ZIBRA AI

Zibra Liquids User Guide

ZIBRA LIQUIDS IS A PLUGIN FOR THE UNITY ENGINE. IT ALLOWS THE USE OF REAL-TIME LIQUID SIMULATION (GPU) POWERED BY AI BASED OBJECT APPROXIMATION.

VERSION 1.5.2

Free vs Full vs Pro version comparison

Features	Zibra Liquids Free	Zibra Liquids	Zibra Liquids Pro
Mobile support (iOS/Android)	☒	☑	☑
Standalone Meta Quest support	☒	☒	☑
Particle count limit	2 mln	10 mln	10 mln
Multiple liquid/material types in single simulation	☒	☒	☑
Foam	☒	☒	☑
Analytic SDF (simple shapes for colliders/manipulators)	up to 5	☑	☑
Neural SDF (static geometry as shape for collider/manipulator)	☒	only for colliders	☑
Skinned mesh SDF (skinned mesh as shape for collider/manipulator)	☒	☒	☑
Collider friction, Surface tension and Minimum speed options	☒	☑	☑
Force interaction feature	☒	☑	☑
Emitters	only 1 instance	☑	☑
Voids	☒	☑	☑
Detectors	☒	☑	☑
Force fields	only radial force field only 1 instance	☑	☑
Manipulator statistics (e.g. number of particles destroyed by void)	☒	☑	☑
Initial state baking	☒	☑	☑

Table of contents:

System Requirements

Editor platforms: PC (DX11/DX12/Vulkan, x86 and x64), macOS(Metal, x64 and arm64)

Build platforms: PC(DX11/DX12/Vulkan, x86 and x64), UWP(DX11/DX12, x86 and x64), macOS(Metal, x64 and arm64), iOS (iOS version 12 or later, iPhone 6s or newer), Android (OpenGL ES 3.1/Vulkan, armv7 and arm64, Android 7.0 or later)
Android support is experimental

Unity version: 2020.3 or later (latest patch version is recommended)

Supported Render Pipelines: SRP, URP, HDRP

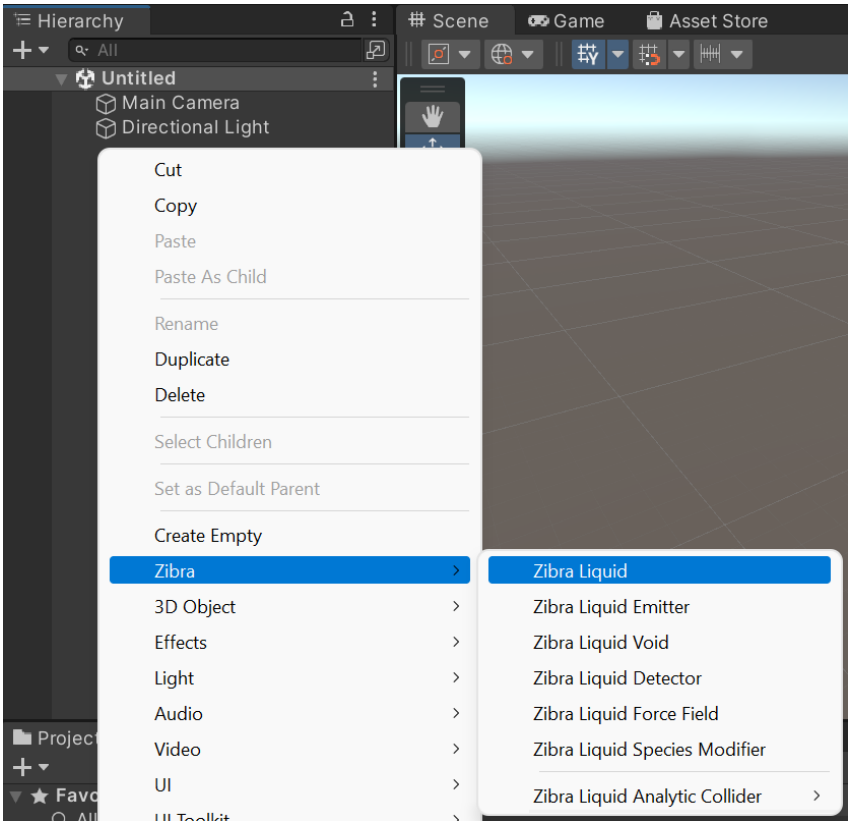
VR is only supported if Render Mode set to Unity Render

Create Zibra Liquid

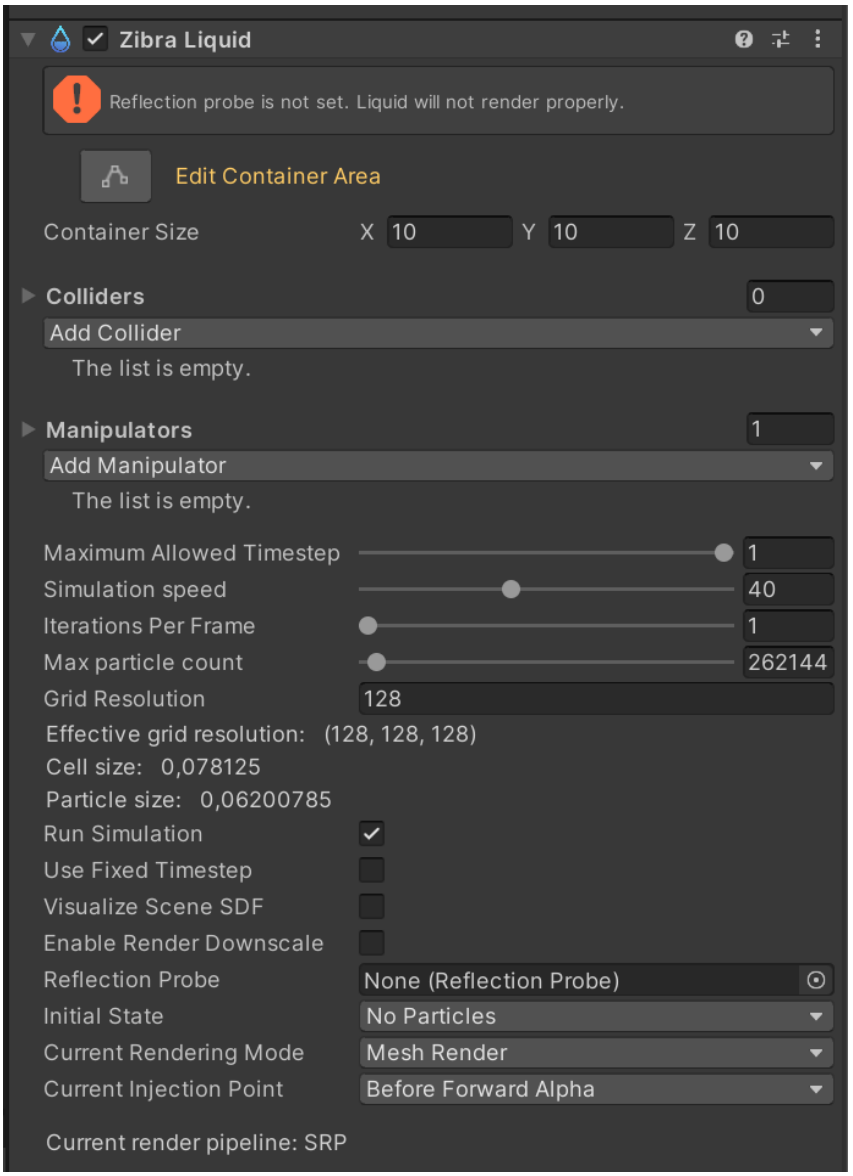
The first thing you need to do is to register your plugin.
After that you can start setting up liquid in your scene.

To create a Zibra Liquid instance:

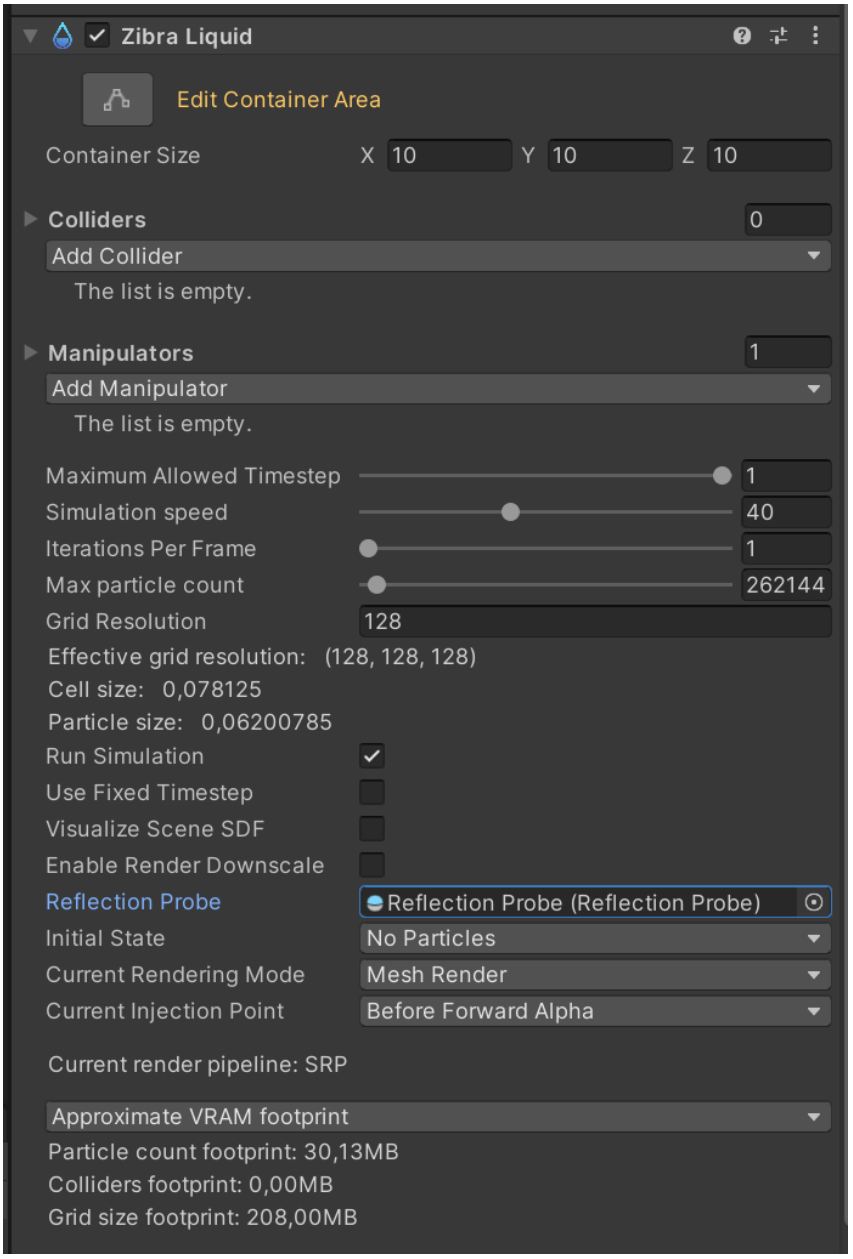
1. Right click in the Hierarchy window and select *“Zibra -> Zibra Liquid”*.



2. In the Inspector window, you'll see the Zibra Liquid parameters:



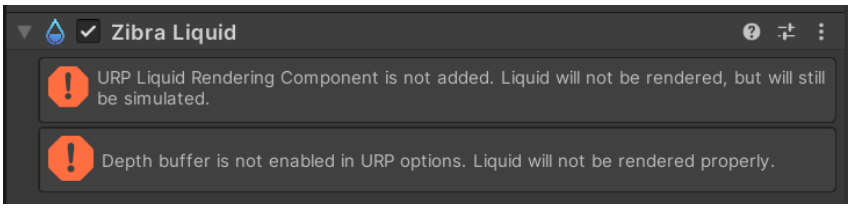
3. Set the “Reflection Probe” parameter. On SRP/URP it’s strongly recommended that you set this parameter, but is not strictly necessary. On HDRP it’s strictly necessary to set this parameter. You can check which render pipeline you are using by looking at the “Current render pipeline” message - on the screenshot the SRP is shown.



4. If you are using SRP, you are now ready to use your liquid. For URP/HDRP please proceed to Additional setup on URP or Additional setup on HDRP accordingly.

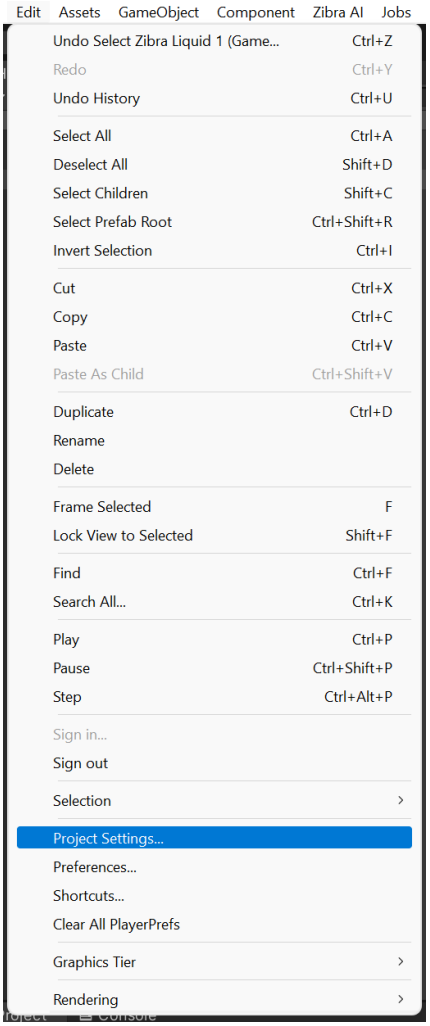
Additional setup on URP

If you are using URP you may see those errors:

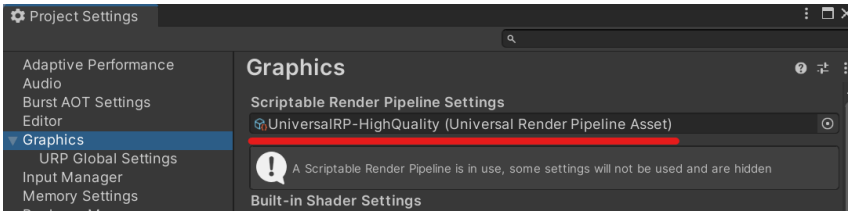


To add *“URP Liquid Rendering Component”* and fix first error:

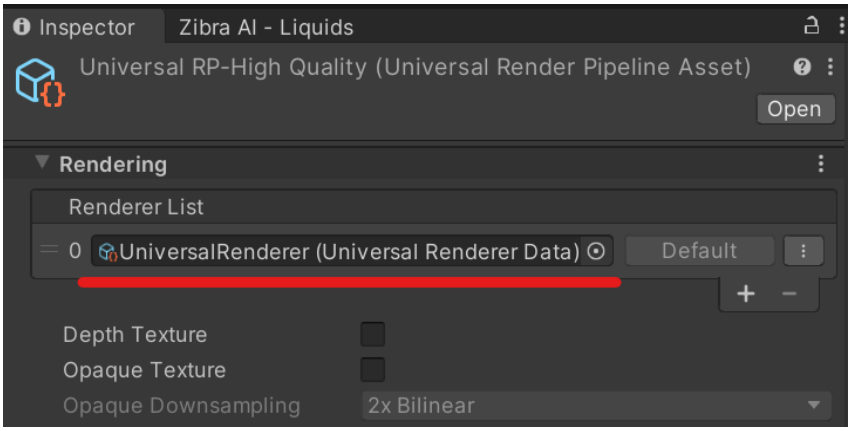
1. Navigate to *“Edit -> Project Settings…”*



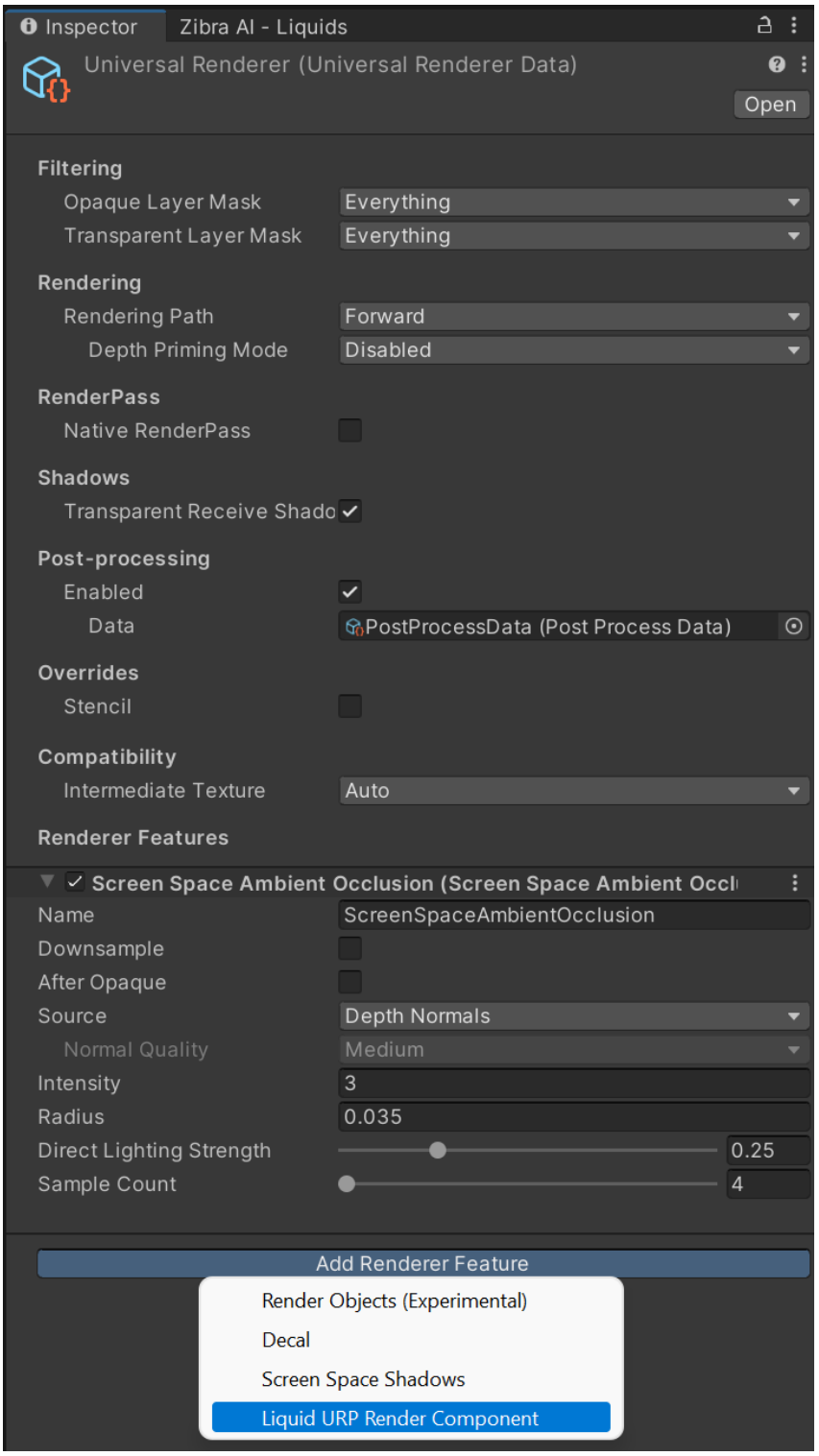
2. From there go to Graphics and open your current Scriptable Render Pipeline Settings asset in the Inspector (*you can do it by double clicking it*).



3. Now you can see the Renderer List. Open your default Renderer asset in the Inspector (*you can do it by double clicking it*). You may need to repeat following steps for non default Renderers if you intend on using liquid with them.

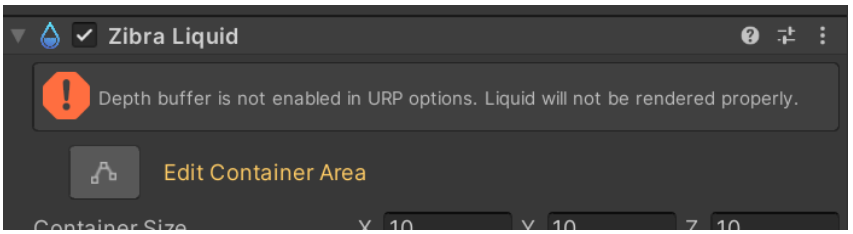


4. Add Liquid the “Liquid URP Render Component”



(Specific UI elements can vary from version to version)

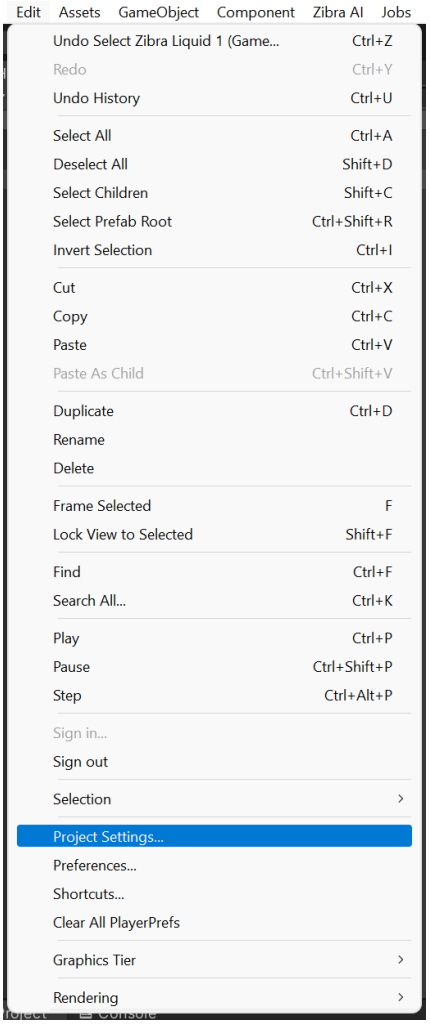
5. If you did everything correctly, that error in Zibra Liquid will disappear:



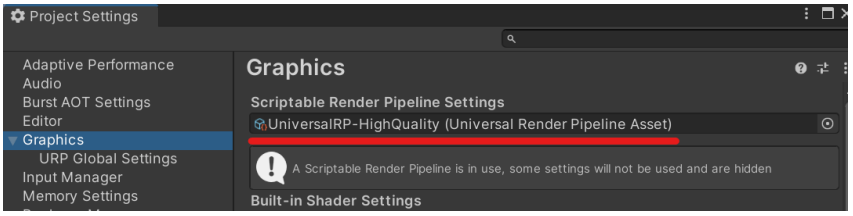
Note that adding *“URP Liquid Rendering Component”* is project wide, and is only needed once.

To enable depth buffer and fix second error:

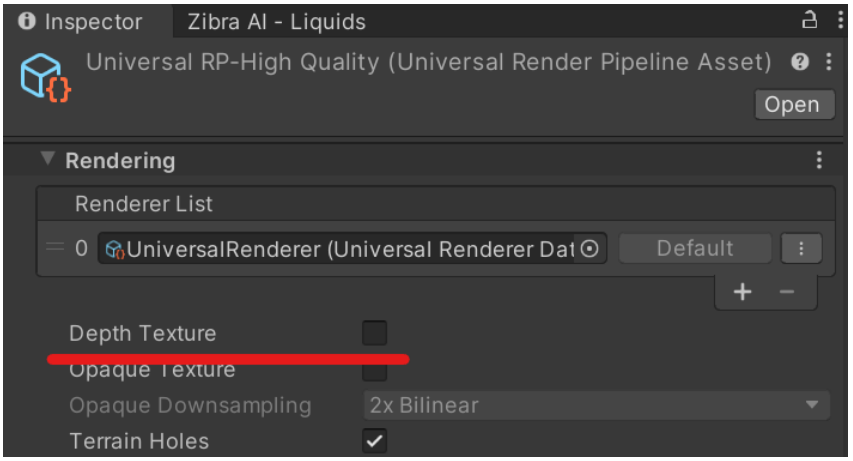
1. Navigate to *“Edit -> Project Settings…”*



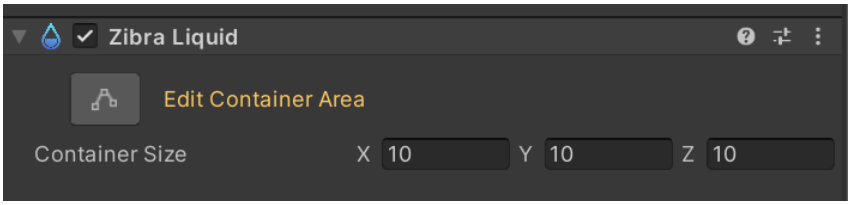
2. From there go to Graphics and open your current Scriptable Render Pipeline Settings asset in the Inspector (*you can do it by double clicking it*).



3. Enable the Depth Texture option



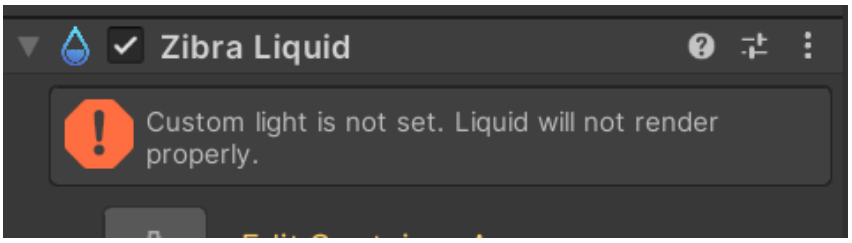
4. If you have any additional Universal Render Pipeline assets used in your project (e.g. for use on Mobile devices), please enable Depth Texture
5. If you did everything correctly, that error in Zibra Liquid will disappear:



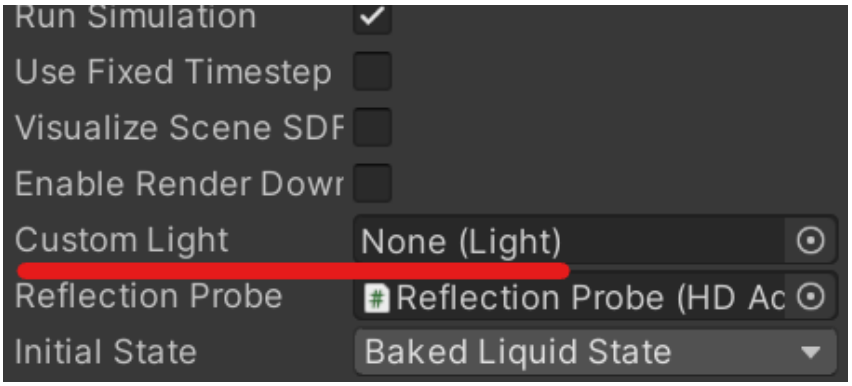
Note that enabling depth texture is project wide, and is only needed once.

Additional setup on HDRP

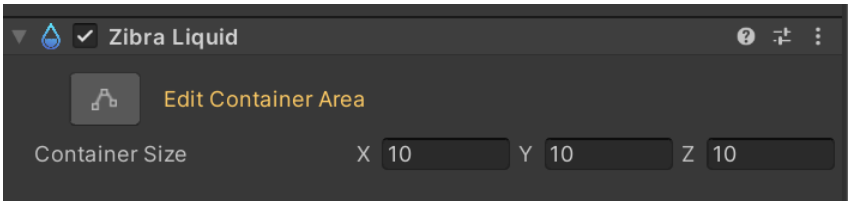
If you are using HDRP you'll see this message:



On HDRP you also need to set the Custom Light parameter.
This parameter sets which light will be used for the liquid lighting.
Currently, only 1 light can be used for lighting the liquid.

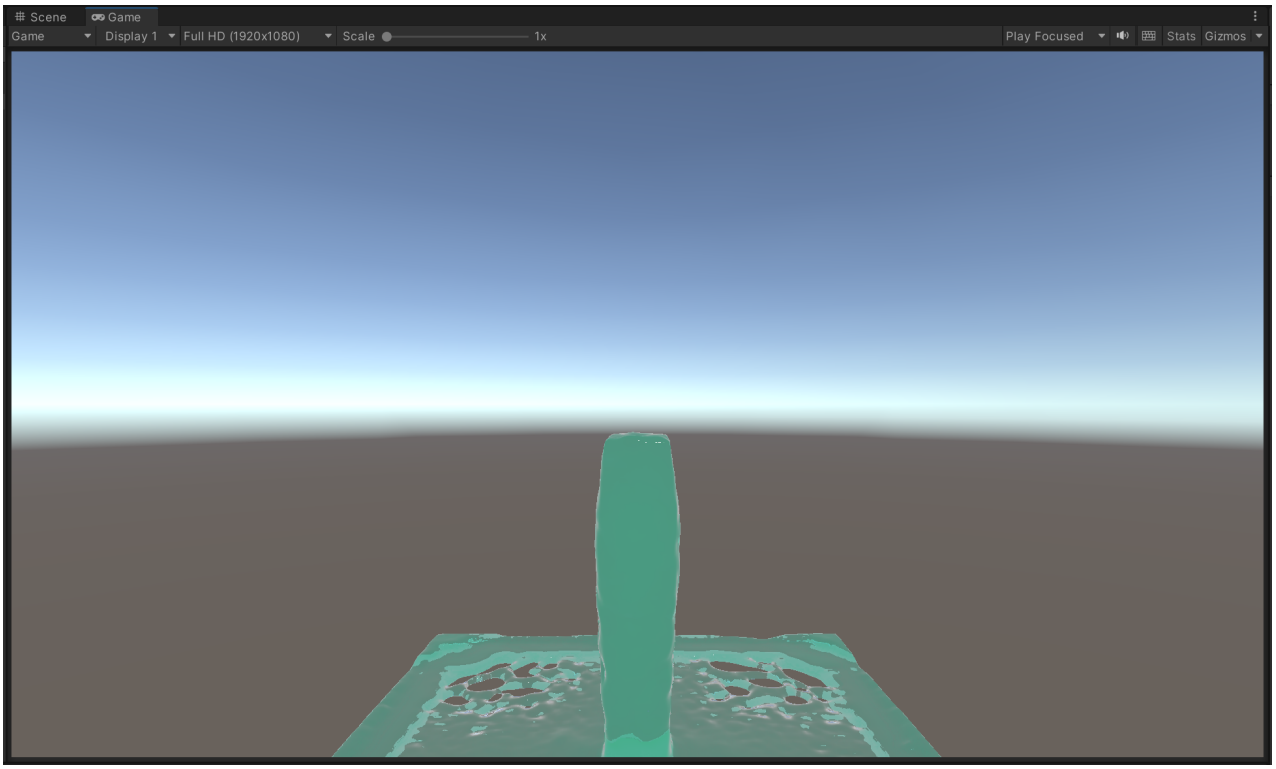


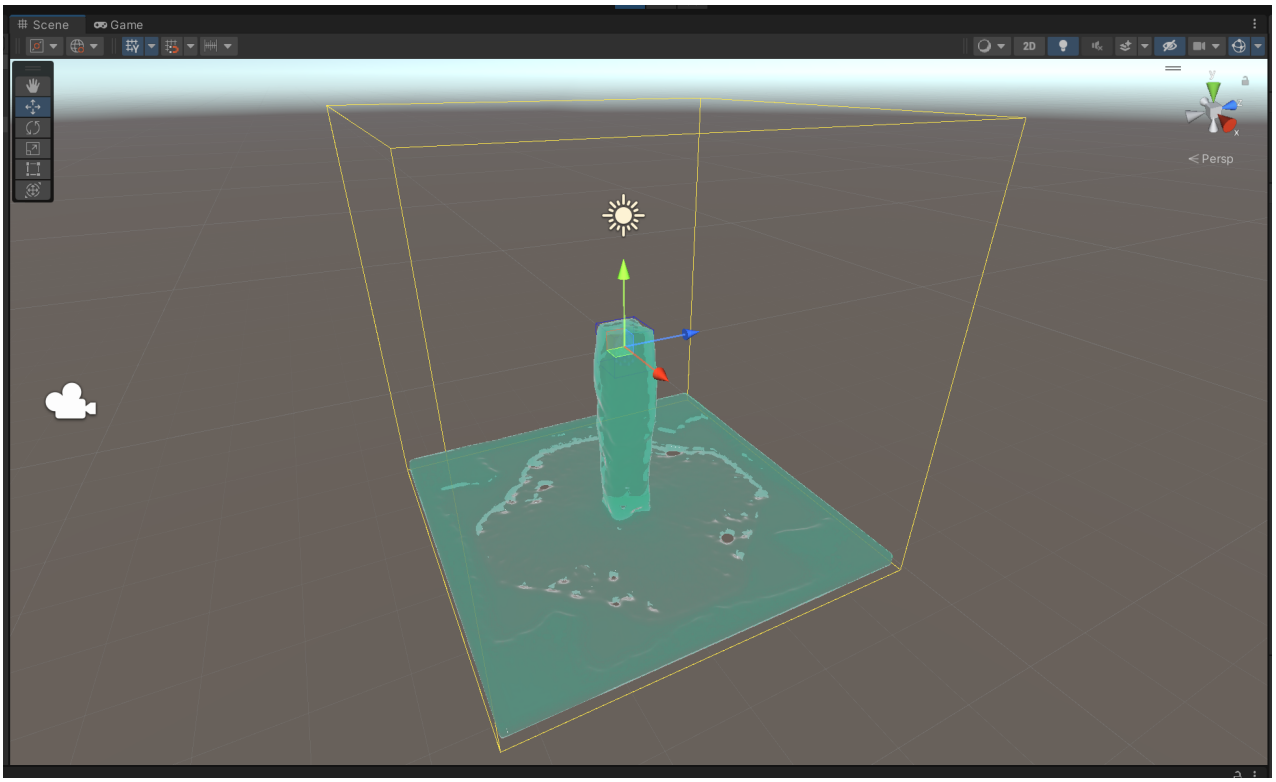
After setting the Custom Light parameter the error message will disappear:



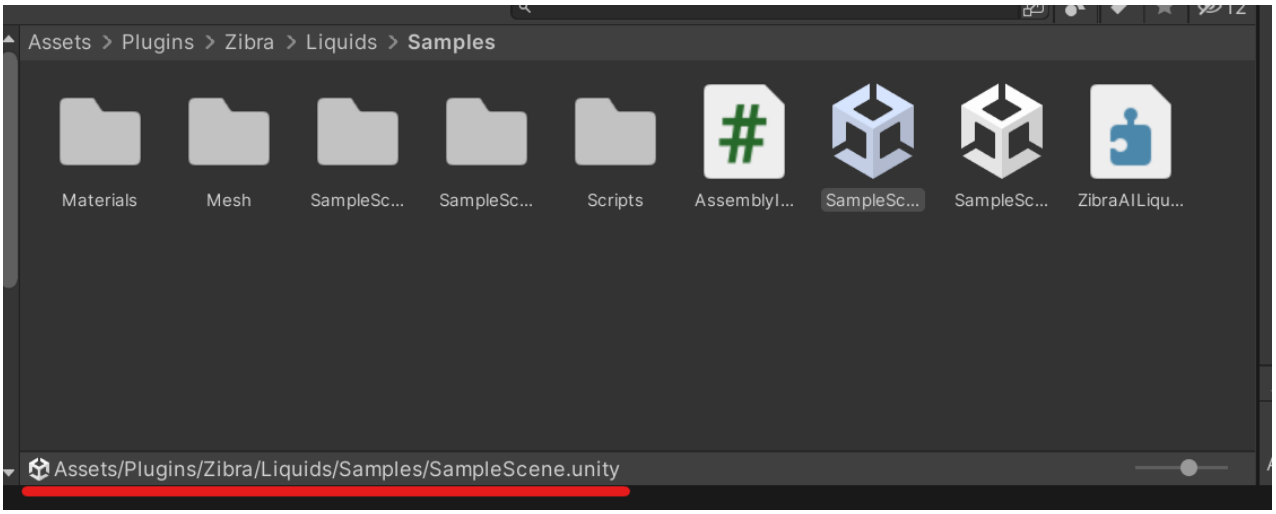
After creating Zibra Liquid

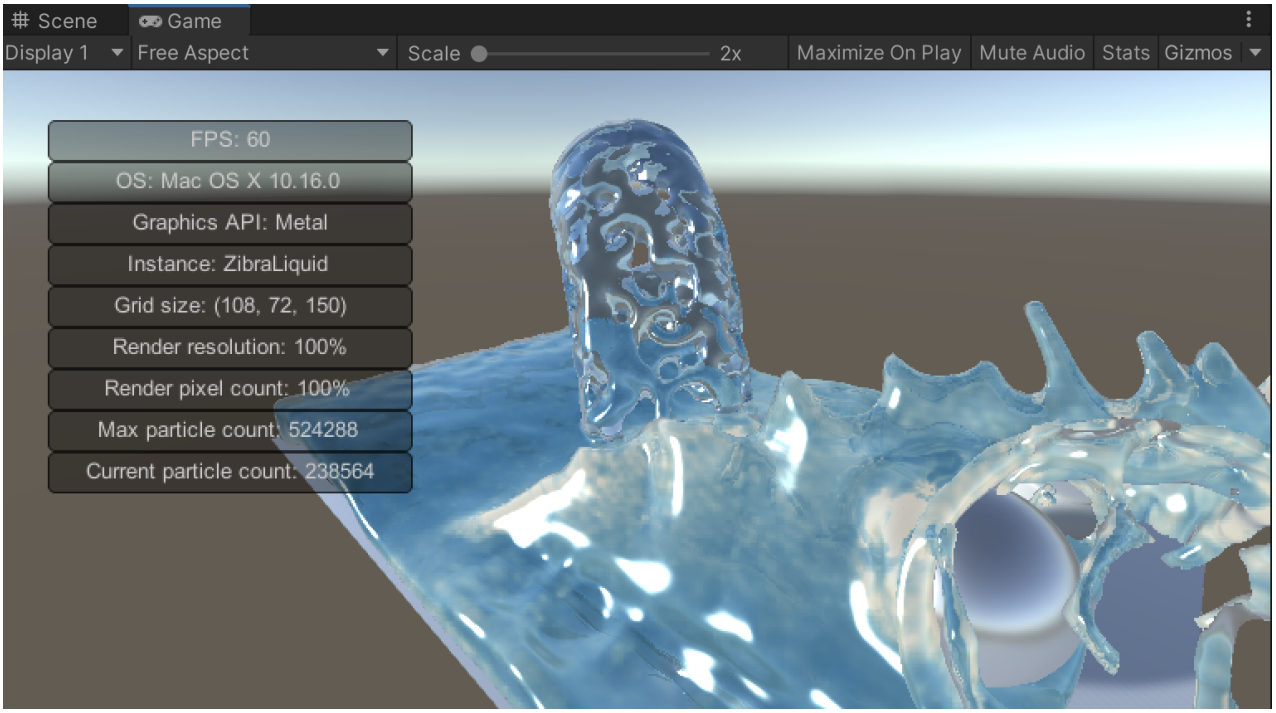
You can now press play, and see liquid



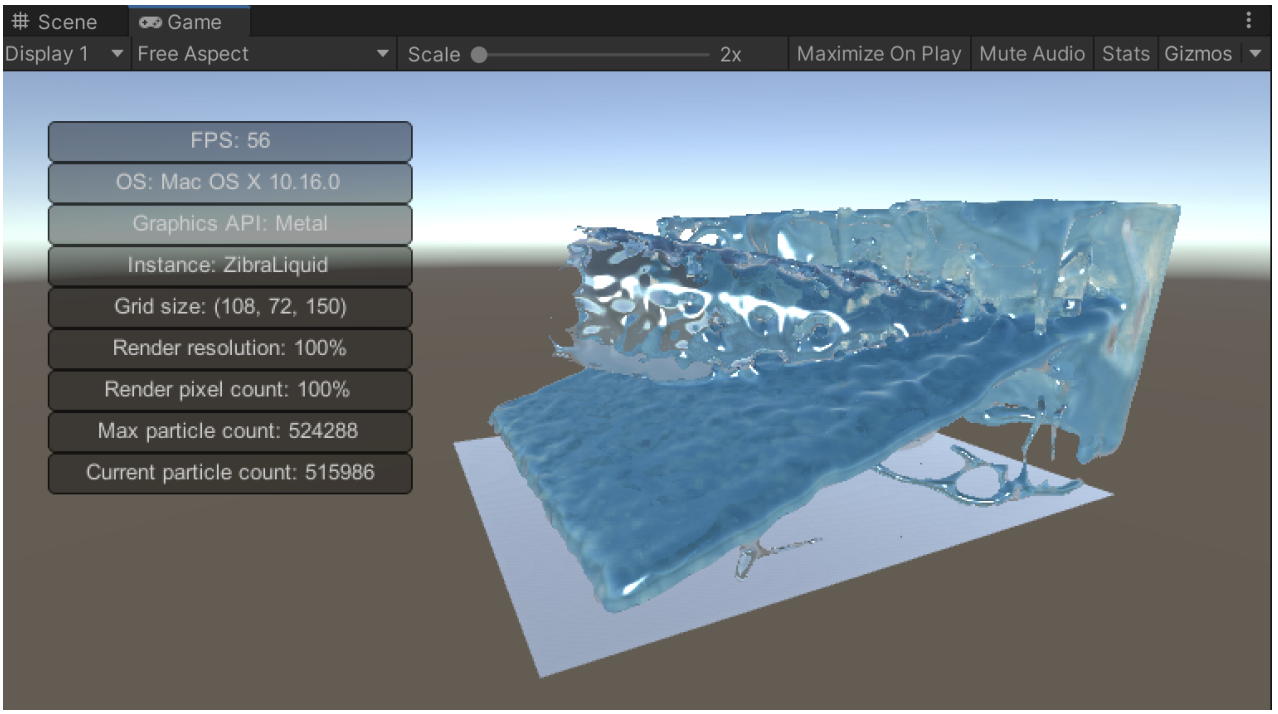


Also, you can check out how the fluid works in the built-in demo scene. To do that, open the Project window and go to *Assets -> Plugins -> Zibra -> Liquids -> Samples* and open the *SampleScene*.
Note that the sample scene is for SRP, and in URP/HDRP non liquid objects may look incorrectly.





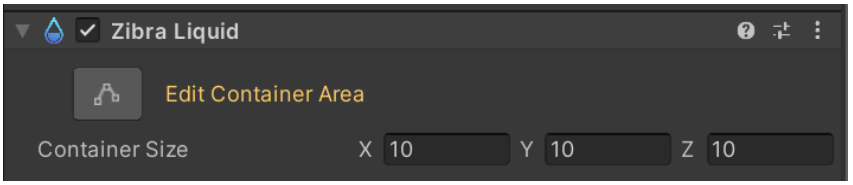
In the sample scene you can see various features of Zibra Liquid.
You can move the camera with *WASD*, control gravity with arrows, remove gravity with *O*, increase and decrease gravity with *Shift* and *Ctrl*



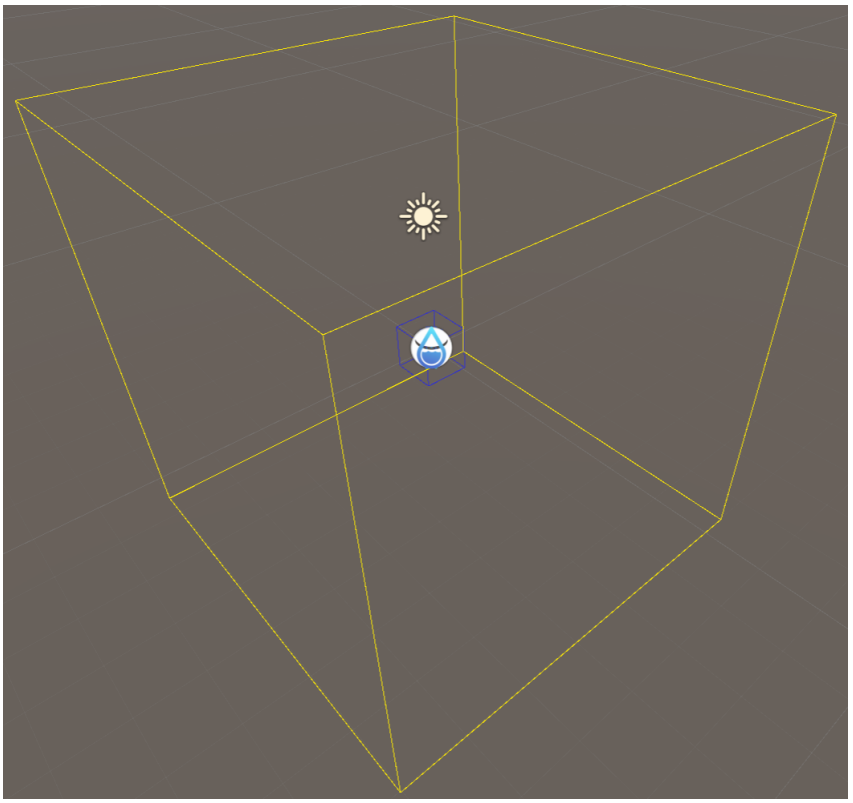
Configure Zibra Liquid

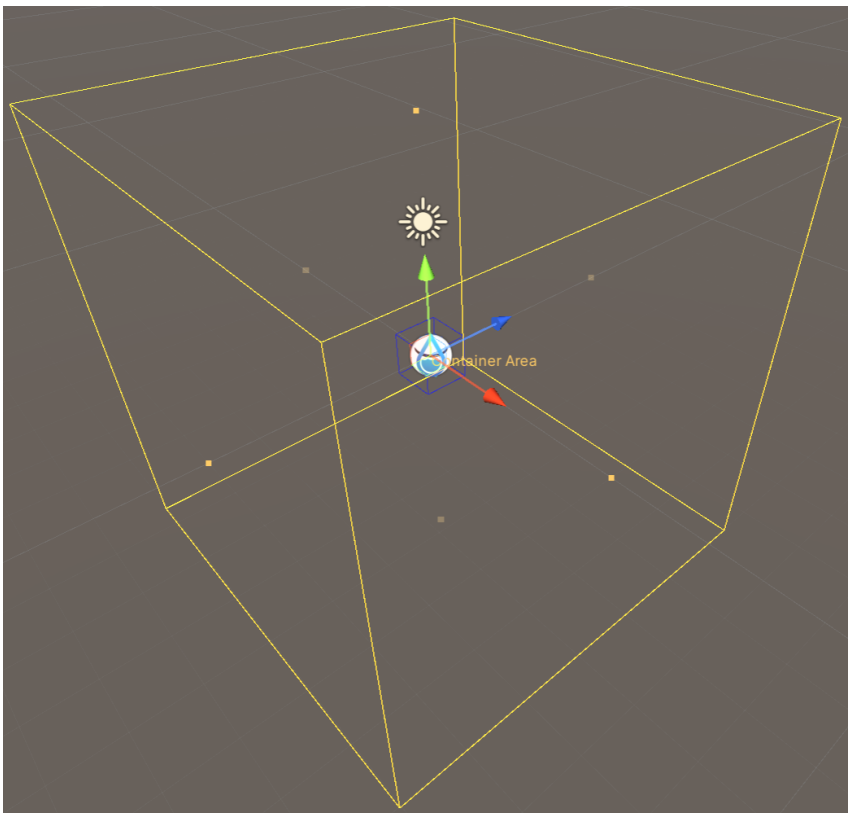
Main Liquid parameters

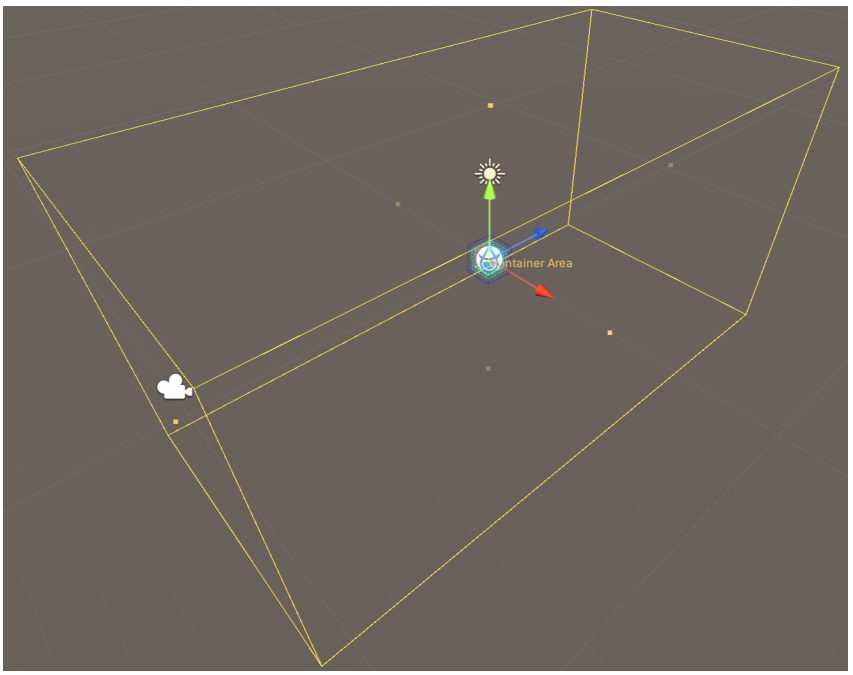
- **Container size**



The Container size parameter changes how large the simulation volume of liquid is.
You can press “Edit Container Area" to enable resizing gizmos.
This parameter cannot be changed on the “live” liquid instance.




ntainer Area


ntainer Area

- **Maximum allowed timestep**

This parameter specifies what the highest delta time that may be used in simulation is, when the FPS drops. If that limit is hit, liquid simulation will slow down.
Higher values correspond to less stable simulation during FPS drops, but will result in less cases when the liquid slows down.

- **Simulation speed**

This parameter controls the speed of the liquid simulation. You can slow down or speed up the liquid with this parameter.

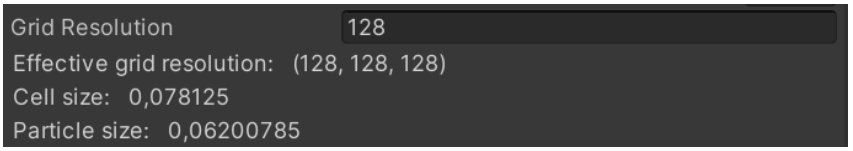
- **Iterations per frame**

Controls how many physic steps will be calculated each Update or FixedUpdate. This affects performance, especially on mobile, but results in better simulation quality. In most cases you will want to set it to 1.

- **Max particle count**

Controls how many liquid particles a liquid instance can have simultaneously. Higher values correspond to a higher possible liquid volume, but results in higher VRAM usage and potentially higher performance cost.
This parameter cannot be changed on the *“live”* liquid instance.

- **Grid resolution**



This parameter controls the size of the liquid simulation grid.
This is the most important parameter for performance adjustment.
Higher size of the grid corresponds to a higher quality simulation, and the smaller size of each particle, but results in higher VRAM usage and a higher performance cost.
Effective grid resolution shows you the size of your liquid grid. VRAM usage and performance cost scales with effective grid resolution.
Cell size and Particle size shows you calculated size of each cell and particle respectively.
This parameter cannot be changed on the *“live”* liquid instance.

- **Run simulation**

Freezes the liquid when disabled. Also decreases performance cost when disabled, since liquid physics won't be simulated.

- **Use Fixed Timestep**

Use Fixed Timestep for physics simulation. Can be used to have similar behavior of liquid between different runs and machines. Use with care, if the physics simulation takes more time than Fixed Timestep, performance will be severely affected.

- **Visualize Scene SDF**

Allows you to visualize analytical and neural colliders. This option is intended only for debugging.

- **Enable Render Downscale**

Enables the usage of lower resolution for liquid rendering to improve performance. The Resolution scale is controlled with the "Downscale Factor". Recommended for mobile devices.
You can not set it to 1.0, if you want full resolution you must disable render downscale.

- **Custom Light (HDRP only)**

Sets which light will be used for liquid shading. It's required that you set that parameter. Currently, only 1 light can contribute to liquid lighting on HDRP.

- **Reflection probe**

Sets reflection probe used for reflection calculation on the liquid. Must be set on HDRP, and it's strongly recommended to be set on SRP/URP.

- **Current Rendering Mode**

Selects which renderer is going to be used for this instance of the liquid. The default and recommended one is Mesh Render. Please note that only Unity Render supports VR in the current release.

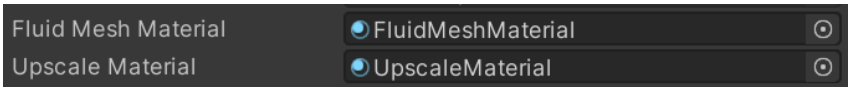
- **Current Injection Point**

Specify at which point in the render graph should liquid be rendered. Recommended options are: Before Forward Alpha if you want other transparent objects to be rendered on top of the liquid, or After Forward Alpha if you want liquid to be rendered on top of the other transparent objects.

Material parameters

Please note that all material parameters are only used in Mesh Render. In the case of Unity Render mode, you are responsible for material and its parameters.

- **Materials**



Materials used to shade liquid surface, and optionally upscale liquid when the Enable Render Downscale option is enabled. Most users will not change that option. To use it you will need to create your own custom version of FluidMeshMaterial, and potentially UpscaleMaterial as well. If it is set to None in Editor it will revert back to the default value.

- **Color**

Liquid color.

- **Reflection Color**

Tint for reflections on the water.

- **Emissive Color**

How much light does liquid emit. Unless it’s pure black, liquid will glow.

- **Roughness**

How rough is the liquid surface. Normally, for liquids, that parameter is really low.

- **Metalness**

How metallic (reflective) the liquid surface is.

- **Scattering amount**

Defines how much light gets scattered inside the liquid. Higher values correspond to more opaque murky liquid.

- **Absorption amount**

Defines how much light gets absorbed inside the liquid. Higher values correspond to more opaque liquid.

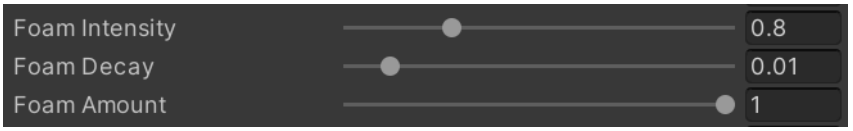
- **Index of Refraction**

Determines how refracted the light coming through the liquid will be.

- **Fluid Surface Blur**

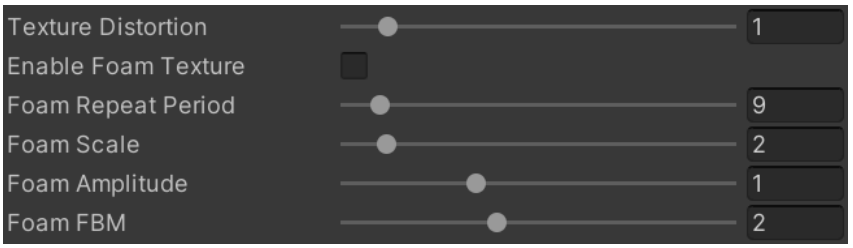
Determines how smooth the surface of liquid will appear.

- **Foam spawn/decay parameters**



Determine how fast foam will spawn and decay

- **Foam texture parameters**



Determine whether and how additional texture will be applied on top of foam to add details

- **Foam blurring**

Diffuses the foam with time

- **Additional materials**

Determine parameters of additional liquid materials that are used by foam and additional particle species
Foam uses Material 1

Advanced Material parameters

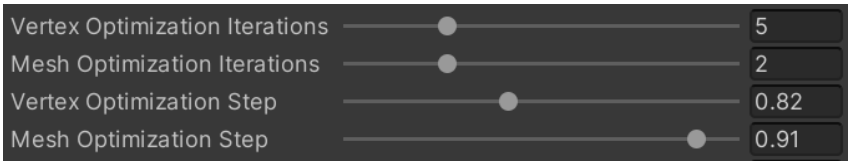
Please note that all material parameters are only used in Mesh Render. In the case of Unity Render mode, you are responsible for material and its parameters.

- **Ray Marching Resolution Downscale**

  Configures resolution of raymarching. Higher values correspond to higher quality refraction and Scattering/Absorption calculations, but also higher performance cost.
- **Refraction Bounces**

  Configures how many refraction bounces will be calculated. Two Bounces has higher quality, especially when underwater, but also has a higher performance impact.
- **Underwater Render**

  Enables underwater render, when the camera is underwater. Enabling this option has a slight performance impact even when the camera is not underwater.
- **Vertex/Mesh optimization parameters**



  Configures how much smoothing will be applied to the liquid surface. Higher iterations count corresponds to higher quality but has a slightly higher performance impact. Higher Step values correspond to more smoothing but potentially more artifacts. You can set all those values to 0 to get voxel liquid.

- **Dual Contouring Iso Surface Level**

  Configures how thick the surface of liquid will be before mesh optimization. Lower values generally result in higher triangle count and so higher quality.
- **Iso Surface Level**

  Configures how thick the surface of liquid will be after mesh optimization. If this option differs too much compared to the previous one you may have some visual artifacts, e.g. bridging between liquid drops.

- **Raymarching parameters**

Configures how liquid is raymarched for refraction/liquid depth calculations. If misconfigured it can result in opaque liquid or bad refraction. Higher values of Iso Surface parameter correspond to thicker liquid surface for the purpose of that calculation. Higher Max Steps count corresponds to higher quality and less artifacts but has higher performance impact. Higher values of Step Size and Step Factor allow you to decrease Max Steps parameters, but corresponds to more artifacts.

Solver parameters

- **Gravity**

Gravity that affects the entire liquid. If you want radial gravity use Force Field instead.

- **Fluid stiffness**

Configures how stiff the liquid is (how resistant it is to compression). Higher values are not recommended and may lead to unstable simulation.

- **Particle density**

Configures the resting particle density. Higher values result in smaller particles, which result in higher quality, but smaller liquid volume.

- **Viscosity**

Configures how viscous the liquid is (how hard it is to change the shape of the liquid)

- **Surface tension**

Configures the surface tension of the liquid. Positive values result in liquid that tries to merge together. Negative values result in liquid that tries to split into as many small liquid parts as possible, resulting in a spaghettification effect.

- **Velocity limits**

Limits the maximum/minimum velocity of the particles. Setting minimum velocity to values higher than 0 results in liquid that will never stop.

- **Force interaction strength**

Scales the force that the liquid applies to objects with force interaction enabled. This has a logarithmic scale.

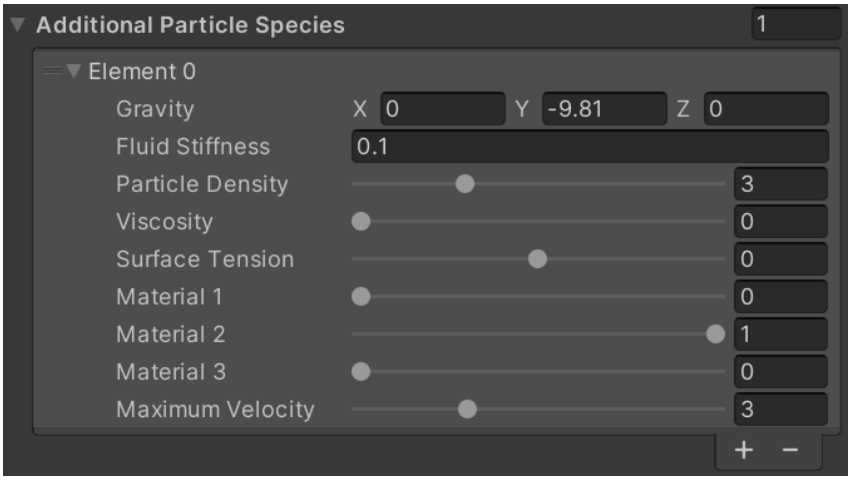
- **Foam Buoyancy**

Determines buoyancy of foam compared to normal liquid
Value of 0 represents same buoyancy between foam and liquid

- **Material indices**

Determines which material will be used for liquid rendering by setting corresponding weights. Default material's weight is equal to 1 - sum of all other weights. You can select a single material by setting its value to 1, or you can mix multiple materials by setting different weights for each that sums up to 1.

- **Additional Particle Species**



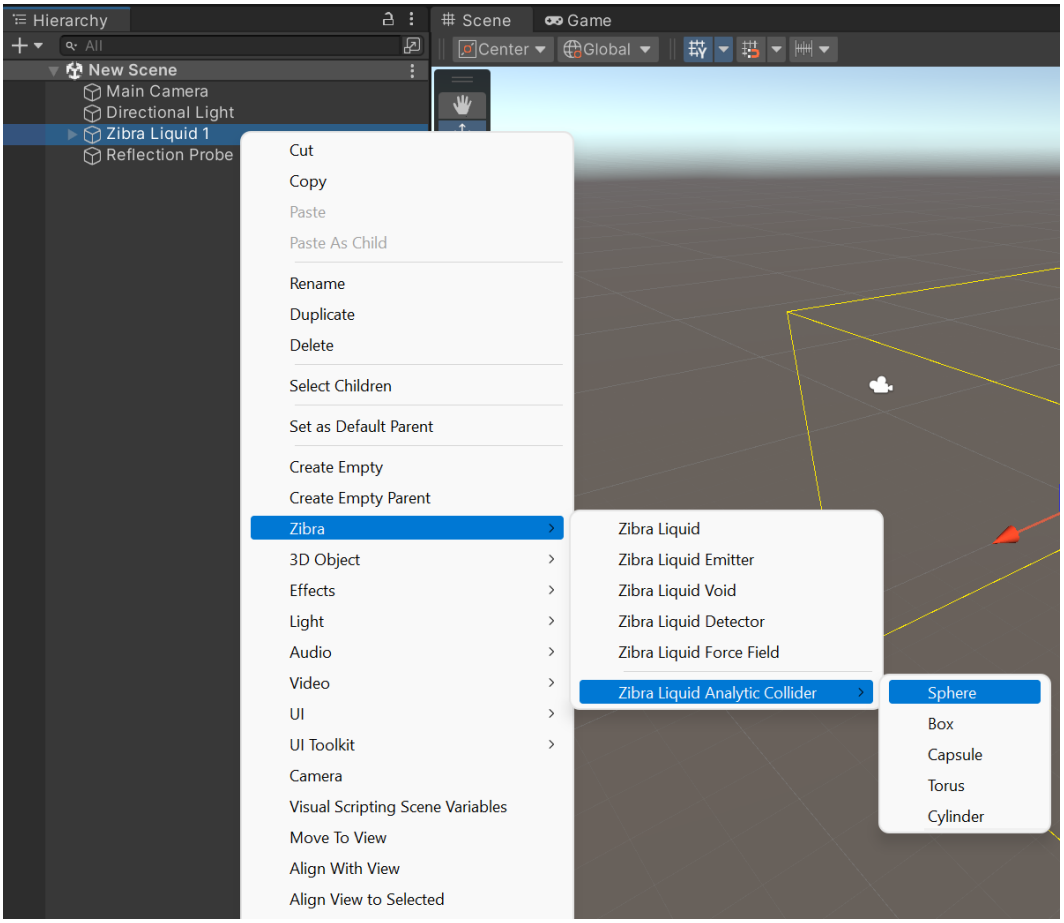
Defines the number of additional particle species and their parameters. You can add additional particle species in runtime, and increase the number of particle species up to 16 over what it was when liquid started simulation.

Colliders

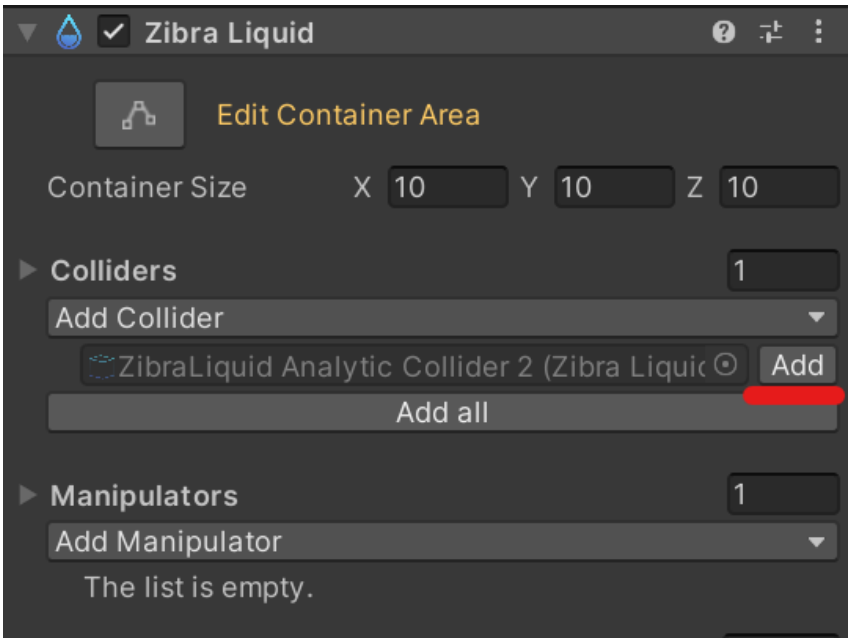
Zibra Liquid has 3 types of colliders: Analytic, Neural and Skinned Mesh colliders.
Analytic colliders are simple shapes: cubes, spheres, capsules, torus, cylinders.
Neural colliders are created from arbitrary static mesh.
Skinned Mesh colliders are created from skinned mesh, and implemented as group of neural colliders, one for each bone.

Creating Analytic Colliders GameObjects

1. Right click in the Hierarchy window, and select Zibra -> Zibra Liquid Analytic Collider -> *Your preferred shape*
   Please note, if you right clicked specifically on your Zibra Liquid object, you can skip step 2



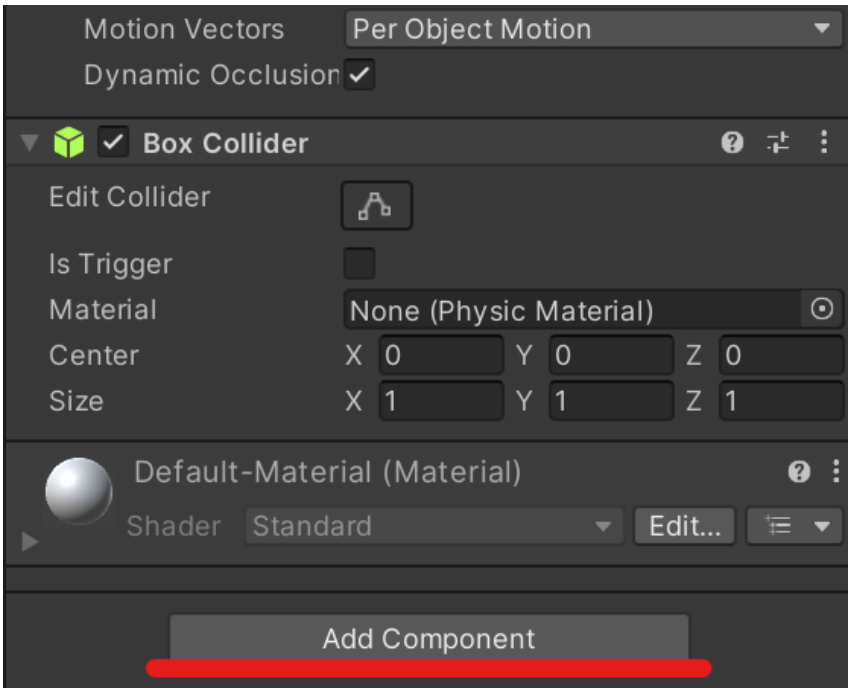
2. Open Inspector for your Liquid object, and press the Add button in the list of colliders to add. You can also use the Add all button to add all colliders to your liquid instance.
   You may also need to expand the list of colliders to add by pressing the “Add Collider” button first.

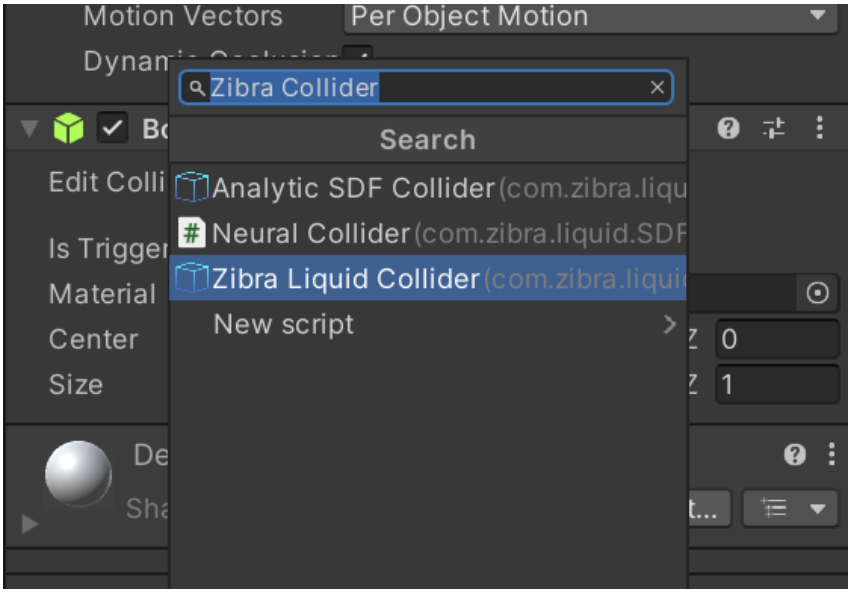


After that your collider is ready to be used. You can set its transform in the editor, or move it at runtime.

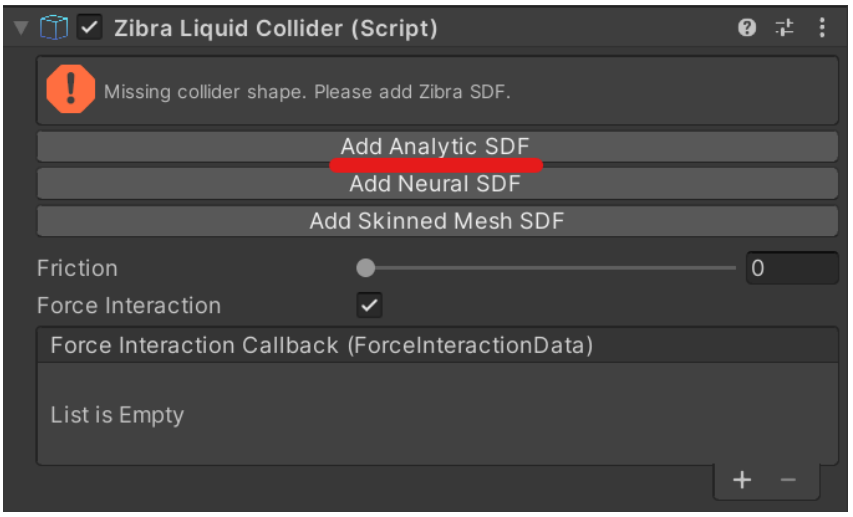
Adding Analytic Colliders to existing GameObjects

1. Open inspector for your selected GameObject. Here we'll use Unity's default Cube GameObject. Press Add Component and select Zibra Liquid Collider.

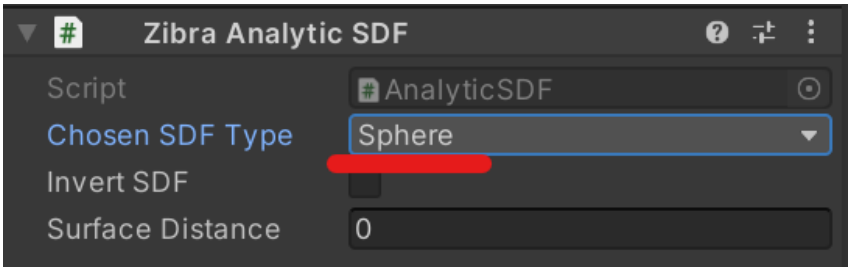




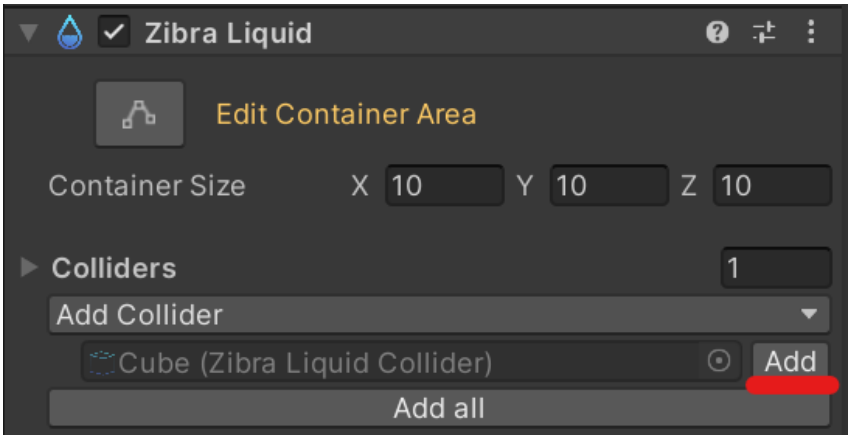
2. In Zibra Liquid Collider Inspector press Add Analytic SDF.



3. Select shape for your collider in Zibra Analytic SDF Inspector.



4. Open Inspector for your liquid object, and press the Add button in the list of colliders to add. You can also use the Add all button to add all colliders to your liquid instance.
   You may also need to expand the list of colliders to add by pressing the “Add Collider” button first.

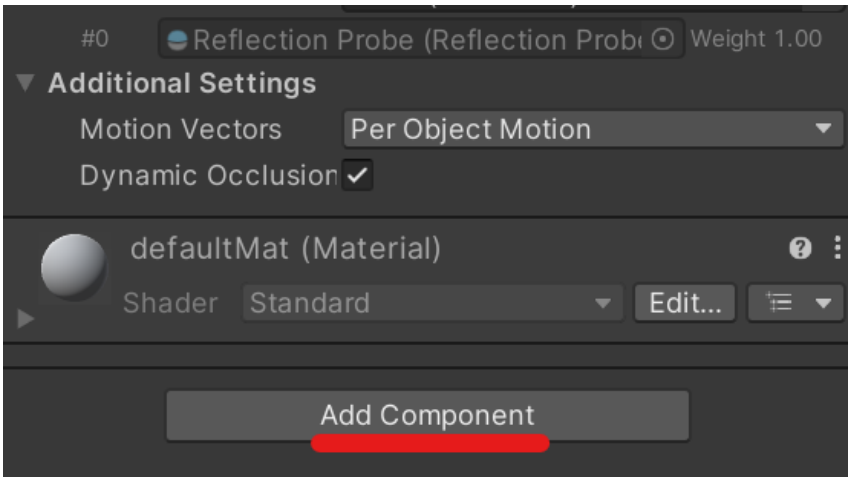


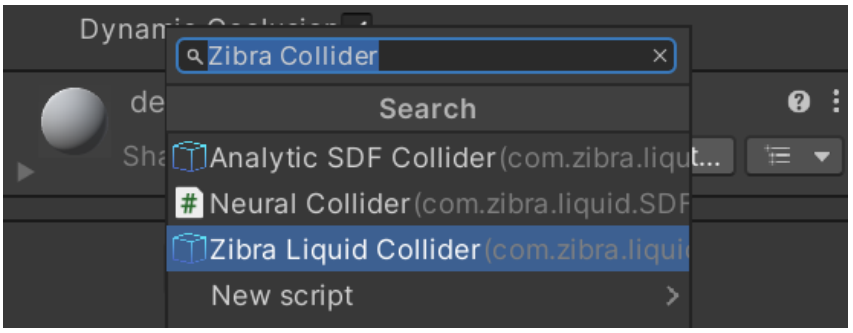
After that your collider is ready to be used. You can set its transform in the editor, or move it at runtime.

Adding Neural Colliders

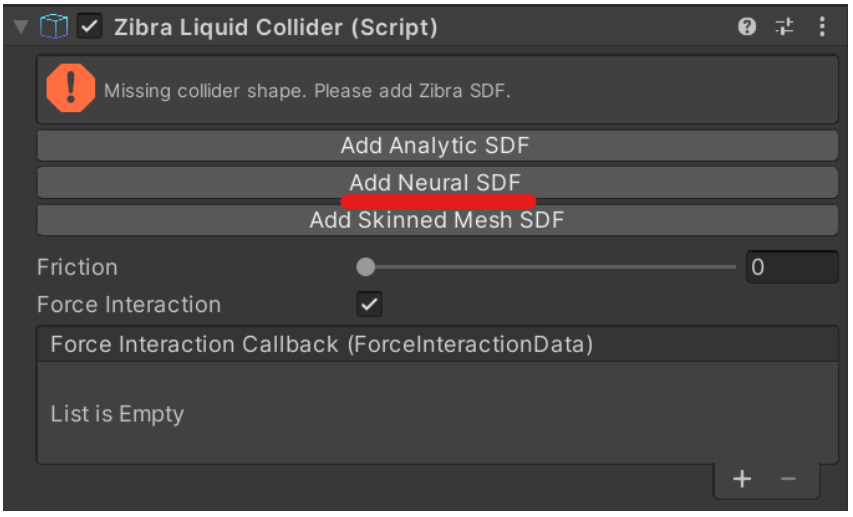
Before you can use the Neural Colliders functionality you need to register the plugin: [Registering Zibra Liquids]
To add a neural collider you first need to have the mesh that you want the water to collide with.

1. Open inspector for your selected GameObject. Press Add Component and select Zibra Liquid Collider.





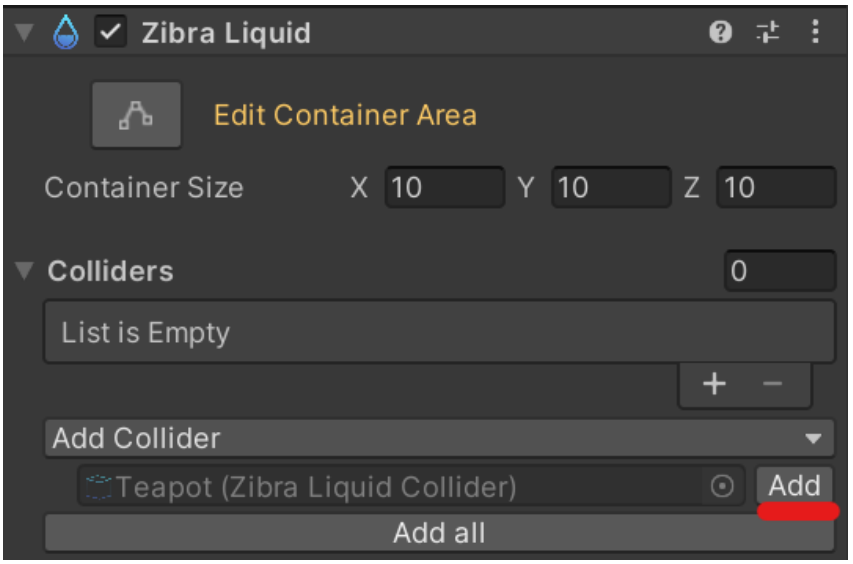
2. In the newly created Zibra Liquid Collider's Inspector press Add Neural SDF.



3. In the newly created Zibra Neural SDF's Inspector press Generate SDF. This will send your mesh to ZibraAI servers for processing. Server will generate data required for the neural network to reconstruct the SDF of your mesh. After processing, you'll see "SDF already generated." message.

If you don't see that button, you probably haven't registered your plugin yet. Register your plugin: Registering Zibra Liquids and try again.

4. Open Inspector for your Liquid object, and press the Add button in the list of colliders to add. You can also use the Add all button to add all colliders to your liquid instance.
You may also need to expand the list of colliders to add by pressing the "Add Collider" button first.

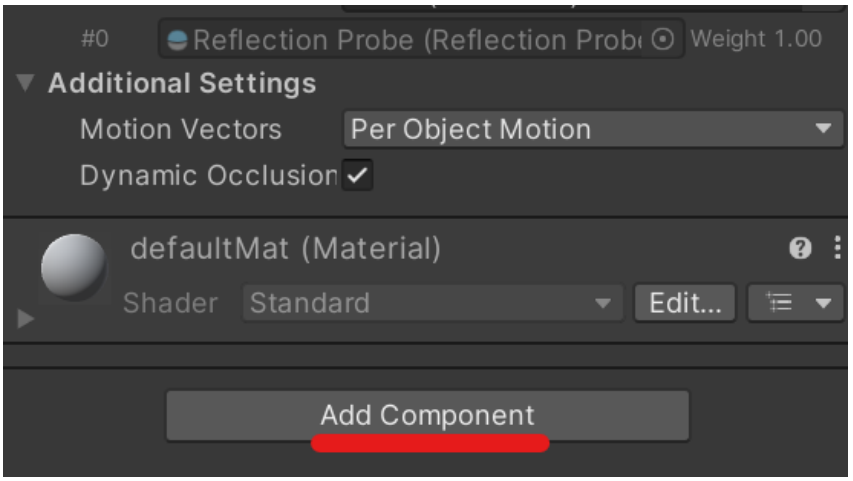


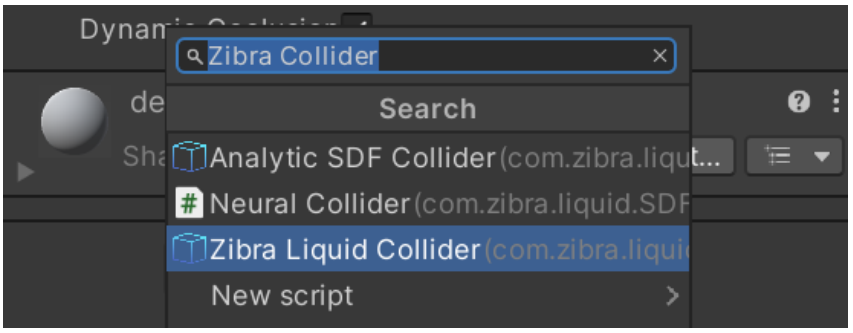
Now your collider is ready to be used. You can set its transform in the editor, or move it at runtime.

Adding Skinned Mesh Colliders

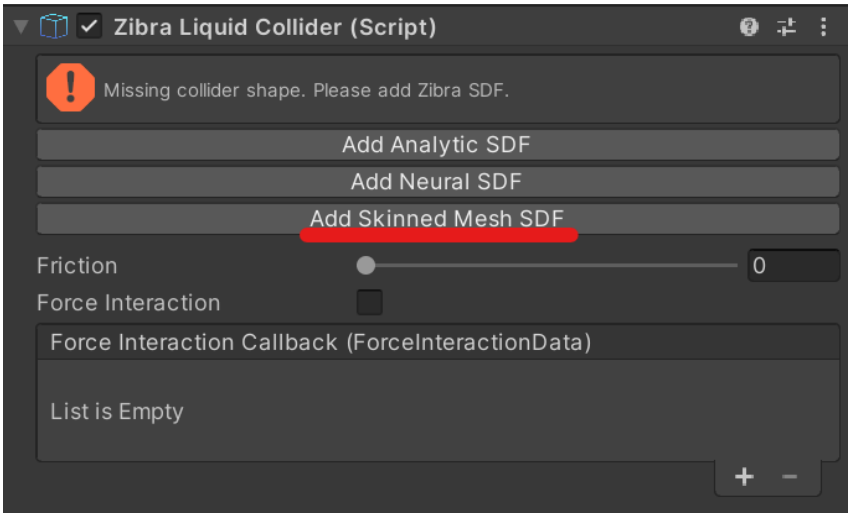
Before you can use the Skinned Mesh Colliders functionality you need to register the plugin: [Registering Zibra Liquids]
To add a skinned mesh collider you first need to have the Skinned Mesh Renderer that you want the water to collide with.

1. Open inspector for your selected GameObject. Press Add Component and select Zibra Liquid Collider.

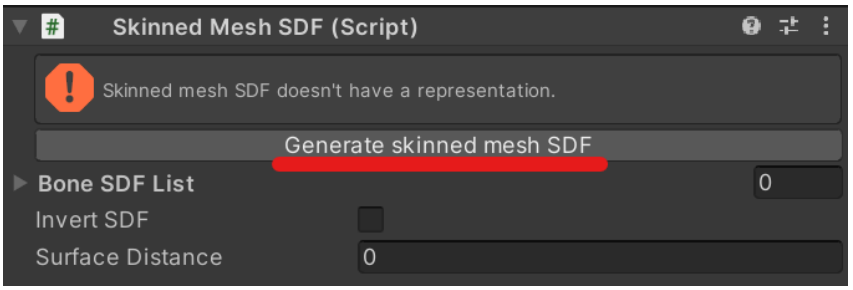




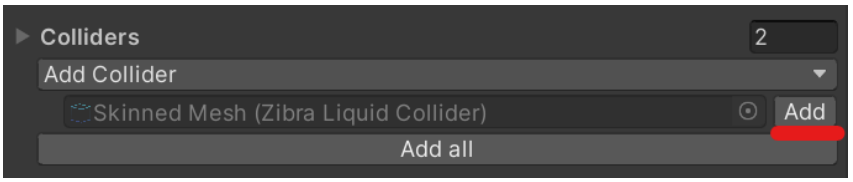
2. In the newly created Zibra Liquid Collider's Inspector press Add Skinned Mesh SDF.



3. In the newly created Skinned Mesh SDF's Inspector press Generate skinned mesh SDF.
   This will send your mesh to ZibraAI servers for processing. Server will generate data required for the neural network to reconstruct the SDF of your mesh. After processing, you'll see "SDF already generated." message.



   If you don't see that button, you probably haven't registered your plugin yet. Register your plugin: Registering Zibra Liquids and try again.
4. Open Inspector for your Liquid object, and press the Add button in the list of colliders to add. You can also use the Add all button to add all colliders to your liquid instance.
   You may also need to expand the list of colliders to add by pressing the "Add Collider" button first.



Now your collider is ready to be used. You can set its transform in the editor, or move it at runtime.

Collider parameters

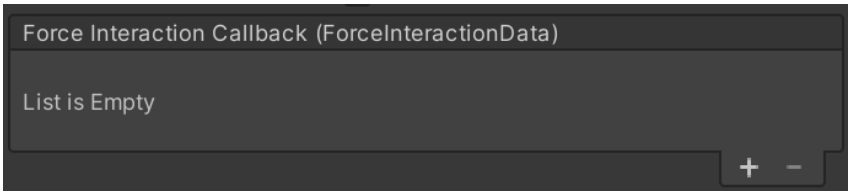
- **Friction**

Controls the surface friction of the collider, used for liquid interaction.

- **Force Interaction**

Controls whether liquid can apply force back to the object. You need to add a rigidbody component to the object that has a collider for liquid to apply force to.

- **Force Interaction Callback**



Specifies a callback that gets Force Interactions force and can modify it. Will be called even when Force Interaction is disabled, with correct calculated force, but that force won’t be applied in this case.

Manipulators

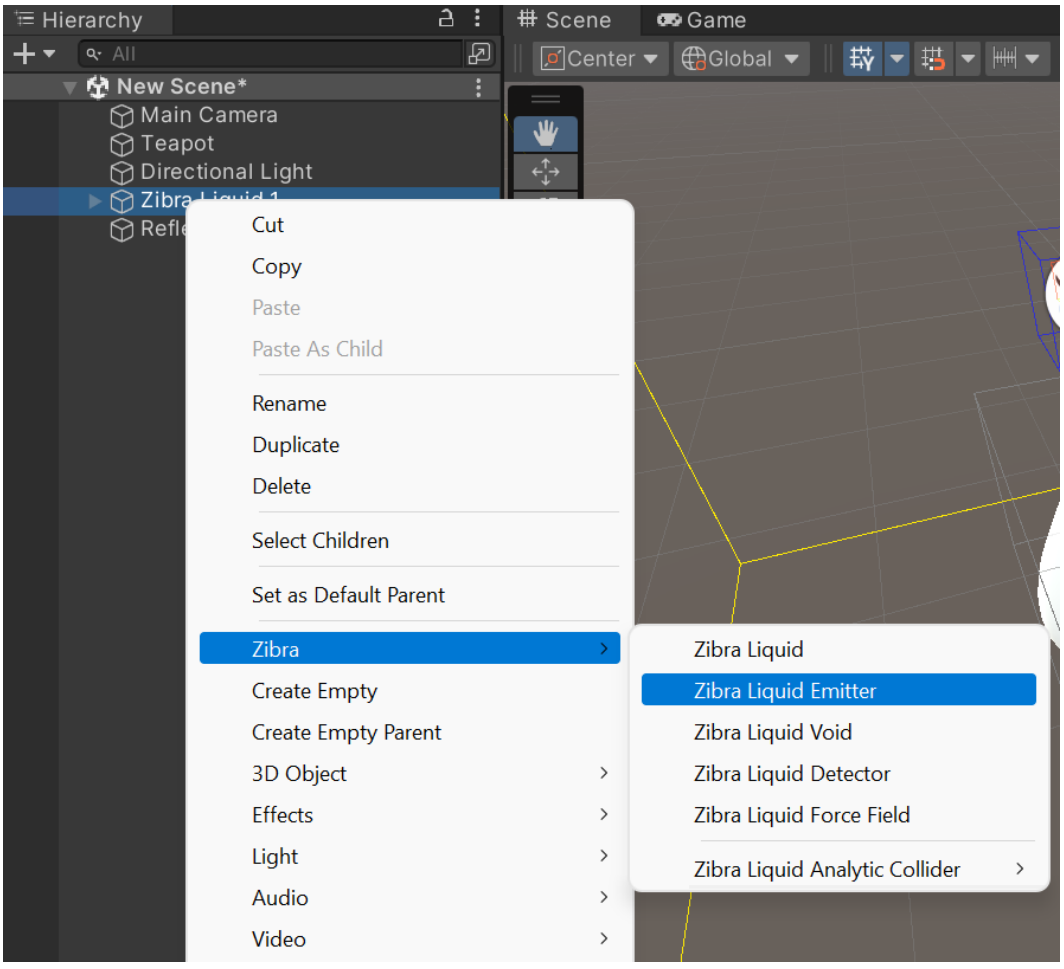
Zibra Liquid has many kinds of manipulators: Emitter, Void, Detector, Force Field

Emitter

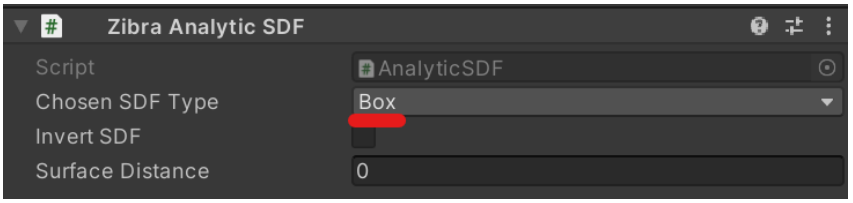
Liquid emitters are used for emitting liquid, if you want to spawn liquid you need these. When you first create Zibra Liquid, the Emitter is automatically created and added to it. If you don't need additional ones you can skip adding additional emitters and go right to Emitter parameters

Adding Emitter

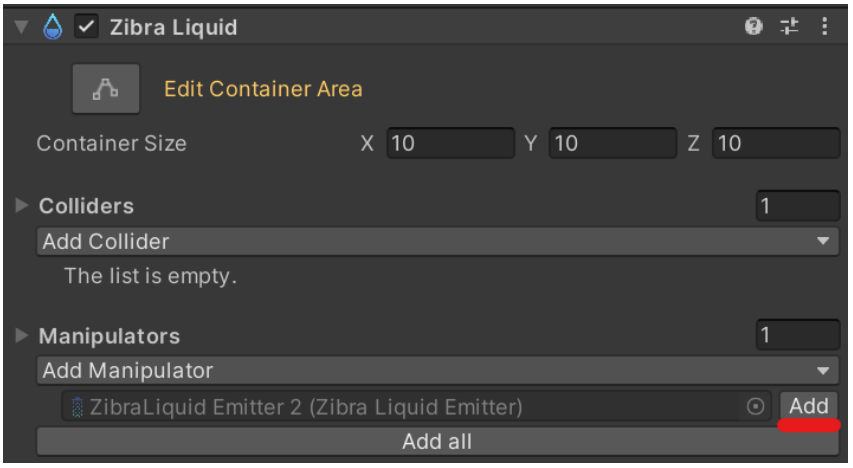
1. Right click in Hierarchy, and select Zibra -> Zibra Liquid Emitter
   Note that if you right click specifically on your liquid object you can skip step 3



2. (optional) Change shape of your emitter in Zibra Analytic SDFs Inspector in emitters GameObject.

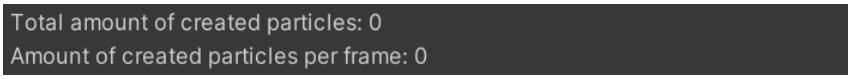


3. Open Inspector for your liquid object, and press the Add button in the list of manipulators to add. You can also use the Add all button to add all manipulators to your liquid instance.
   You may also need to expand the list of colliders to add by pressing the “Add Manipulators” button first.



Emitter parameters

- **Emitter statistics *(read only)***



  When you start the play mode you’ll be able to keep track of how many particles were emitted. You can also access those values via scripts, via *“CreatedParticlesTotal”* and *“CreatedParticlesPerFrame”* attributes.
- **Volume Per Sim Time**

  Amount of liquid that is emitted per unit of Simulation Time.
- **Initial Velocity**

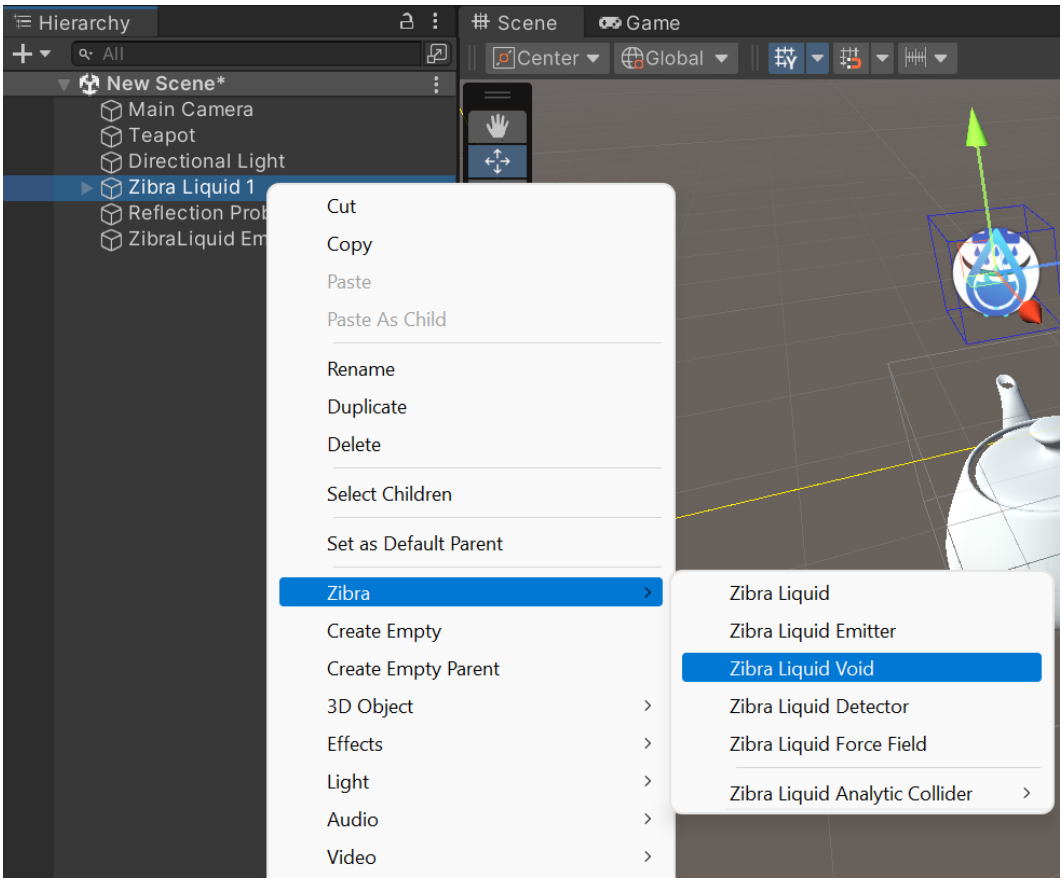
  Initial speed of the emitted particles.

Void

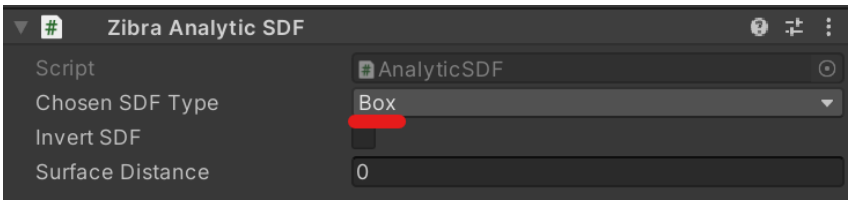
Voids are used to destroy liquid. Since there's a limit to the maximum number of particles in Zibra Liquid instance, you need to destroy particles for your emitters to keep emitting liquid continuously. But if it's fine for you that the emitters will stop at some point, the usage of voids is optional.

Adding Void

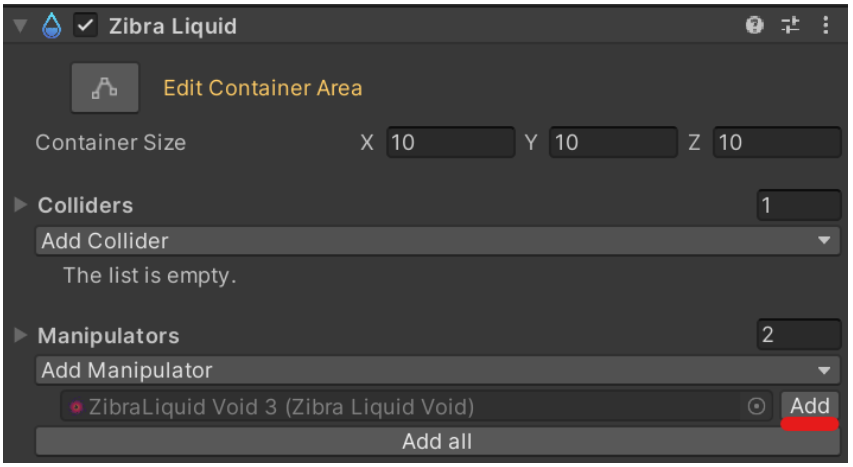
1. Right click in Hierarchy, and select Zibra -> Zibra Liquid Void
   Note that if you right click specifically on your liquid object you can skip step 3



2. (optional) Change shape of your void in Zibra Analytic SDFs Inspector in emitters GameObject.



3. Open Inspector for your liquid object, and press the Add button in the list of manipulators to add. You can also use the Add all button to add all manipulators to your liquid instance.
   You may also need to expand the list of colliders to add by pressing the “Add Manipulators” button first.



Void parameters

- **Void statistics *(readonly)***

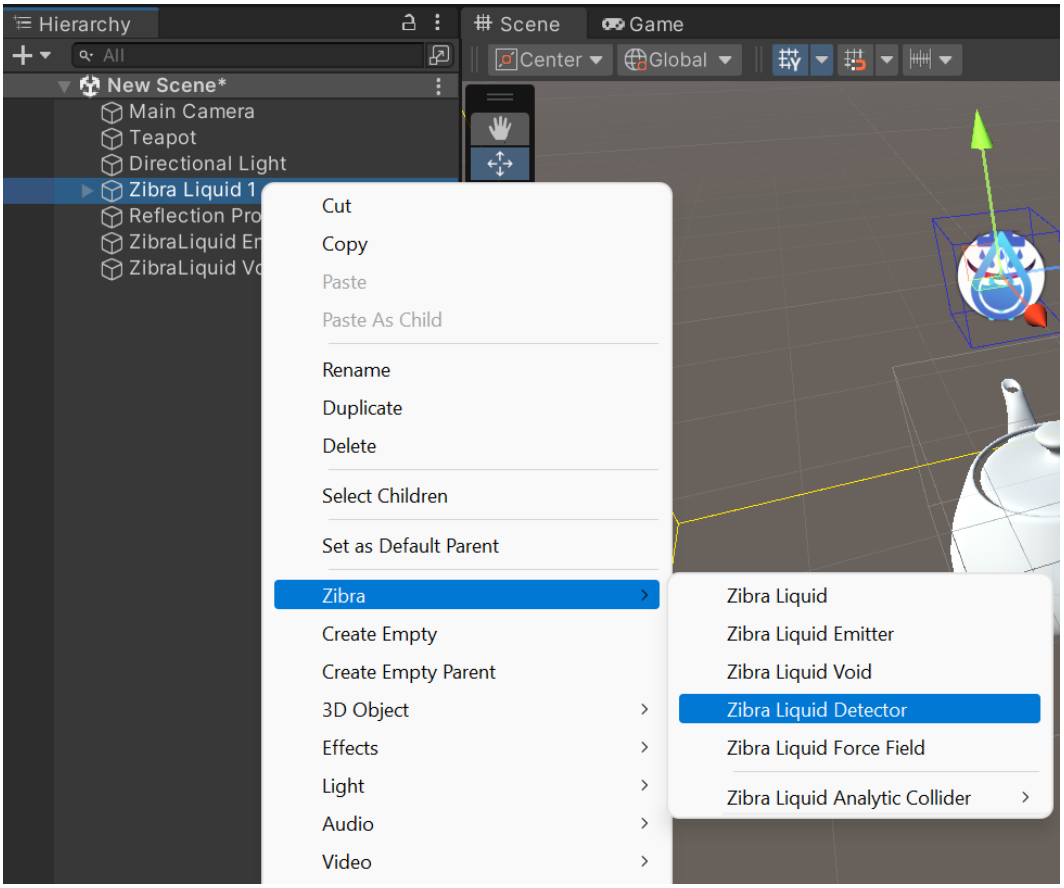
When you start play mode you’ll be able to keep track of how many particles were deleted. You can also access those values via scripts, via *“DeletedParticleCountTotal”* and *“DeletedParticleCountPerFrame”* attributes.

Detector

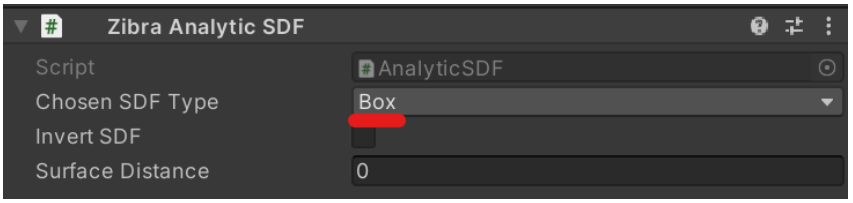
Detector is used to detect how many liquid particles are in the specified volume.

Adding Detector

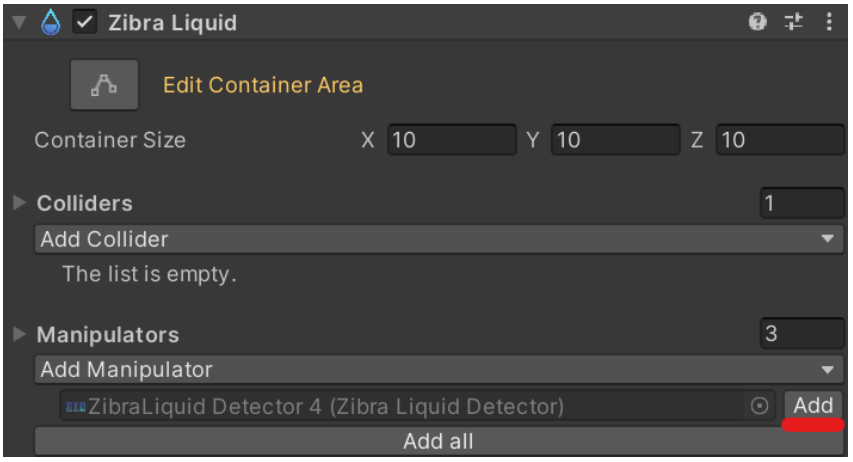
1. Right click in Hierarchy, and select Zibra -> Zibra Liquid Detector
   Note that if you right click specifically on your liquid object you can skip step 3



2. (optional) Change shape of your detector in Zibra Analytic SDFs Inspector in emitters GameObject.

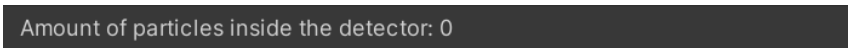


3. Open Inspector for your liquid object, and press the Add button in the list of manipulators to add. You can also use the Add all button to add all manipulators to your liquid instance.
   You may also need to expand the list of colliders to add by pressing the “Add Manipulators” button first.



Detector parameters

- Detected particle count (read only)



  When you start play mode you’ll be able to keep track of how many particles are inside the detector. You can also access those values via scripts, via the “ParticlesInside” attribute.

- Bounding box (read only)

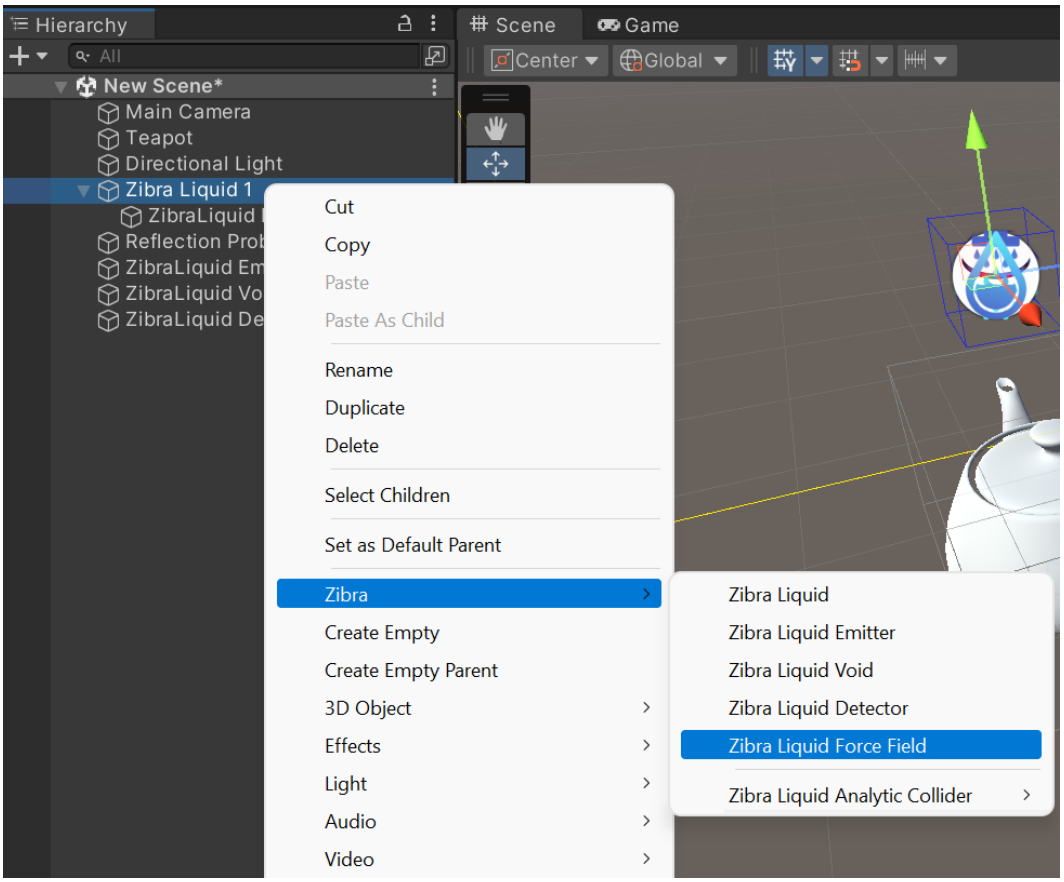
  When you start play mode you’ll be able to keep track of the bounding box of detected particles. You can also access those values via scripts, via the “BoundingBoxMin”/”BoundingBoxMax” attribute.

Force Field

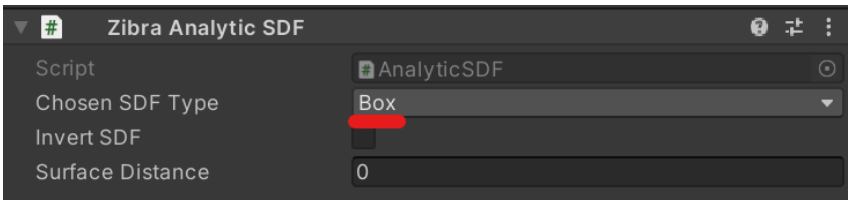
Force fields are used to apply force to liquid in a specific way, and in a specific region. You can for example use force fields to create radial gravity, to use instead of the default vector one. Or you can create a fountain by putting a force field that pushes liquid up in the body of water.

Adding Force Field

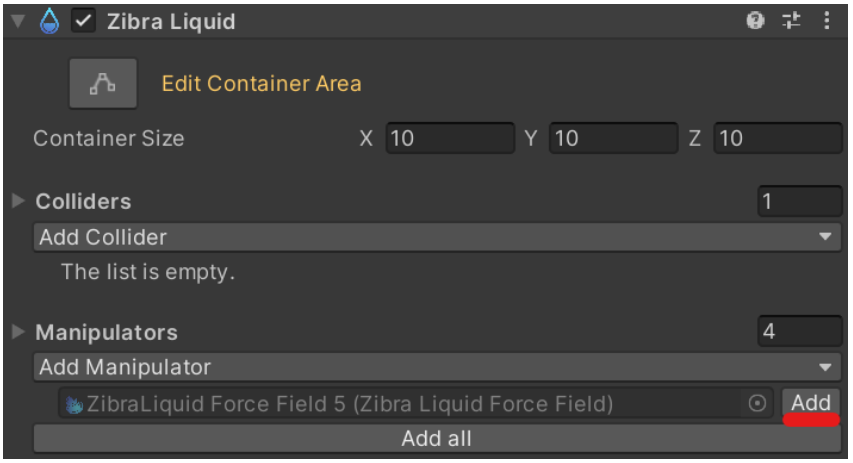
1. Right click in Hierarchy, and select Zibra -> Zibra Liquid Force Field
   Note that if you right click specifically on your liquid object you can skip step 3



2. (optional) Change shape of your force field in Zibra Analytic SDFs Inspector in emitters GameObject.



3. Open Inspector for your liquid object, and press the Add button in the list of manipulators to add. You can also use the Add all button to add all manipulators to your liquid instance.
   You may also need to expand the list of colliders to add by pressing the “Add Manipulators” button first.



Force Field parameters

- **Type**

Configures how Force Field applies force to liquid.
Radial - pushes or pulls liquid towards the Force Field.
Directional - pushes liquid in a specified direction.
Swirl - creates a whirlpool force around the Force Field.
In the case of the Directional and Radial settings, the direction can be adjusted by rotating the Force Field.

- **Strength**

Configures how strong the strength of the force applied by Force Field will be. Can be negative to reverse direction. (For radial type, positive force is pulling, and negative is pushing)

- **Distance decay**

Configures how fast the force strength decays with the distance from the Force Field.

- **Distance Offset**

Offset for distance to surface. That offset will be applied for purposes of calculating Distance decay.

- **Disable Force Inside**

Enable to disable applying force to the liquid inside force field.

- **Force Direction**

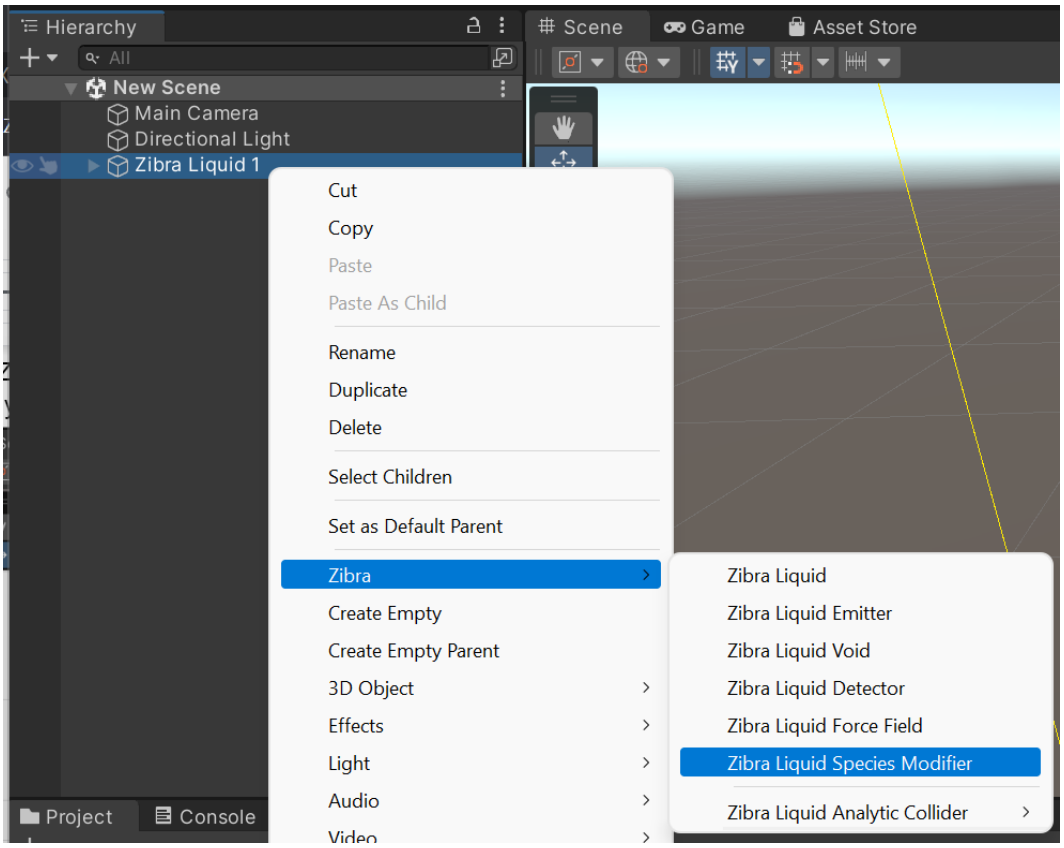
Direction of the force applied by force field. Only used if Force Field’s Type is Directional.

Species Modifier

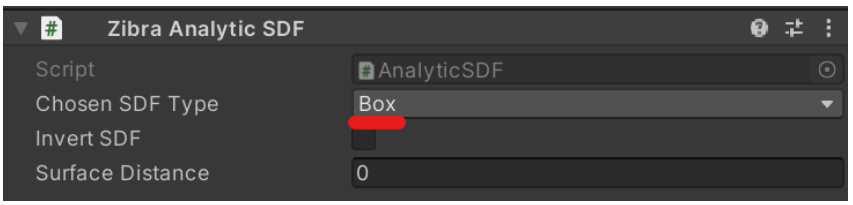
Species modifiers are used to change species of liquid particles. You can for example use it to change color of the liquid, invert gravity, or change any other parameter that can be configured per species.

Adding Species Modifier

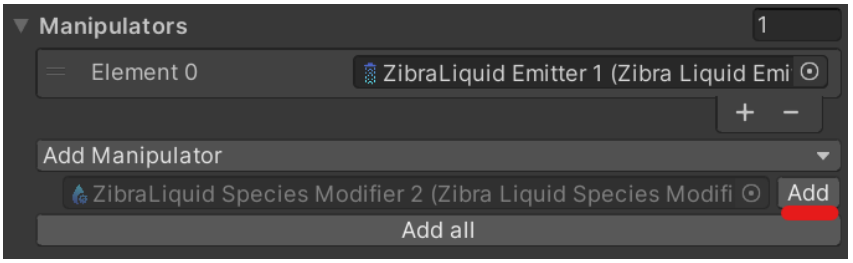
1. Right click in Hierarchy, and select Zibra -> Zibra Liquid Force Field
   Note that if you right click specifically on your liquid object you can skip step 3



2. (optional) Change shape of your force field in Zibra Analytic SDFs Inspector in emitters GameObject.



3. Open Inspector for your liquid object, and press the Add button in the list of manipulators to add. You can also use the Add all button to add all manipulators to your liquid instance.
   You may also need to expand the list of colliders to add by pressing the “Add Manipulators” button first.



Species Modifier parameters

- **Target Species**

  Defines which species the particle will get after modification.
- **Probability**

  Defines probability that a particle inside a species modifier will change its species each frame.

Common manipulator parameters

Particle species parameters are shared between all manipulator types. They are used to filter which particle species can interact with each manipulator.

- **Current Interaction Mode**

  Defines which particles can be affected by manipulator
  All Particle Species - Manipulator affects everything
  Only Selected Particle Species - Only affects particles of species specified in Target Species parameter
  Except Selected Particle Species - Affects all particles except species specified in Target Species parameter

- **Particle Species**

  Selects particle species for use with Current Interaction Mode parameter

SDFs

Each collider or manipulator has an SDF component that defines its shape.
You can switch any SDF component for any other, e.g. you can create a force field object, and then switch from analytic SDF to neural SDF.
Those SDF components also have some parameters you can change.

SDF parameters

- **Chosen SDF type (Analytic SDF only)**

  Defines which shape SDF has.
- **Invert SDF**

  Inverts your shape. For example inverted box will include all points in space, except those that are inside box
- **Surface Distance**

  Offset for distance to surface. With this parameter you can make the SDF a little bit “smaller” or “bigger”.

To adjust size, position and rotation of an SDF, adjust the transform of the GameObject containing corresponding SDF.

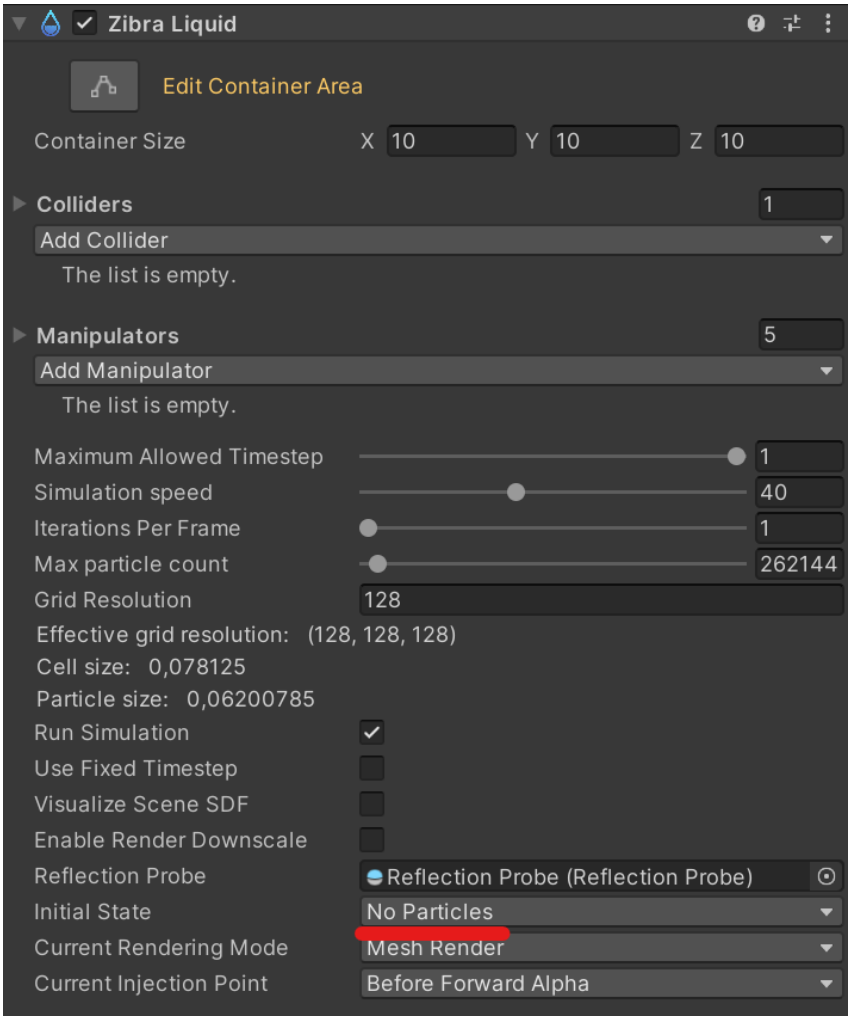
Liquid Initial State Baking

If you want to have a Zibra Liquid instance pre-filled on startup you will need to use Initial State baking.
After baking, the Zibra Liquid instance will remember its initial state and will start simulation from it. Note that we do not support baking animation, simulation over time, etc. Simulation itself is always performed on the device in real time, only the initial state is baked.
Initial State Baking will simulate the first N seconds of your liquid instance, save it, and simulation in play mode will start from that point.
Note that after changing some parameters you'll have to rebake your liquid, specifically: Grid Resolution, Container Size (in case you changed the size non-uniformly, and as a result changed the Effective grid resolution), Max particle count (in case your baked state has more particles that the current max number).

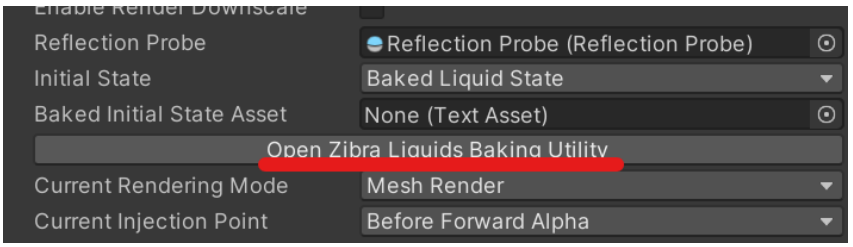
To bake initial state:

1. If you haven't already, add all emitters, voids, and colliders that you are going to have to your liquid.
2. Open the baking utility and select your Zibra Liquid instance to bake.
   There are 2 options to do it:
   2.1. Via the Zibra Liquid Inspector

2.1.1. Open the Zibra Liquid Inspector and set Initial State to “Baked Liquid State”.

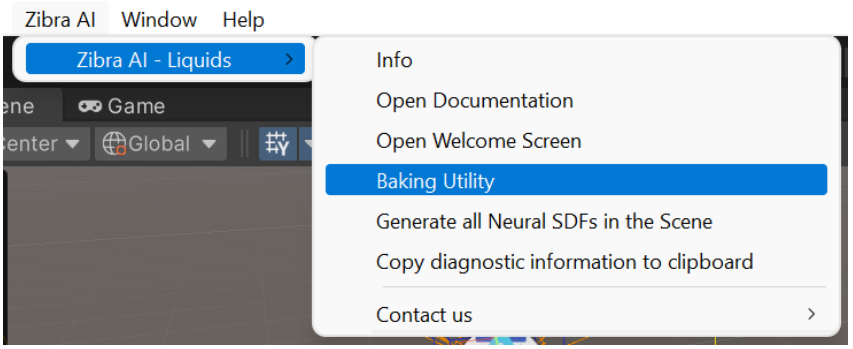


2.1.2. Press *“Open Zibra Liquids Baking Utility”*

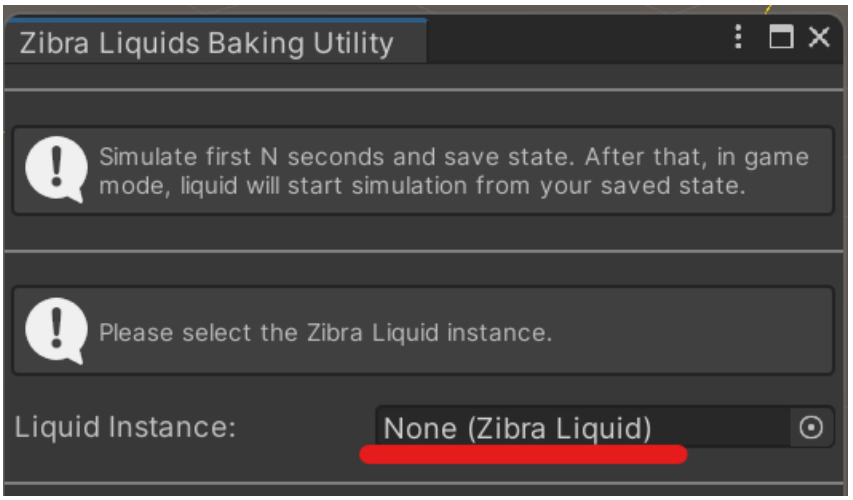


2.2. Via Unity’s menu

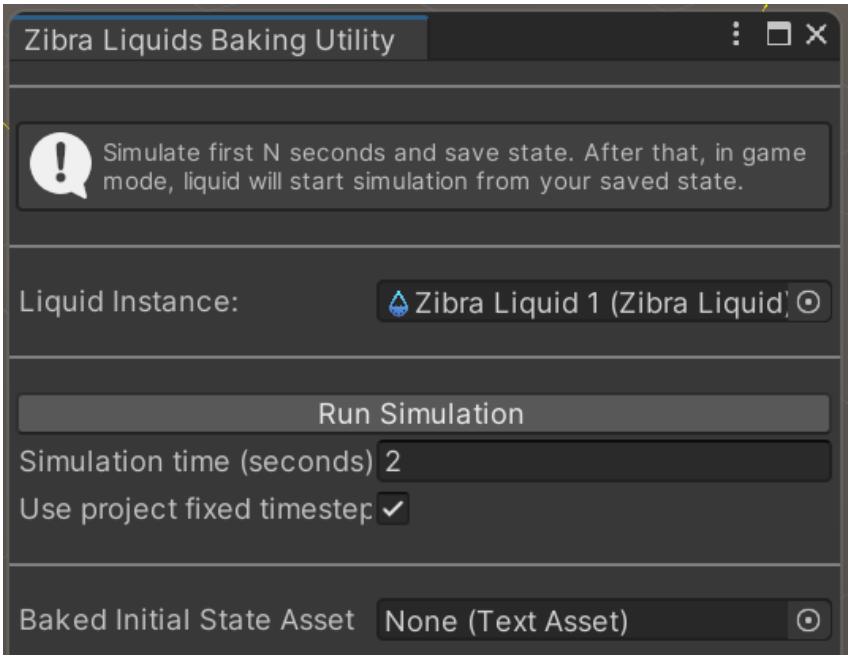
2.2.1. Press *“Zibra AI -> Zibra AI - Liquids -> Baking Utility”*



2.2.2. If the Zibra Liquid instance wasn’t selected automatically, select the Liquid Instance you want to bake.

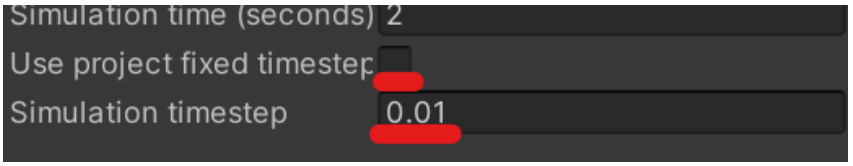


3. Now you see the Baking Utility window

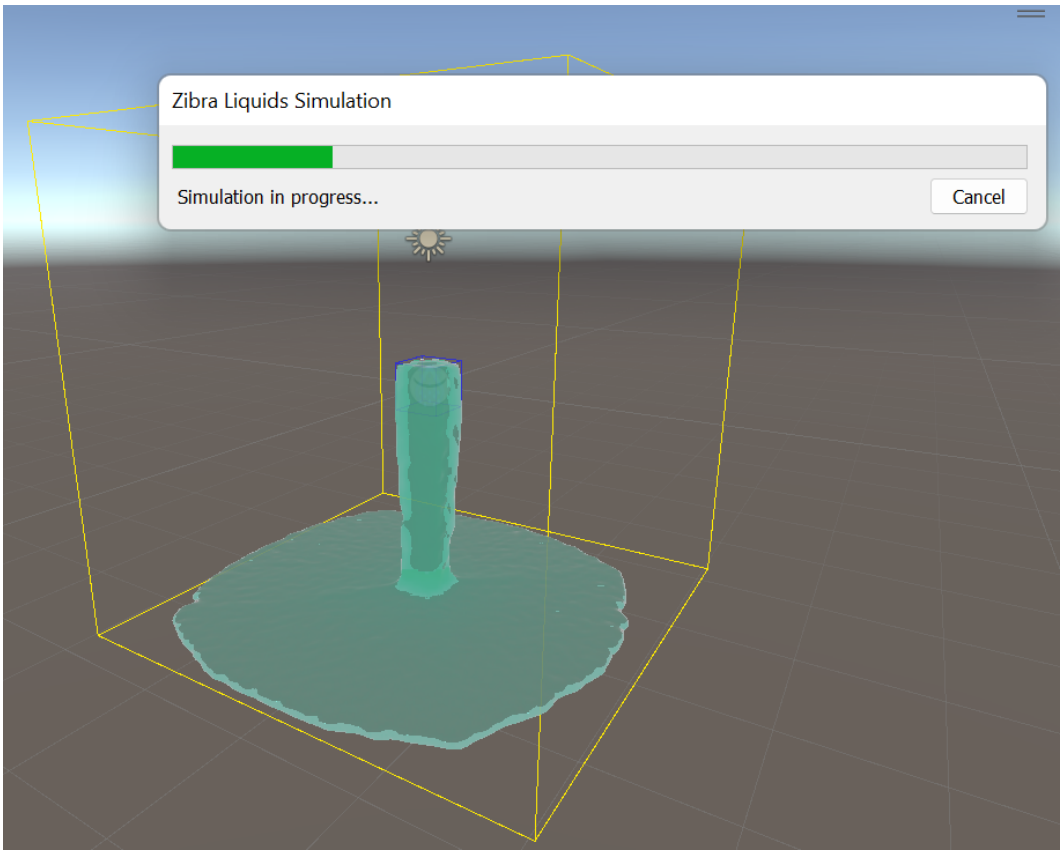


4. Set the Simulation time to the number of seconds you want to simulate.

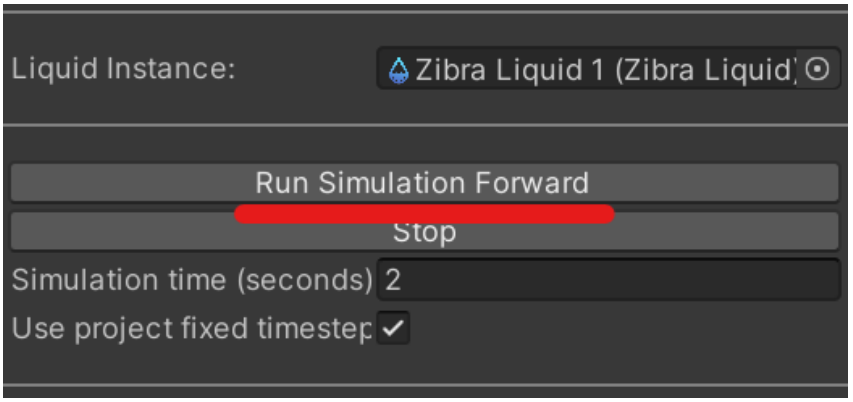
5. (optional) If you want to run a baking simulation with specific timestamp, you can disable “Use project fixed timestep” and set your desired “Simulation timestep”



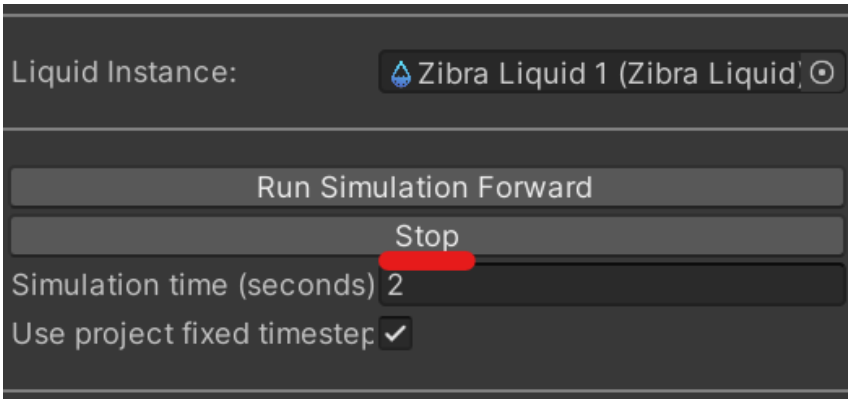
6. Press *“Run Simulation”.*
You’ll see the simulation running in Scene/Game view *(only Game view on URP)*



7. *(optional)* If you want to run the simulation further, set the Simulation time to the amount of time you want to run simulation forward, and press *“Run Simulation Forward”.*



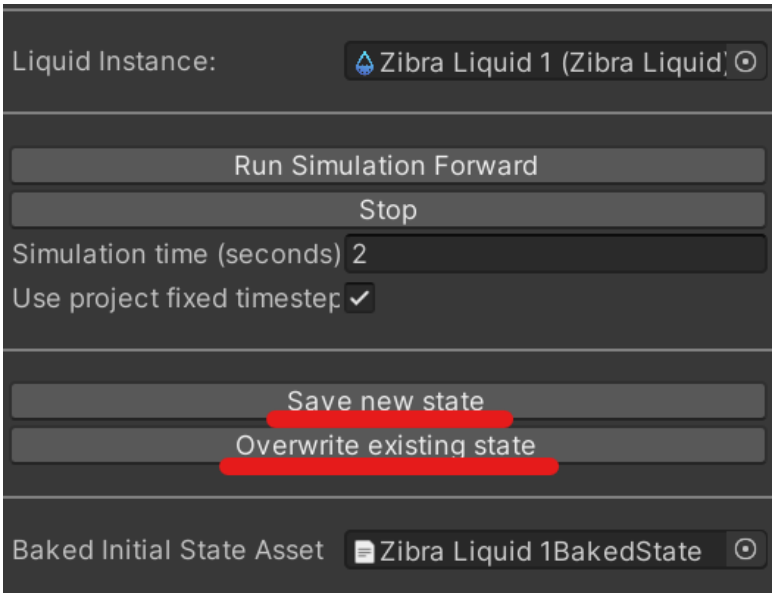
8. (optional) If you simulated too far, and want to start over, press *"Stop"* and go back to step 4



9. Press *"Save new state"* to save baked state.

That will save the new state to the file in the subdirectory next to your scene file. e.g. for Liquid Instance *"ZibraLiquid1"* in *"/Assets/NewScene.unity"* scene, liquid baked state will be saved into *"/Assets/NewScene/"* folder as *"ZibraLiquid1BakedState.bytes"*.

9.1. If you already had a previous saved state you'll have 2 buttons: *"Save new state"* and *"Overwrite existing state".*



9.2. "Save new state" always creates a new file, while "Overwrite existing state" always overwrites the old file with a new state.

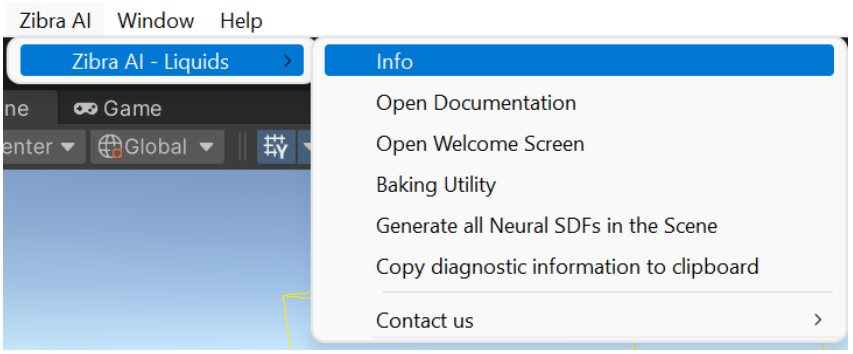
10. (optional) After you are done with baking, press "Stop" to stop the liquid simulation and unlock the parameters that are disabled when the liquid is "live". Next time you'll change a scene or enter playmode this state will reset, so you usually don't need to manually stop it.

Registering Zibra Liquids

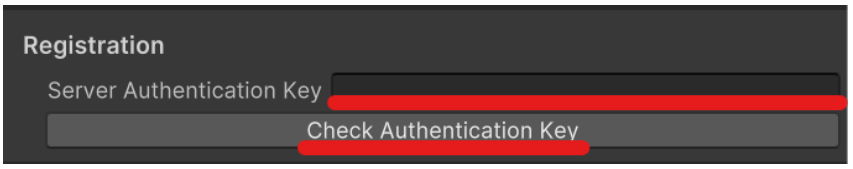
Access to Neural and Skinned Mesh Collider generation and usage of Zibra Liquids in the Editor requires registering.

To register your copy of Zibra Liquids:

1. Make sure that you launched the Unity Editor via Unity Hub and are logged into the correct Unity account.
2. Press *"Zibra AI -> Zibra AI - Liquids -> Info"*



3. Enter your key into the *"Server Authentication Key"* field and press Check Authentication Key.

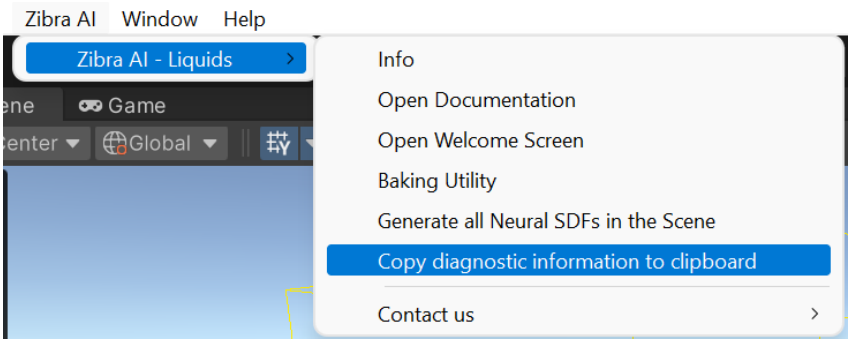


4. After successful registration you should see this message

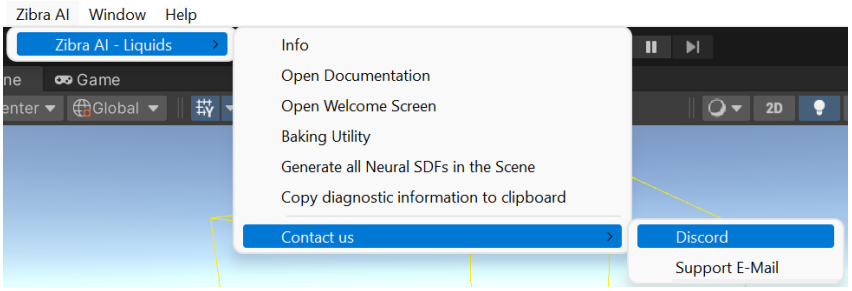
After registration you can generate neural and skinned mesh colliders and run liquid in the editor. Note that generation sends your mesh to the Zibra AI servers for processing.
Please note that the Pro version of the plugin will need to contact the server to validate your license on each startup.

Troubleshooting

If you have any issues or feedback feel free to contact us on Discord
If you are going to reach out to us for troubleshooting, please export the diagnostics info via this button and send it to us

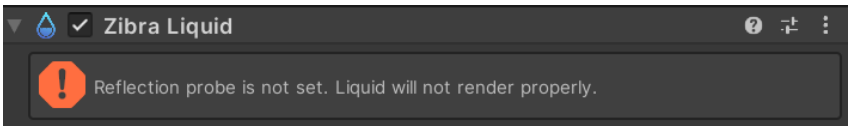


After copying that you can go directly to our discord or email with another buttons



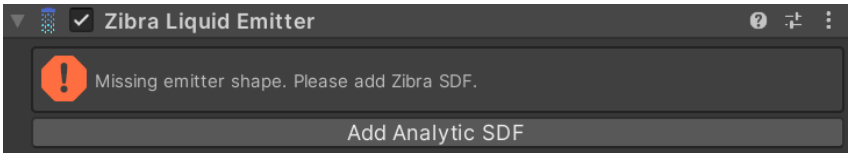
Some common issues you may have and ways to troubleshoot them:

- First thing you want to do in case of any error is to check your Console for any errors.
- Issue: Liquid doesn't work
  - Solution 1: Check whether you have any errors in the Liquid Instance e.g.



  - Solution 2: (In case of URP) Add "URP Liquid Rendering Component" and enable "Depth buffer"
    See Additional setup on URP
  - Solution 3: Double check System Requirements. We have unsupported platforms *(e.g. webGL)*, Unity version (anything before 2019.4.31) and some restrictions *(For VR to work you need to select Unity Render)*

- Issue: manipulators/colliders don't work
    - Solution: check their inspector and ensure all manipulators/colliders have an SDF component.
      e.g.



- Issue: On HDRP, the liquid is too bright
    - Solution: Decrease *“Reflection color*” in Liquid Instance’s Material Parameters