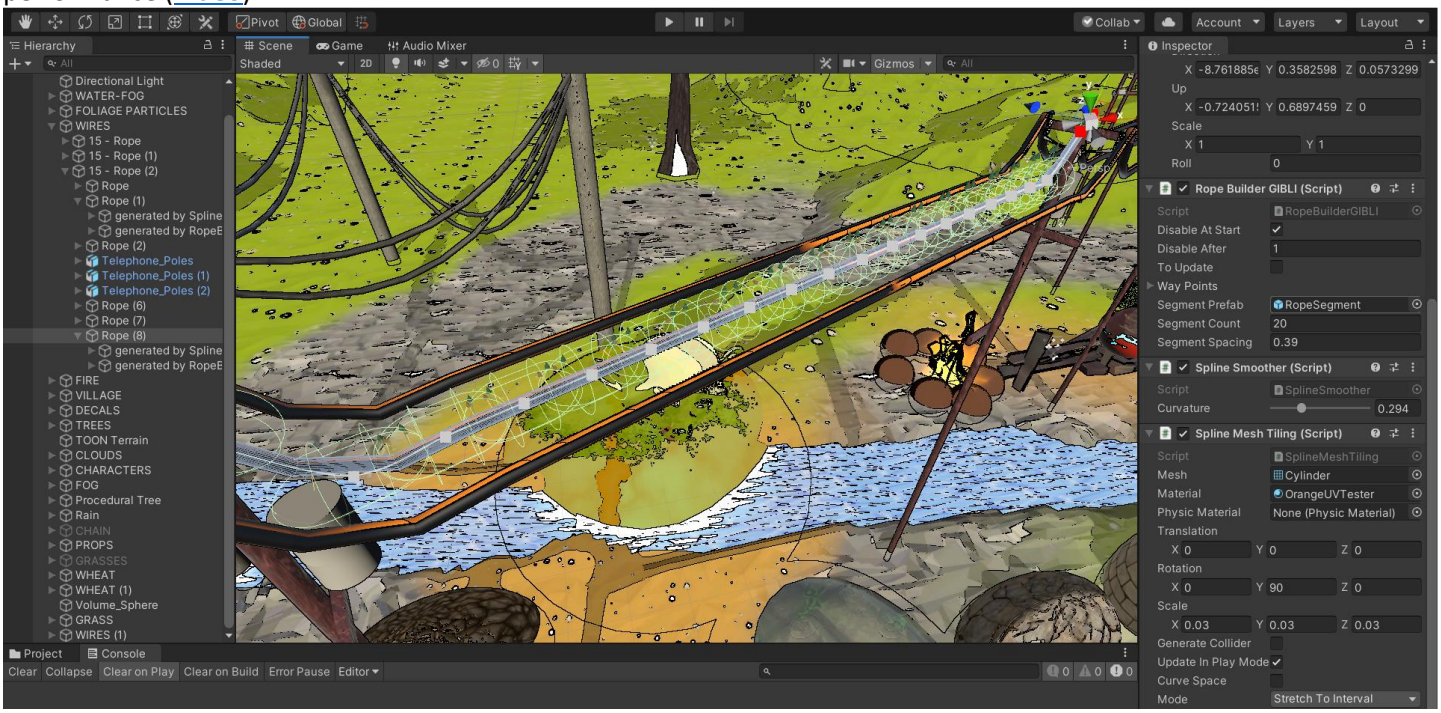


Dynamic wires

The dynamic wires are based on the SplineMesh mesh generation module and the **“Rope Builder GIBLI”** scrip that controls the wire creation and dynamics handling.



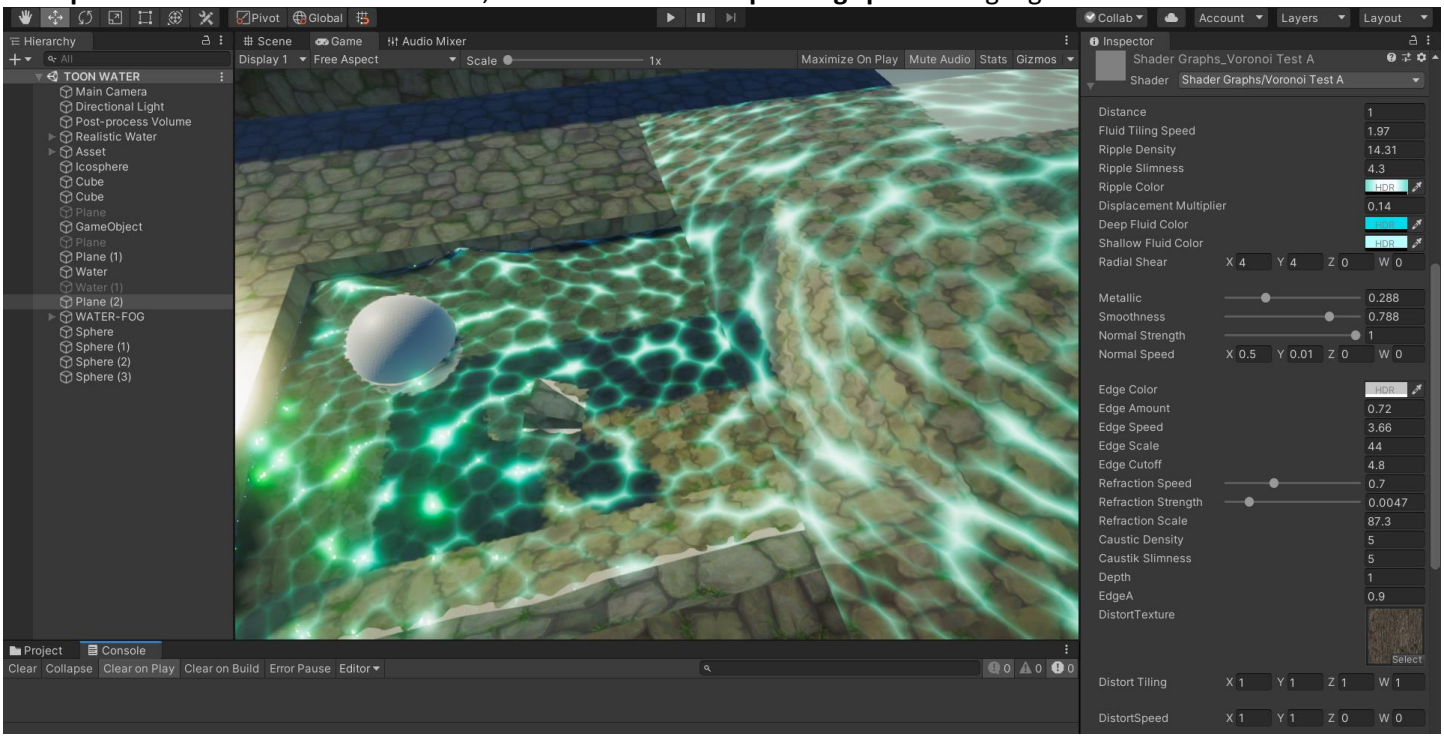
Two prefabs are available in the PREFABS folder for insertion in the scene of the dynamic wires, “WIRES FULL DYNAMIC” and “WIRES DYNAMIC”. The full dynamic version will have the wire dynamic active at all times and is heavier on performance (Video).

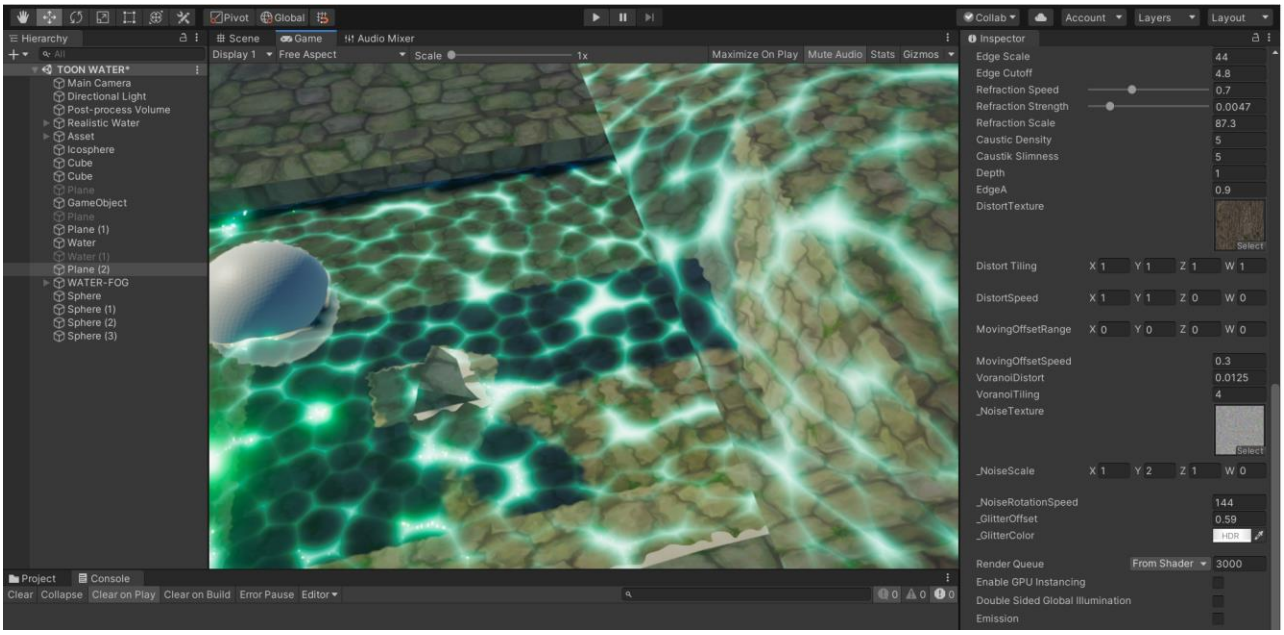


Toon Sea and Water Sub Modules

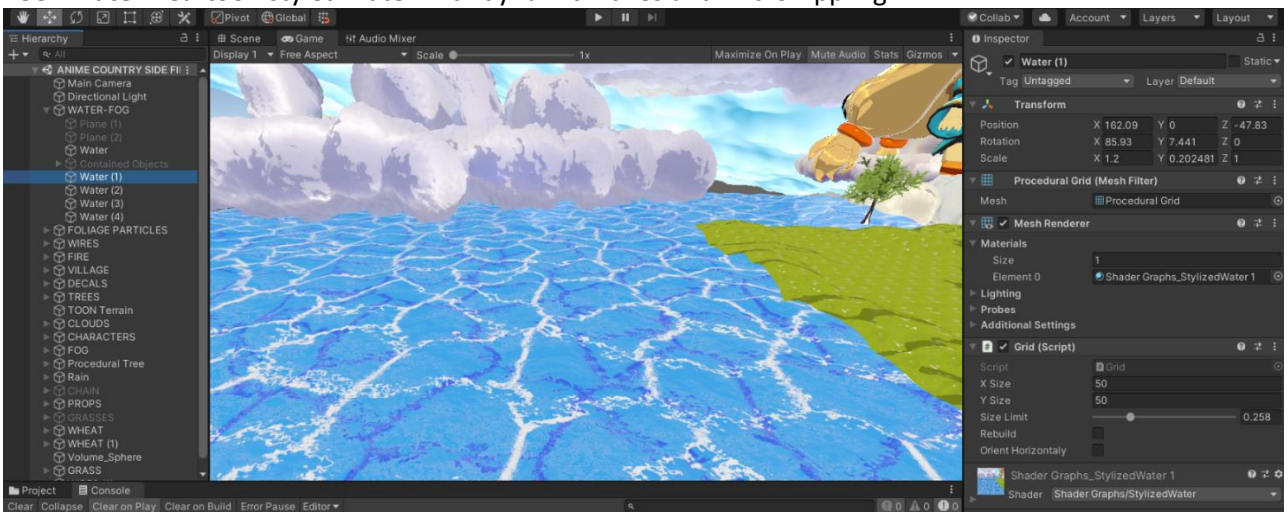
The water modules may be used to create water bodies with various tweaking options. The two water types are presented in the following.

Transparent toon water with refraction, **Voronoi waves and sparkling specular** highlights.

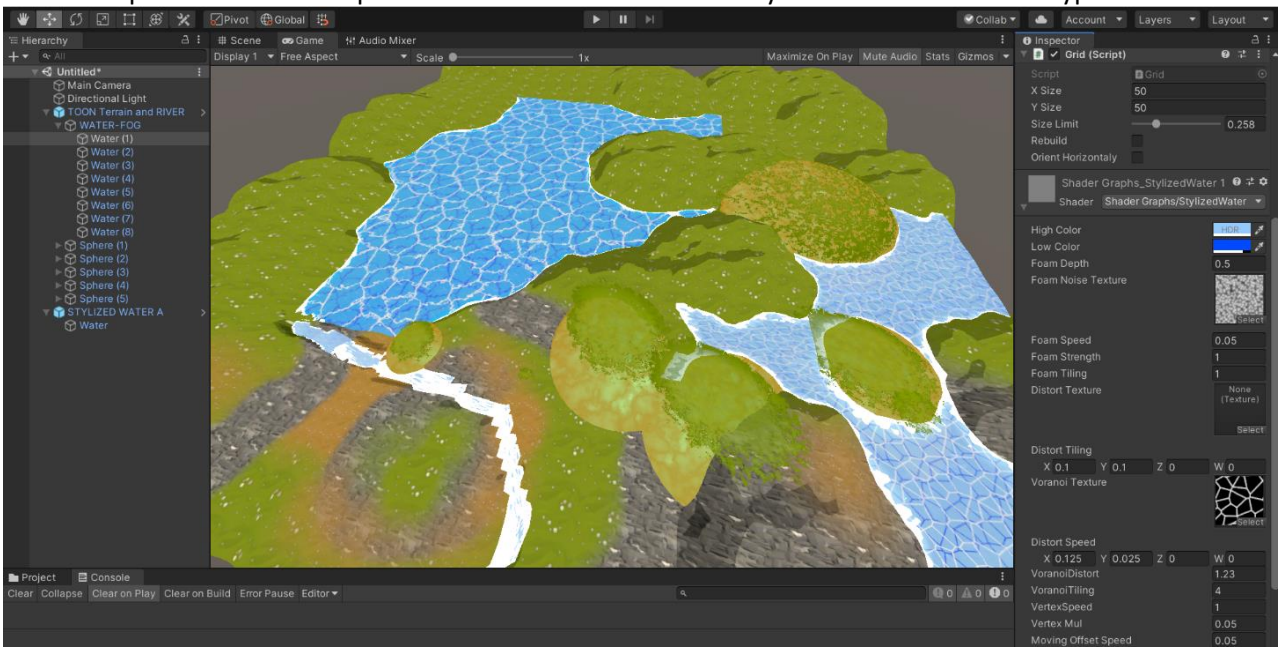




Toon water - Cartoon styled water with dynamic waves and micro rippling

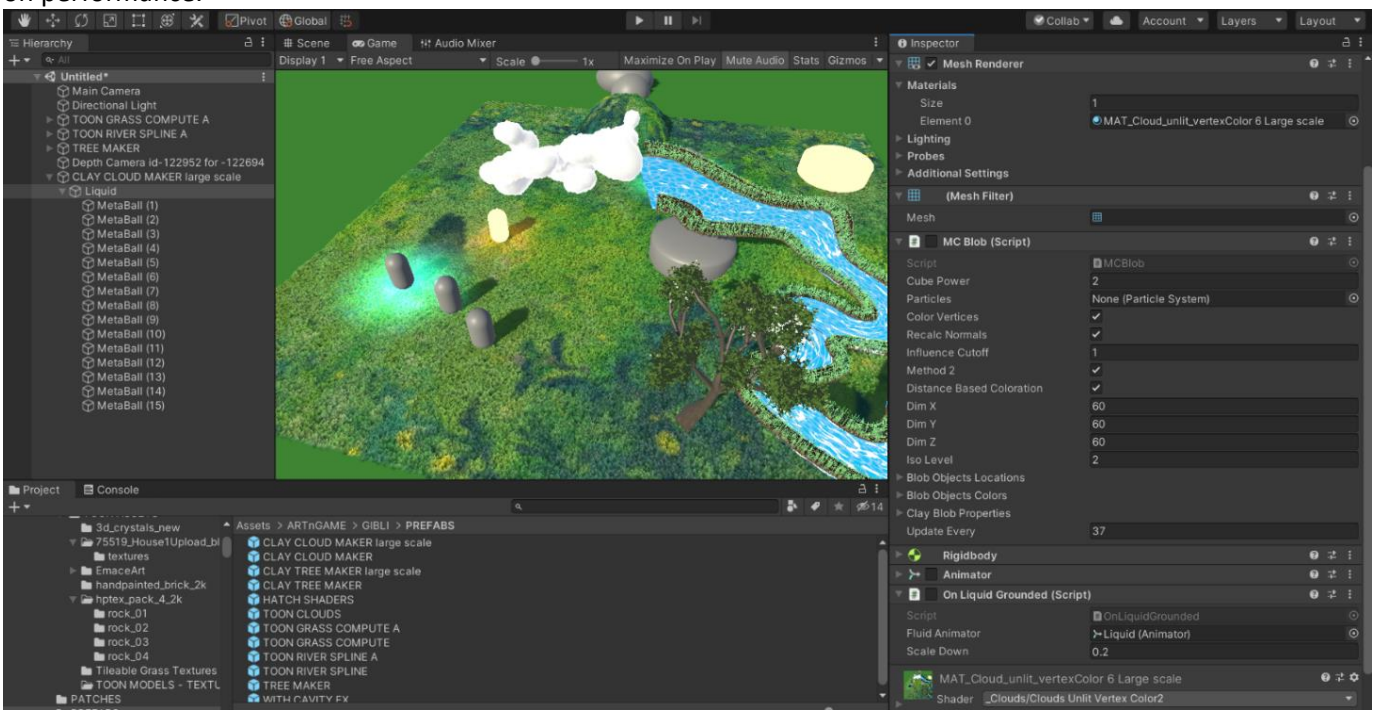


Below are presented the two prefabs that can be used to directly insert the Toon water type in the scene.

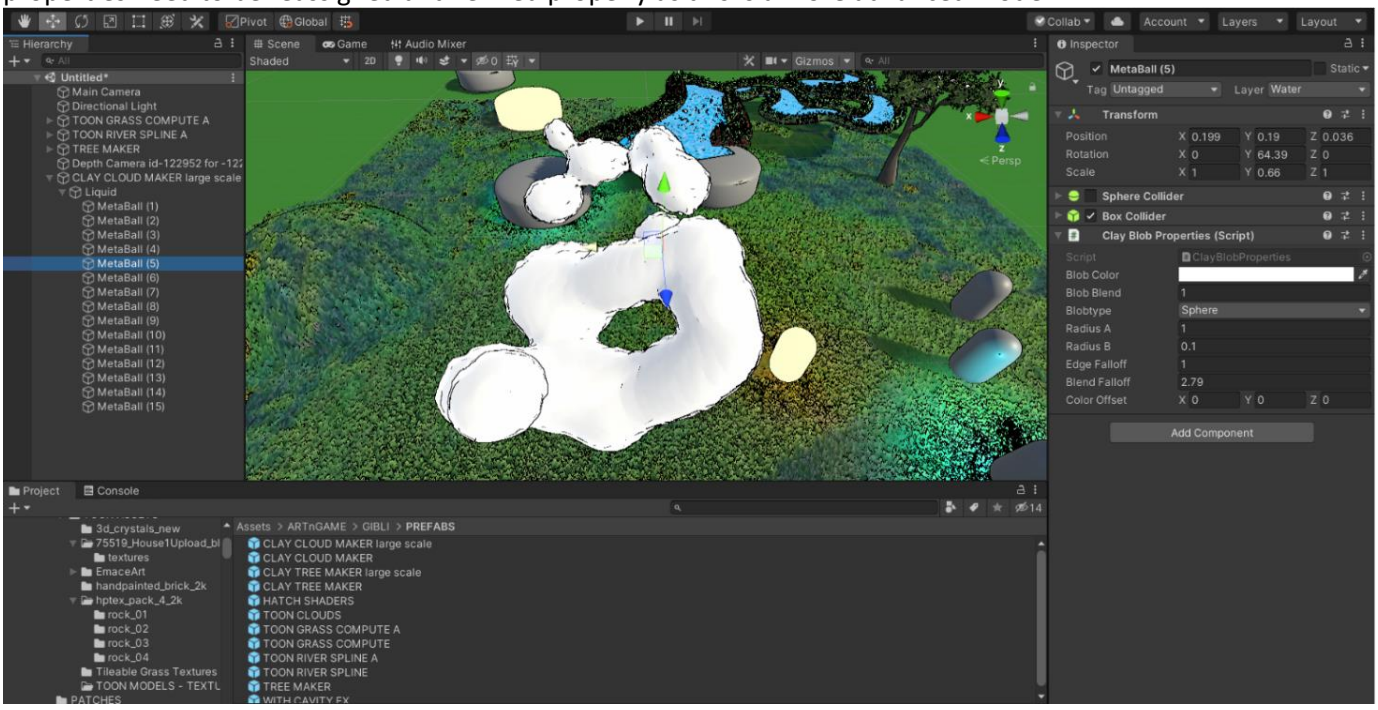


Meta Blob tree and cloud maker

The meta-blob system can be used to define custom shapes for toon trees, especially for Gibli style and other formations like clouds or ground formations. Make sure to enable the “MCBlob” script only in a single blob maker as can be heavy on performance.



Then move the MetaBall items around to reshape the cloud and set their desired shape in Blobtype. Best practice is to use a sphere for clouds and trees, as given in the sample prefab. Squares can also be used, but the Falloff and Blend properties need to be reassigned and refined properly as this is a more advanced mode.

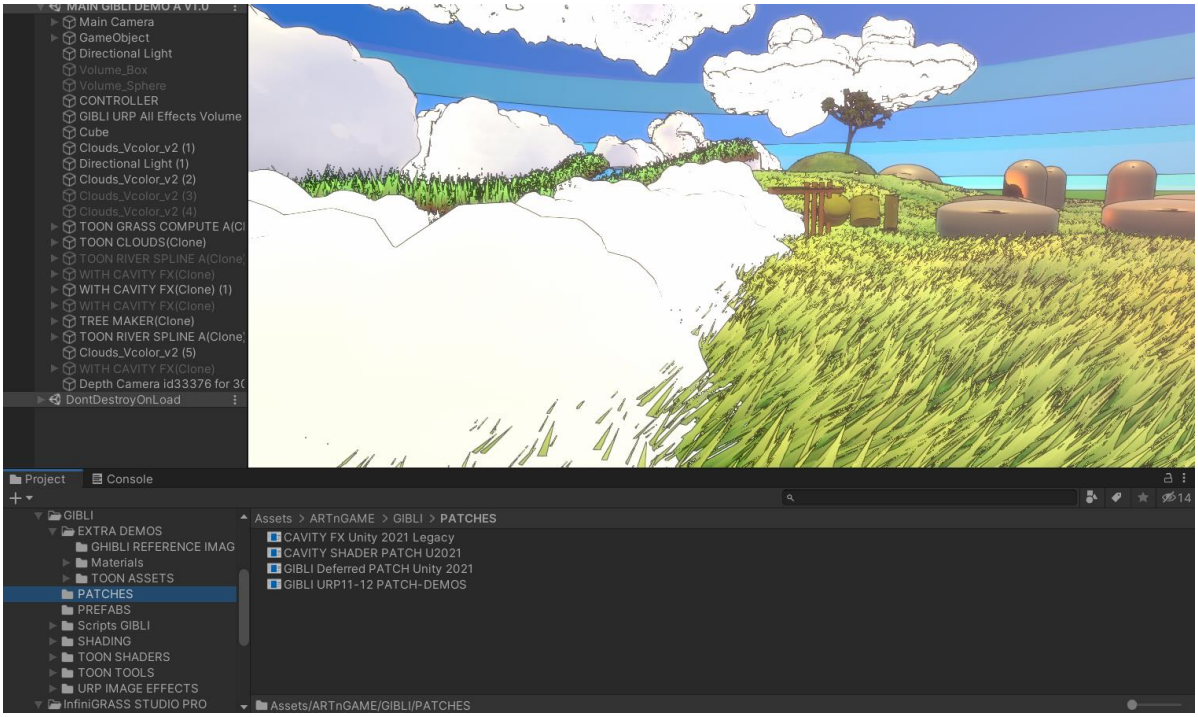


GIBLION PATHES for Unity 2021

Update for GIBLION v1.2 – The system is now available natively for Unity 2021.3 LTS, so if this version and above is used there is no requirement to apply any patches.

The system offers a number of **patches for working perfectly in Unity 2021**, as the main package is exported and supports directly only Unity 2019. For use in Unity 2021.3 LTS, Install the three patches except the “CAVITY FX 2021 Legacy” patch. The “GIBLI Deferred PATCH Unity 2021” is for compatibility of the clouds and grass with Unity 2021 URP new Deferred rendering mode.

For reference, **the main change in the shaders for working with Deferred mode**, is adding of the “Only” in “LightMode”=”UniversalForward” light mode declarations, so becomes “LightMode”=”UniversalForwardOnly”.



GIBLI Forward Renderer reference, with the various Renderer Features applied.